

## 1. Giới thiệu

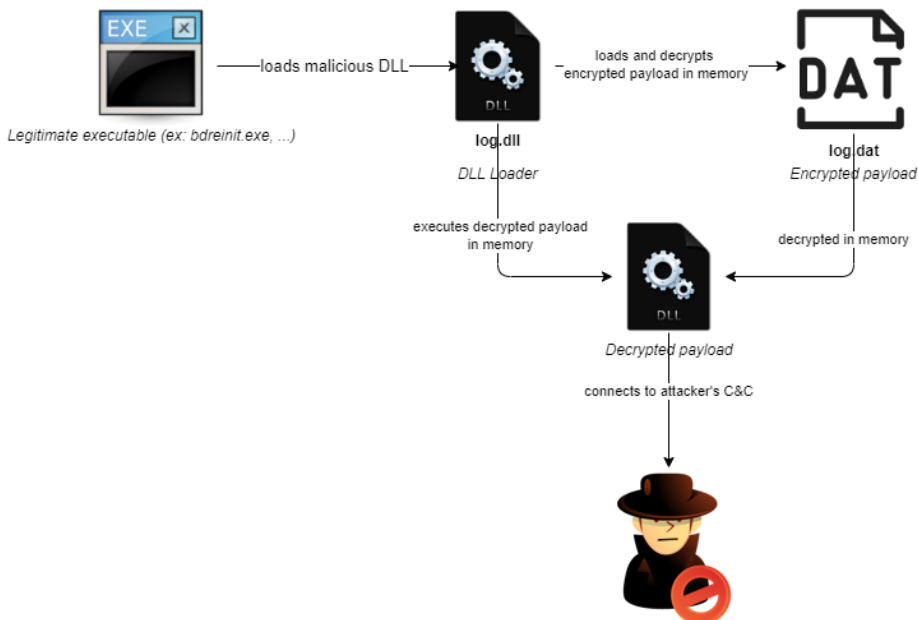
Tại VinCSS, chúng tôi liên tục chủ động theo dõi tình hình an ninh mạng, săn tìm các mẫu mã độc và đánh giá mức độ nguy hiểm của chúng, đặc biệt là các mẫu mã độc nhắm tới Việt Nam. Gần đây, trong quá trình thực hiện hunting trên nền tảng của [VirusTotal](#), thực hiện tìm kiếm các mẫu byte đặc trưng liên quan tới nhóm **Mustang Panda (PlugX)**, chúng tôi đã phát hiện một loạt mẫu mã độc mà chúng tôi nghi ngờ là của nhóm này được tải lên từ Việt Nam.

Tất cả các mẫu này đều có chung tên là "log.dll" và có tỉ lệ phát hiện khá thấp.

FILES 5 / 5		90 days	🔍	🔒	🔗	🔗	🔗	🔗	🔗
		Detections	Size	First seen	Last seen	Submitters			
<input type="checkbox"/>	D80C9D0A56A0338FA48C7280001F8ED2480545883282C2135887E89A56807721 log.dll	11 / 68	864.00 KB	2022-05-07 01:33:18	2022-05-07 01:33:18	1	🔗	🔗	🔗
<input type="checkbox"/>	84893F36D4C38B468F89E4804D507B96888892F76A7C251430CEBE58EC88DC5D 84893F36dac3bba6bf09ea04da5d7b9608b892f76a7c25143deeb50ecbbdc5d.sample	9 / 68	103.00 KB	2022-05-05 12:42:34	2022-05-05 17:58:50	2	🔗	🔗	🔗
<input type="checkbox"/>	3171285C4A8463689379688F538C48AEC980FE3280DE10CF022689122576F4E log.dll.sc	13 / 67	377.50 KB	2022-04-25 14:04:36	2022-04-25 14:04:36	1	🔗	🔗	🔗
<input type="checkbox"/>	6048202CBE5E97C7C8A74A12E1F88E843C08AE088E34DC68851889417C133A9 log.dll	13 / 69	52.00 KB	2022-04-12 02:36:42	2022-04-12 02:36:42	1	🔗	🔗	🔗
<input type="checkbox"/>	DA28E84F4A66C2561CE1B9E827C7C8E4B10AFE8EE3EFD82E3CC2110178C987A log.dll	10 / 55	576.50 KB	2022-03-26 13:16:05	2022-03-26 13:16:05	1	🔗	🔗	🔗

Dựa vào thông tin trên, chúng tôi cho rằng có khả năng mã độc đã được cài cắm vào một vài đơn vị ở Việt Nam, do đó chúng tôi quyết định phân tích các mẫu mã độc này. Trong quá trình phân tích, dựa vào các dấu hiệu tìm được, chúng tôi tiếp tục hunting các dữ kiện còn thiếu để bổ sung bức tranh đầy đủ hơn cho quá trình phân tích.

Để tiện hình dung, chúng tôi vẽ lại luồng thực thi của mã độc như sau:



Bài phân tích này của chúng tôi bao gồm:

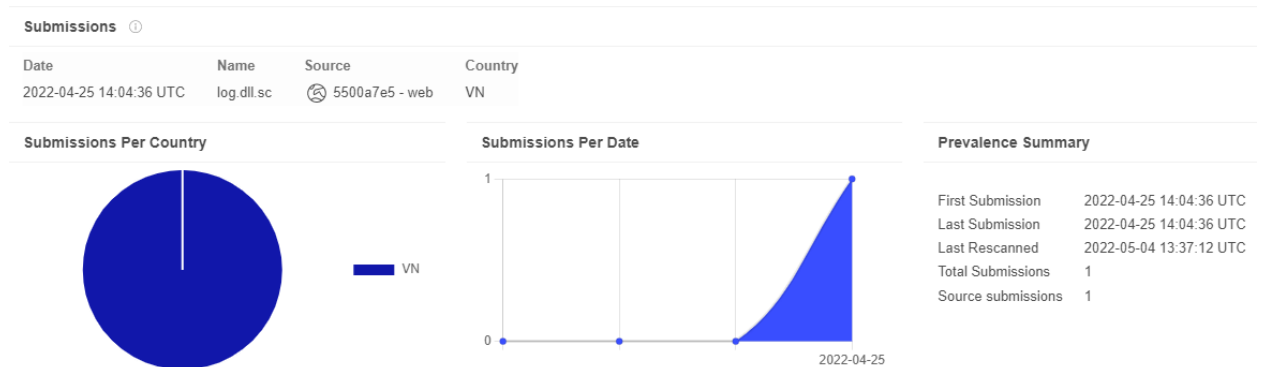
- ♦ Phân tích kỹ thuật của file log.dll.
- ♦ Phân tích kỹ thuật của shellcode được giải mã từ log.dat.

- ♦ Phân tích PlugX DLL cũng như giải mã thông tin cấu hình của PlugX.

## 2. Phân tích log.dll

Trong danh sách các mẫu hunt được ở trên, chúng tôi lựa chọn mẫu có hash: [3171285c4a846368937968bf53bc48ae5c980fe32b0de10cf0226b9122576f4e](https://www.virustotal.com/gui/sample/3171285c4a846368937968bf53bc48ae5c980fe32b0de10cf0226b9122576f4e)

Sample này được submit lên VirusTotal từ **Việt Nam** vào thời gian **2022-04-25 14:04:36 UTC**



Thông tin từ Rich Header cho thấy khả năng nó được compile bằng **Visual Studio 2012/2013**:

product-id (8)	build-id (4)
<a href="#">Implib1100</a>	Visual Studio 2012 - 11.0
<a href="#">Import</a>	Visual Studio
<a href="#">Utc1800_CPP</a>	Visual Studio 2013 - 12.0
<a href="#">Masm1200</a>	Visual Studio 2013 - 12.0
<a href="#">Utc1800_C</a>	Visual Studio 2013 - 12.0
<a href="#">Import (old)</a>	Visual Studio
<a href="#">Export1200</a>	Visual Studio 2013 - 12.0 RTM
<a href="#">Linker1200</a>	Visual Studio 2013 - 12.0 RTM

Kiểm tra các sections của file thì có thể thấy nó bị packed hoặc code bị 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 ...	□ ▲ □0□0 ...	Not packed - 16.6992 % ZERO

Sample có tên gốc là `ljAt.dll`, và nó export hai hàm là `LogFree` và `LogInit`:



Cũng tương tự như trên, code tại hàm LogInit\_0 cũng đã bị obfuscated hoàn toàn, phải mất rất lâu IDA mới decompile được code của hàm này:



```

4967 v200 = v196 ^ v199 ^ 0x77D9D620;
4968 v201 = ~v196 & (v199 ^ 0x882629DF) | (v199 ^ 0x77D9D620) & v196;
4969 v202 = ~((v199 ^ 0x77D9D620) & v200) & 0xF52388E | (v199 ^ 0x77D9D620) & v200 &
4970 v203 = v202 & (v202 ^ 0x89026167) & ~v202 & 0x89026167 | ~v202 & 0x89026167 ^ v20
4971 v204 = v203;
4972 v205 = ~v203 & 0xA13D01E2;
4973 v206 = (~v203 & 0xA21D4CC | v203 & 0xF5D2B33) ^ 0xF5D2B33;
4974 read_content_status = (v206 & ((v205 | v204 & 0x5EC2FE1D) ^ 0x5EC2FE1D) | (v205 |
4975 control_var = 0x09DD068D;
4976 vu786 = read_content_status;
4977 vu787 = dword_1005FE74 < 0xA;
4978 do
4979 {
4980 LABEL_17:
4981 while ( control_var <= (int)0xC28CF813 )
4982 {
4983 if ( control_var > (int)0xA34633B6 )
4984 {
4985 if ( control_var = 0xA34633B7 )
4986 {
4987 control_var = 0x70D8932E;
4988 goto LABEL_3;
4989 }
4990 (*decrypted_shellcode)(); // exec decrypted shell
4991 control_var = 0x8988A65F;
4992 }
4993 else
4994 {
4995 if ( control_var = 0x8988A65F )
4996 {
4997 (*decrypted_shellcode)();
4998 v2489 = dword_1005FE78 + (dword_1005FE78 - 1); // exec decrypted shell
4999 v2490 = ~v2489;
5000 v2491 = v2489 & ((dword_1005FE78 + (dword_1005FE78 - 1)) ^ 0x25430972);
5001 v2492 = ~(v2491 & ~v2489 & 0x25430972 | ~v2489 & 0x25430972 ^ v2491) & 0x

```

Nhiệm vụ chính của hàm LogInit\_0 là gọi tới hàm f\_read\_content\_of\_log\_dat\_file\_to\_buf để đọc nội dung của file log.dat và thực thi shellcode đã giải mã:

```

public LogInit
proc near
; DATA XREF: .rdata:off_1005D0B8+0
jmp LogInit_0 ; TAGS: ['Enum', 'FileWIN']
endp

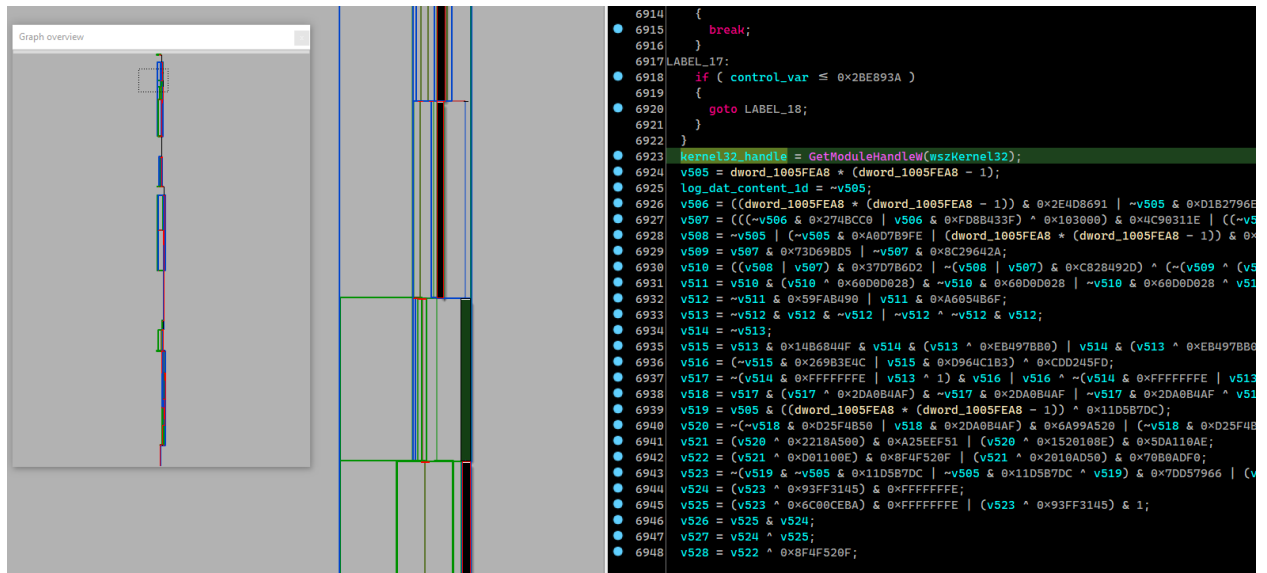
23 calls, 1 strings

calls:
- call dword ptr[ecx]
- call ds:CloseHandle ; call CloseHandle
- call ds:CreateFileA ; call CreateFileA to open file
- call ds:ReadFile ; call ReadFile to read file content
- call _strncmp ; call _strcmp to compare string
- 2 call dword ptr[ecx] ; exec decrypted payload/shellcode
- call ds:CloseHandle ; call CloseHandle
- call ds>DeleteFileA ; call DeleteFileA
- call ds:CloseHandle ; call CloseHandle
- call ds>DeleteFileA ; call DeleteFileA
- 1 call f_read_content_of_log_dat_file_to_buf ; call f_read_content_of_log_dat_file
- call ds:GetModuleHandleA ; call GetModuleHandleA to retrieve kernel32.dll handle
- call ds:GetProcAddress ; retrieve api address
- call ecx ; call API func
- call ds:ExpandEnvironmentStringsA ; call ExpandEnvironmentStringsA
- call ds:CreateFileA ; call CreateFileA for retrieving handle to create tmp file
- call _strlen ; call _strlen
- call ds:WriteFile ; call WriteFile to write content to file
- call ds:ExpandEnvironmentStringsA ; call ExpandEnvironmentStringsA
- call ds:CreateFileA ; call CreateFileA
- call _strlen ; call _strlen
- call ds:WriteFile ; call WriteFile
- call _security_check_cookie(x)

strings:
- kernel32

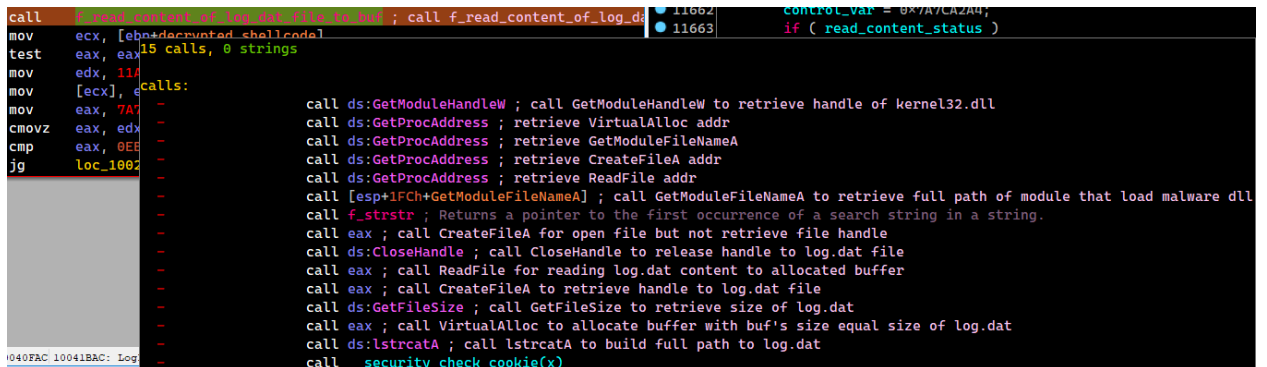
```

Code tại hàm f\_read\_content\_of\_log\_dat\_file\_to\_buf cũng bị obfuscated hoàn toàn:

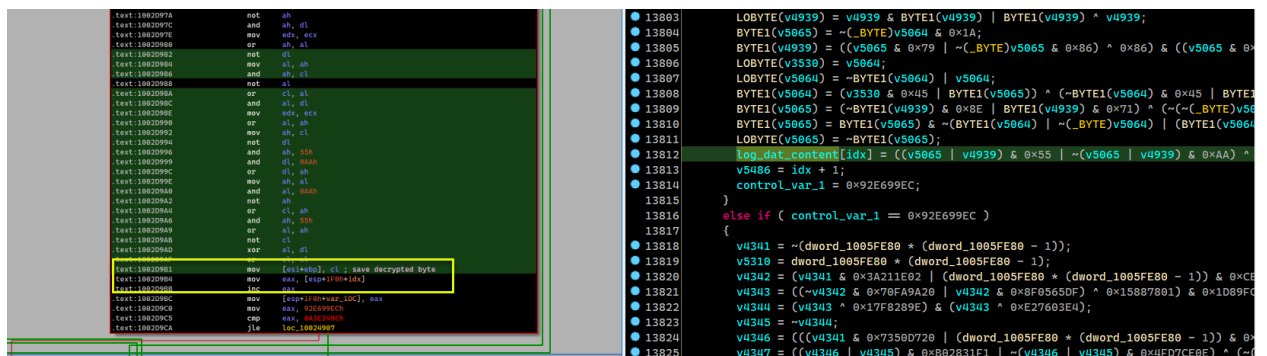


Nhiệm vụ chính của hàm này như sau:

- Gọi hàm `GetModuleHandleW` để lấy handle của `kernel32.dll`.
- Gọi hàm `GetProcAddress` để lấy địa chỉ của các hàm APIs: `VirtualAlloc`, `GetModuleFileNameA`, `CreateFileA`, `ReadFile`.
- Sử dụng các hàm APIs trên để lấy đường dẫn tới file `log.dat` và đọc nội dung của file này vào vùng nhớ đã được cấp phát.



- Thực hiện giải mã nội dung của `log.dat` thành shellcode để sau đó shellcode này được thực thi bởi lời gọi từ hàm `LogInit_0`.



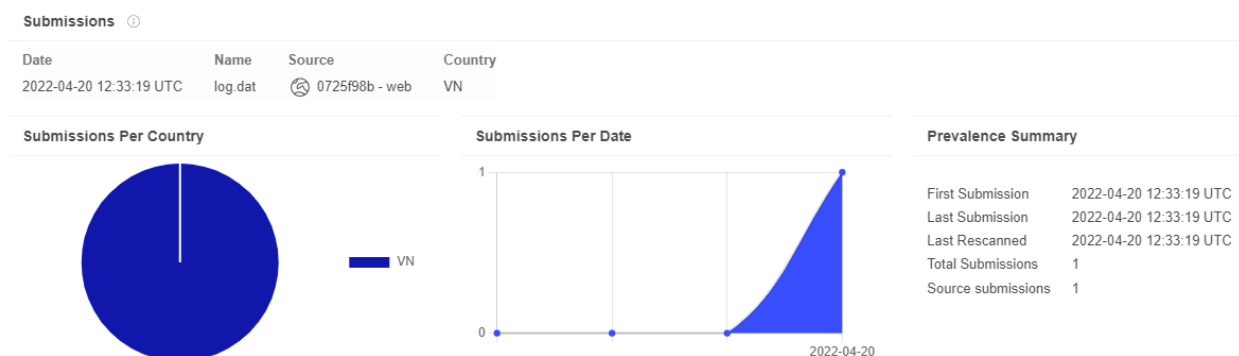
### 3. Phân tích shellcode

Căn cứ vào thông tin đã phân tích ở trên, chúng tôi biết được file `log.dll` sẽ đọc nội dung từ file `log.dat` và giải mã ra shellcode để thực thi tiếp. Dựa vào thông tin này, chúng tôi tiếp tục hunting file `log.dat` trên VirusTotal với phạm vi giới hạn nguồn submit là từ Việt Nam.

Kết quả có được như sau:

FILES 4 / 4						90 days	🔍	🔗	🔒	🔧	📄	📁
		Detections	Size	First seen	Last seen	Submitters						
<input type="checkbox"/>	3268DC1CD5C629289DF16B120E22F601A7642A85628882C4715FE2B9FBC19EB0 🔍 🌐 📄 log.dat	0 / 57	194.66 KB	2022-05-07 01:32:51	2022-05-07 01:32:51	1						
<input type="checkbox"/>	02A9B3BEAA34A754E2788E8F7038AAF2B9C633A680BF77188284B7330FA0C5 🔍 🌐 📄 log.dat	2 / 59	189.23 KB	2022-05-05 12:44:31	2022-05-05 12:44:31	1						
<input type="checkbox"/>	0E9E270244371A51FBB0991EE246EF34775787132822085D40C99F10817539C0 🔍 🌐 📄 log.dat.sc	0 / 57	194.66 KB	2022-04-25 14:07:46	2022-04-25 14:07:46	1						
<input type="checkbox"/>	20E77804E2B09B843A826F194389C2605CFC17FD2FAFDE1B8EB2F819FC6C0C84 🔍 🌐 📄 log.dat	0 / 57	194.66 KB	2022-04-20 12:33:19	2022-04-20 12:33:19	1						

Với kết quả trên, tại thời điểm phân tích, chúng tôi lựa chọn file `log.dat` ([2de77804e2bd9b843a826f194389c2605cfc17fd2fafde1b8eb2f819fc6c0c84](#)) được submit lên VirusTotal vào thời gian **2022-04-20 12:33:19 UTC** (tức là trước 5 ngày so với file `log.dll` ở trên).



Tiến hành debug và dump ra shellcode đã được giải mã:

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	<òùè....^%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	''''''KKKKKKKKKK
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	%''''''nnnnnnnnnn
000000B0	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
000000C0	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
000000D0	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
000000E0	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
000000F0	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
00000100	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
00000110	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
00000120	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
00000130	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
00000140	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
00000150	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
00000160	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
00000170	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn
00000180	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	B5	nnnnnnnnnnnnnnnn

Sử dụng hai công cụ là [FLOSS](#) và [scdbg](#) để có cái nhìn tổng quan về shellcode này. Kết quả có được như sau:

```

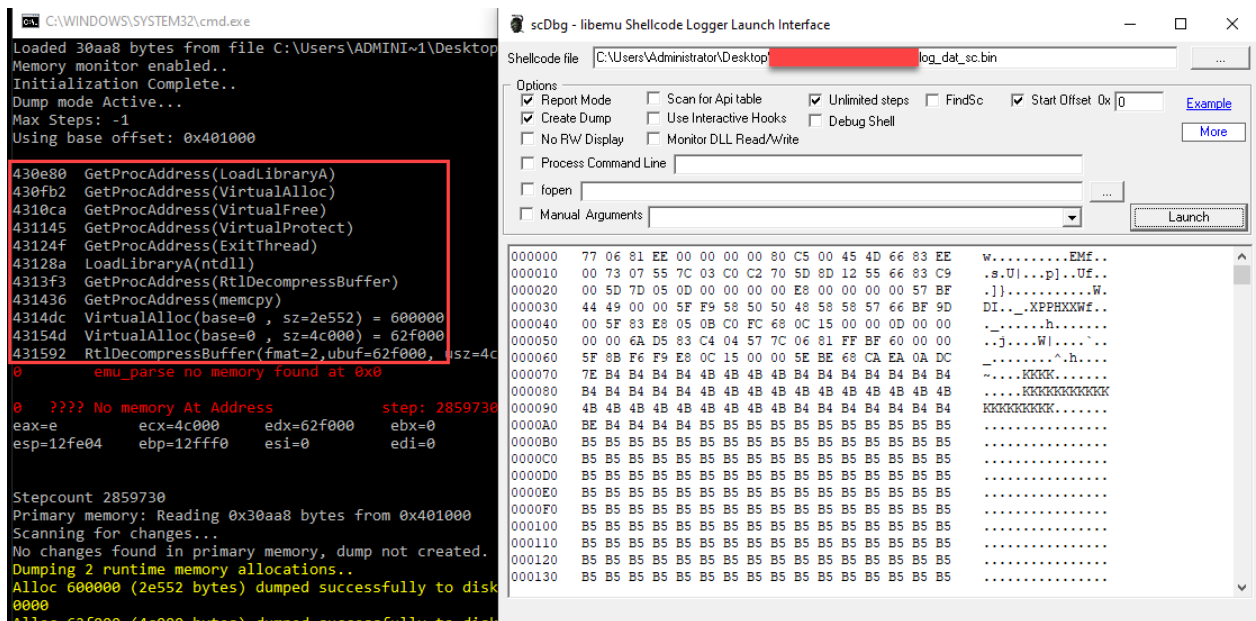
FLOSS static Unicode strings

FLOSS decoded 2 strings
(EAA
&EAA

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

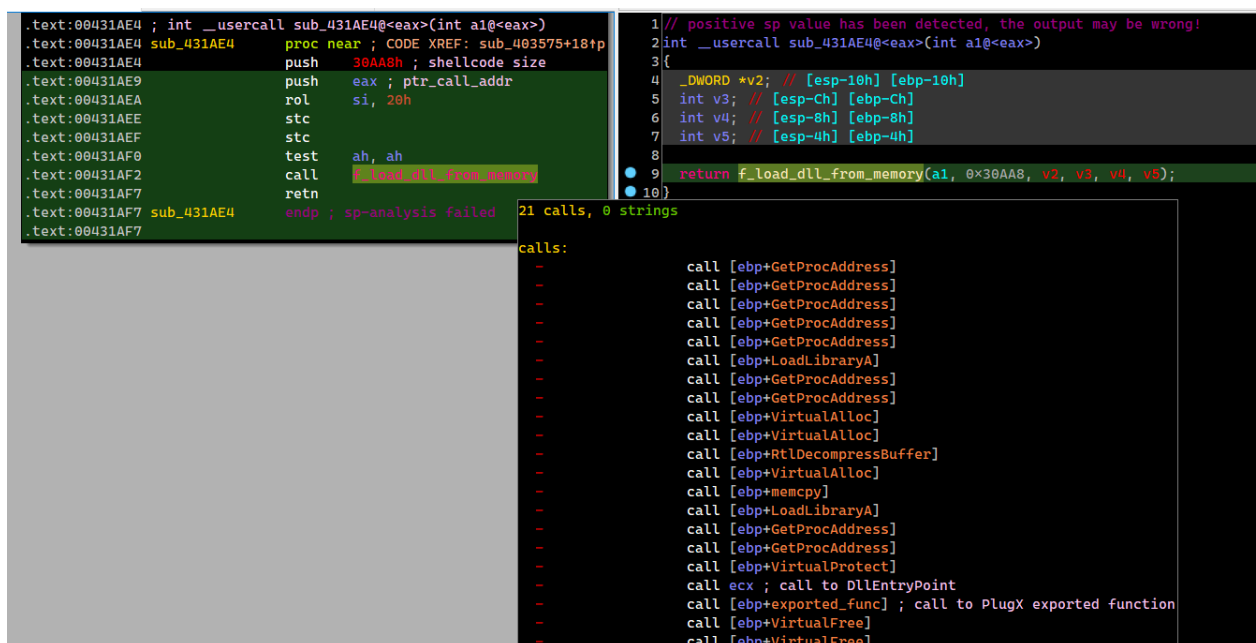
```





Với các kết quả có được ở trên, có thể thấy shellcode này sẽ thực hiện cấp phát vùng nhớ và sau đó gọi hàm `RtlDecompressBuffer` để giải nén dữ liệu với tham số truyền cho hàm là `COMPRESSION_FORMAT_LZNT1`.

Tiếp tục sử dụng IDA để phân tích shellcode này, nhiệm vụ chính của nó là giải nén ra một DLL và gọi tới hàm được export của DLL này để thực thi. Hàm thực hiện nhiệm vụ này được tôi đặt tên là `f_load_dll_from_memory`:



Code tại hàm này đầu tiên sẽ lấy ra địa chỉ base của `kernel32.dll` dựa vào giá trị hash được tính toán trước là `0x6A4ABC5B`. Giá trị hash này cũng đã từng được chúng tôi đề cập trong [bài phân tích này](#).



```

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

Tiếp theo sẽ lấy địa chỉ hàm API 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;

```

Sau đó bằng kĩ thuật [stackstring](#), shellcode cấu thành tên các hàm APIs và lấy địa chỉ của các hàm sau:

```

.text:00401E99 loc_401E99: ; CODE XREF: f_load_dll_from_memory+395j
.text:00401E99 mov     [ebp+var_4], 0
.text:00401E9B mov     edx, [ebp+var_4]
.text:00401E9D mov     [ebp+edx+szVirtualAlloc], 'V'; VirtualAlloc
.text:00401E9F mov     eax, [ebp+var_4]
.text:00401EA1 mov     [ebp+var_4], eax
.text:00401EA3 mov     ecx, [ebp+var_4]
.text:00401EA5 mov     [ebp+ecx+szVirtualAlloc], 't'
.text:00401EA7 mov     edx, [ebp+var_4]
.text:00401EA9 add     edx, 1
.text:00401EAB mov     [ebp+var_4], edx
.text:00401EAD mov     eax, [ebp+var_4]
.text:00401EAF mov     [ebp+eax+szVirtualAlloc], 'r'
.text:00401EB1 mov     ecx, [ebp+var_4]
.text:00401EB3 add     ecx, 1
.text:00401EB5 mov     [ebp+var_4], ecx
.text:00401EB7 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401EB9 mov     eax, [ebp+var_4]
.text:00401EBB add     eax, 1
.text:00401EBD mov     [ebp+var_4], eax
.text:00401EBF mov     [ebp+ecx+szVirtualAlloc], 'u'
.text:00401EC1 mov     edx, [ebp+var_4]
.text:00401EC3 add     edx, 1
.text:00401EC5 mov     [ebp+var_4], edx
.text:00401EC7 mov     [ebp+ecx+szVirtualAlloc], 't'
.text:00401EC9 mov     eax, [ebp+var_4]
.text:00401ECB add     eax, 1
.text:00401ECD mov     [ebp+var_4], eax
.text:00401ECF mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401ED1 mov     eax, [ebp+var_4]
.text:00401ED3 add     eax, 1
.text:00401ED5 mov     [ebp+var_4], eax
.text:00401ED7 mov     [ebp+ecx+szVirtualAlloc], 'u'
.text:00401ED9 mov     edx, [ebp+var_4]
.text:00401EDB add     edx, 1
.text:00401ED5 mov     [ebp+var_4], edx
.text:00401EF1 mov     [ebp+eax+szVirtualAlloc], 't'
.text:00401EF3 mov     ecx, [ebp+var_4]
.text:00401EF5 add     ecx, 1
.text:00401EF7 mov     [ebp+var_4], ecx
.text:00401EF9 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401EFB mov     eax, [ebp+var_4]
.text:00401EFD add     eax, 1
.text:00401EFF mov     [ebp+var_4], eax
.text:00401F01 mov     [ebp+ecx+szVirtualAlloc], 't'
.text:00401F03 mov     ecx, [ebp+var_4]
.text:00401F05 add     ecx, 1
.text:00401F07 mov     [ebp+var_4], ecx
.text:00401F09 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F0B mov     eax, [ebp+var_4]
.text:00401F0D add     eax, 1
.text:00401F0F mov     [ebp+var_4], eax
.text:00401F11 mov     [ebp+ecx+szVirtualAlloc], 't'
.text:00401F13 mov     ecx, [ebp+var_4]
.text:00401F15 add     ecx, 1
.text:00401F17 mov     [ebp+var_4], ecx
.text:00401F19 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F1B mov     eax, [ebp+var_4]
.text:00401F1D add     eax, 1
.text:00401F1F mov     [ebp+var_4], eax
.text:00401F21 mov     [ebp+ecx+szVirtualAlloc], 't'
.text:00401F23 mov     ecx, [ebp+var_4]
.text:00401F25 add     ecx, 1
.text:00401F27 mov     [ebp+var_4], ecx
.text:00401F29 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F2B mov     eax, [ebp+var_4]
.text:00401F2D add     eax, 1
.text:00401F2F mov     [ebp+var_4], eax
.text:00401F31 mov     [ebp+ecx+szVirtualAlloc], 't'
.text:00401F33 mov     ecx, [ebp+var_4]
.text:00401F35 add     ecx, 1
.text:00401F37 mov     [ebp+var_4], ecx
.text:00401F39 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F3B mov     eax, [ebp+var_4]
.text:00401F3D add     eax, 1
.text:00401F3F mov     [ebp+var_4], eax
.text:00401F41 mov     [ebp+ecx+szVirtualAlloc], 't'
.text:00401F43 mov     ecx, [ebp+var_4]
.text:00401F45 add     ecx, 1
.text:00401F47 mov     [ebp+var_4], ecx
.text:00401F49 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F4B mov     eax, [ebp+var_4]
.text:00401F4D add     eax, 1
.text:00401F4F mov     [ebp+var_4], eax
.text:00401F51 mov     [ebp+ecx+szVirtualAlloc], 't'
.text:00401F53 mov     ecx, [ebp+var_4]
.text:00401F55 add     ecx, 1
.text:00401F57 mov     [ebp+var_4], ecx
.text:00401F59 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F5B mov     eax, [ebp+var_4]
.text:00401F5D add     eax, 1
.text:00401F5F mov     [ebp+var_4], eax
.text:00401F61 mov     [ebp+ecx+szVirtualAlloc], 't'
.text:00401F63 mov     ecx, [ebp+var_4]
.text:00401F65 add     ecx, 1
.text:00401F67 mov     [ebp+var_4], ecx
.text:00401F69 mov     [ebp+edx+szVirtualAlloc], 't'
.text:00401F6B mov     eax, [ebp+var_4]
.text:00401F6D 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

Tiếp theo, shellcode thực hiện cấp phát vùng nhớ (compressed\_buf) có kích thước 0x2E552, sau đó đọc dữ liệu từ offset 0x1592 (on disk) và thực hiện vòng lặp xor với key là 0x72 để điền dữ liệu vào compressed\_buf. Trên thực tế, kích thước của compressed\_buf là 0x2E542, tuy nhiên 16 bytes đầu của nó được dùng để lưu thông tin về signature, uncompressed\_size, compressed\_size nên được cộng thêm 0x10.

Shellcode tiếp tục cấp phát vùng nhớ (uncompressed\_buf) có kích thước là 0x4C000 và gọi hàm RtlDecompressBuffer để giải nén dữ liệu tại compressed\_buf vào uncompressed\_buf với định dạng nén là 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 )


```

Dựa vào kết quả phân tích ở trên, dễ dàng có được file DLL đã giải nén (tuy nhiên file này đã bị hủy thông tin header):

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	wsWfhSkErKwhUlRx
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...0f.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	10	00	00	..<.....
00000110	00	10	00	00	00	00	00	10	00	10	00	00	00	02	00	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	40	01	00	..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	00	.....E...
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	00	.....Pz..@...
000001B0	00	00	00	00	00	00	00	00	00	90	04	00	30	01	00	00	.....0...
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	00	......text...
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	90	04	00	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...€...

Sửa lại thông tin của header và kiểm tra bằng [PE-bear](#), DLL này có tên gốc là RFPmzNfQQFPXX và chỉ export một hàm duy nhất là 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		

Quay trở lại với shellcode, sau khi giải nén ra được DLL trên memory, nó sẽ tiến hành nhiệm vụ của một loader thực hiện mapping DLL này vào một vùng nhớ mới. Sau đó, gọi tới hàm mà DLL này export (ở đây là hàm Main) để thực thi nhiệm vụ chính:

```

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 Ră..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;

```

**Lưu ý:** Tại thời điểm phân tích shellcode này, chúng tôi vẫn chưa khẳng định nó là một biến thể của mã độc PlugX, mà chỉ đặt ra câu hỏi nghi ngờ. Chỉ tới khi phân tích DLL được giải nén ở trên, chúng tôi mới khẳng định chắc chắn đây là biến thể của PlugX và đặt lại tên các trường trong struct cho tường minh lại như trên.

#### 4. Phân tích DLL đã giải nén

Chúng tôi sẽ không đi sâu vào phân tích chi tiết DLL này mà chỉ cung cấp những thông tin cần thiết để chứng minh đây là biến thể của PlugX cũng như quá trình giải mã thông tin cấu hình mà mã độc sẽ sử dụng.

##### 4.1. Cách PlugX gọi hàm API

Ở biến thể này, thông tin về các hàm API được lưu tại các xmmword, sau đó được nạp vào thanh ghi xmm0 (128-bit), phần còn thiếu của tên hàm sẽ được nạp thông qua stack. Mã độc thực hiện lấy handle của DLL tương ứng tới các hàm API này, sau đó sử dụng GetProcAddress để lấy ra địa chỉ hàm API cần dùng:

```

.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. Tạo thread chính để thực thi

Mã độc thực hiện điều chỉnh các token SeDebugPrivilege và SeTcbPrivilege cho chính tiến trình của nó nhằm có được quyền truy cập đầy đủ tới các tiến trình hệ thống. Sau đó, nó tạo một thread chính được đặt tên là "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. Giao tiếp với C2

Mã độc có thể giao tiếp với C2 thông qua các giao thức TCP, HTTP hoặc UDP:

```
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. Các lệnh mã độc thực thi

Mã độc sẽ nhận lệnh từ kẻ tấn công để thực thi các chức năng tương ứng liên quan tới Disk, Network, Process, Registry, v.v...

```
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;
```

Dưới đây là bảng tổng kết danh sách toàn bộ cách lệnh mà kẻ tấn công thực hiện thông qua mẫu mã độc này:

Nhóm lệnh	Lệnh con	Mô tả
Disk	0x3000	Lấy thông tin các ổ đĩa

	0x3001	Tìm file
	0x3002	Tìm file đệ quy
	0x3004	Đọc nội dung file
	0x3007	Ghi nội dung vào file
	0x300A	Tạo thư mục
	0x300C	Tạo tiến trình trình trên màn hình ẩn
	0x300D	Thực hiện sao chép, di chuyển, đổi tên hoặc xóa file
	0x300E	Thay thế tham số biến môi trường bằng giá trị chỉ định
Nethood	0xA000	Liệt kê các tài nguyên mạng
Netstat	0xD000	Thu thập thông tin kết nối TCP
	0xD001	Thu thập thông tin kết nối UDP
	0xD002	Thiết đặt trạng thái của kết nối TCP
Option	0x2000	Khóa màn hình của máy
	0x2001	Tắt máy ngay lập tức
	0x2002	Khởi động lại máy
	0x2003	Tắt máy an toàn
	0x2005	Hiển thị thông báo
PortMap	0xB000	Thực hiện port mapping
Process	0x5000	Liệt kê danh sách các tiến trình
	0x5001	Liệt kê danh sách các modules
	0x5002	Tắt tiến trình được chỉ định
RegEdit	0x9000	Thu thập thông tin registry
	0x9001	Tạo registry
	0x9002	Xóa registry



	0x9003	Sao chép registry
	0x9004	Liệt kê thông tin các giá trị của một key cụ thể
	0x9005	Thiết lập giá trị cho một key cụ thể
	0x9006	Xóa giá trị của một key cụ thể
	0x9007	Lấy giá trị của một key
Service	0x6000	Thu thập cấu hình của service
	0x6001	Thay đổi các thông số cấu hình service
	0x6002	Khởi chạy service
	0x6003	Gửi mã điều khiển tới service
	0x6004	Xóa service
Shell	0x7002	Tạo pipe và thực thi command line
SQL	0xC000	Lấy thông tin nguồn dữ liệu
	0xC001	Thu thập thông tin trình điều khiển
	0xC002	Thực thi câu lệnh SQL
Telnet	0x7100	Thiết lập telnet server
Screen	0x4000	Mô phỏng hoạt động trên giao thức RDP
	0x4100	Chụp ảnh màn hình
KeyLog	0xE000	Thực hiện chức năng của key logger, lưu vào file "%allusersprofile%\MSDN\6.0\USER.DAT"

#### ***4.5. Giải mã thông tin cấu hình của PlugX***

Như đã phân tích ở trên, mã độc sẽ kết nối tới địa chỉ C2 thông qua các giao thức HTTP, TCP hoặc UDP tùy vào cấu hình được chỉ định. Vậy cấu hình này được lưu ở đâu? Với những mẫu cũ mà chúng tôi đã từng phân tích ([1](#), [2](#), [3](#), [4](#)) thì cấu hình của PlugX thường được lưu tại section .data với độ lớn là 0x724 (1828) bytes.

```
f_MemCpy(&spMalConfig, &encoded_config_data, 0x724u);
result = f_memcmp(&spMalConfig, "XXXXXXXX", 8u);
if ( result )
{
    // 123456789
    strcpy(xor_key, "123456789");
    xor_key_len = f_lstrlenA(xor_key);
    result = f_XorDecode(&spMalConfig, 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
```

Quay trở lại với mẫu đang phân tích, chúng tôi thấy rằng trước bước thực hiện kiểm tra các tham số truyền vào khi mã độc thực thi, nó sẽ gọi tới hàm thực hiện nhiệm vụ giải mã cấu hình:

```
ptr_cmd_line = GetCommandLineW();
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

Đi sâu vào hàm này, kết hợp với việc debug thêm từ shellcode, đặt lại tên các trường trong struct đã tạo, chúng tôi có được thông tin:

- ◆ Cấu hình của PlugX được nhúng trong shellcode và bắt đầu từ offset 0x69.
- ◆ Độ lớn của cấu hình là 0x0150C (5388) bytes.

- ◆ Key giải mã là 0xB4.

```
plugh_mapped_dll->signature = 0;
plugh_decrypted_dll->ptr_shellcode_base = ptr_call_addr; // 00402029 E8 00 00 00 00
plugh_decrypted_dll->shellcode_size = end_sc_offset;
plugh_decrypted_dll->ptr_encrypted_PlugX = ptr_enc_compressed_dll_addr; // 00403592 1C 9B ....
plugh_decrypted_dll->encrypted_PlugX_size = compressed_dll_size; // 0x2E552
plugh_decrypted_dll->PlugX_config = config; // 0x0402069 (offset 0x69 on disk)
plugh_decrypted_dll->PlugX_config_size = config_size; // 0x0150C
plugh_decrypted_dll->ptr_PlugX_entry_point = plugh_mapped_dll + payload_nt_headers->OptionalHeader.AddressOfEntryPoint;
VirtualProtect(lpAddress, payload_raw_size, PAGE_EXECUTE_READWRITE, &fOldProtect);
if ( (plugh_decrypted_dll->ptr_PlugX_entry_point->Plugh_mapped_dll, 1, 0) )
    return 0x15;
if ( ExportProc )
    ExportProc(); // execute export function
return 0;

PlughX mapped dll base = f_create_unnamed_event(0, &dll_base;
ptr_plugh_config = PlughX_mapped_dll_base->PlughX_config;
signature = ptr_plugh_config->signature; // 0xKA68BE5E
if ( ptr_plugh_config->signature == ptr_plugh_config->compressed_value )
    goto setup_config_buffer;
if ( PlughX_mapped_dll_base->PlughX_config_size != 0x150C )
    goto setup_config_buffer;
sub_10007D00(0);
xor_key = signature + 0x56; // 0xKA68BE5E + 0x86865656 = 0x50FE14B4
// xor_key = 0xB4
i = 0;
while ( i < 0x150C )
{
    ptr_decrypted_config = &decrypted_config[i++];
    *ptr_decrypted_config = xor_key * ptr_plugh_config - decrypted_config;
}
while ( i < 0x150C )
{
    if ( ptr_plugh_config->signature != signature_computed )
    {
        setup_config_buffer;
        fmemset_ret(g_std_decrypt_data, 0, 0x150C);
        result = 0;
    }
    else
    {
        fdecrypt_embedded_config_on_from_file_and_copy_to_mem
    }
}
```

Với toàn bộ thông tin đầy đủ ở trên, hoàn toàn có thể giải mã được thông tin cấu hình một cách dễ dàng:

IP	Port
86.78.23.152	53
86.78.23.152	22
86.78.23.152	8080
86.78.23.152	23

[illegible]

Ngoài danh sách các địa chỉ C2 như trên, còn có thêm thông tin liên quan đến thư mục được tạo trên máy nạn nhân để chứa các file của mã độc cũng như tên của service có thể được tạo:

// "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

Để dễ dàng hơn, tôi đã viết một python script để tự động trích xuất thông tin cấu hình cho biến thể này. Kết quả sau khi chạy script như sau:

```
$ 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. Kết luận

Các nhà nghiên cứu của CrowdStrike lần đầu tiên công bố thông tin về Mustang Panda vào tháng 6 năm 2018, sau khoảng một năm quan sát các hoạt động của nhóm này. Tuy nhiên, theo nghiên cứu và tổng hợp từ nhiều công ty an ninh mạng khác nhau thì nhóm APT này đã tồn tại trong hơn một thập kỷ với các biến thể khác nhau được tìm thấy trên khắp thế giới. Mustang Panda, được cho là nhóm tin tặc có trụ sở hoạt động tại Trung Quốc, được đánh giá là một trong những nhóm APT có động cơ cao, áp dụng kĩ thuật tinh vi để lây nhiễm và cài cắm mã độc, mục tiêu có được quyền truy cập vào máy của nạn nhân để từ đó thực hiện các hoạt động gián điệp và đánh cắp thông tin.

Trong bài viết này, chúng tôi đã phân tích các kĩ thuật mà dòng PlugX RAT áp dụng để thực thi và tránh bị phát hiện. Qua đó, có thể thấy nhóm APT này vẫn đang hoạt động và liên tục tìm cách cải tiến kĩ thuật. VinCSS sẽ tiếp tục tìm kiếm các mẫu và biến thể khác có thể được liên kết với biến thể mà chúng tôi phân tích trong bài viết này.

## 6. Tài liệu tham khảo

- ♦ [\[RE012-1\] 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. Tổng hợp IOCs

log.dll - db0c90da56ad338fa48c720d001f8ed240d545b032b2c2135b87eb9a56b07721
log.dll - 84893f36dac3bba6bf09ea04da5d7b9608b892f76a7c25143deeb50ecbbdc5d
log.dll - 3171285c4a846368937968bf53bc48ae5c980fe32b0de10cf0226b9122576f4e
log.dll - da28eb4f4a66c2561ce1b9e827cb7c0e4b10afe0ee3efd82e3cc2110178c9b7a
log.dat - 2de77804e2bd9b843a826f194389c2605cfc17fd2fafde1b8eb2f819fc6c0c84 Decrypted config: [+] 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
log.dat - 0e9e270244371a51fbb0991ee246ef34775787132822d85da0c99f10b17539c0 Decrypted config: [+] Folder name: %ProgramFiles%\BitDefender Update [+] Service name: BitDefender Crash Handler [+] Proto info: HTTP:// [+] C2 servers: 86.79.75.55:80 86.79.75.55:53 86.79.75.46:80 86.79.75.46:53 [+] Campaign ID: 1234
log.dat - 3268dc1cd5c629209df16b120e22f601a7642a85628b82c4715fe2b9fbc19eb0 Decrypted config: [+] Folder name: %ProgramFiles%\Common Files\ARO 2012 [+] Service name: BitDefender Crash Handler [+] Proto info: HTTP:// [+] C2 servers: 86.78.23.152:23 86.78.23.152:22 86.78.23.152:8080 86.78.23.152:53 [+] Campaign ID: 1234

log.dat - 02a9b3beaa34a75a4e2788e0f7038aaf2b9c633a6bdbfe771882b4b7330fa0c5 (THOR PlugX variant)

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

Dang Dinh Phuong – Threat Hunter

Tran Trung Kien (aka m4n0w4r) - Malware Analysis Expert

R&D Center - VinCSS (a member of Vingroup)