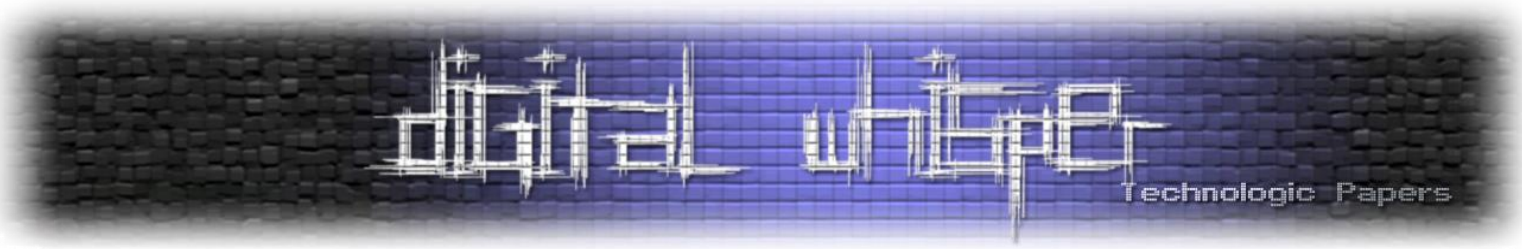


Solving FireEye FLARE #5

מאת שמואל ירוחם (sub)

מבוא

בשנת 2014 הוקם צוות בשם FLARE (FireEye Labs Advanced Reverse Engineering). הצוות פרסם 7 אתגרים (חלקם קשורים לניתוח בינארים). מטרת פרסום אתגרים אלו הייתה מציאת אנשים בעלי כישרונות בתחום ה-RE. מכל אתגר ניתן להוציא את הסיסמא בצורה שונה, הסיסמא היא כתובת מייל שאליה אפשר לשלוח קו"ח.

במאמר זה נפתור את האתגר החמישי. את האתגרים ניתן להוריד מכאן.

בנוסף כדי להבין לעומק את הפונקציות עליהם נדבר במאמר מומלץ להיעזר במקורות אלו:

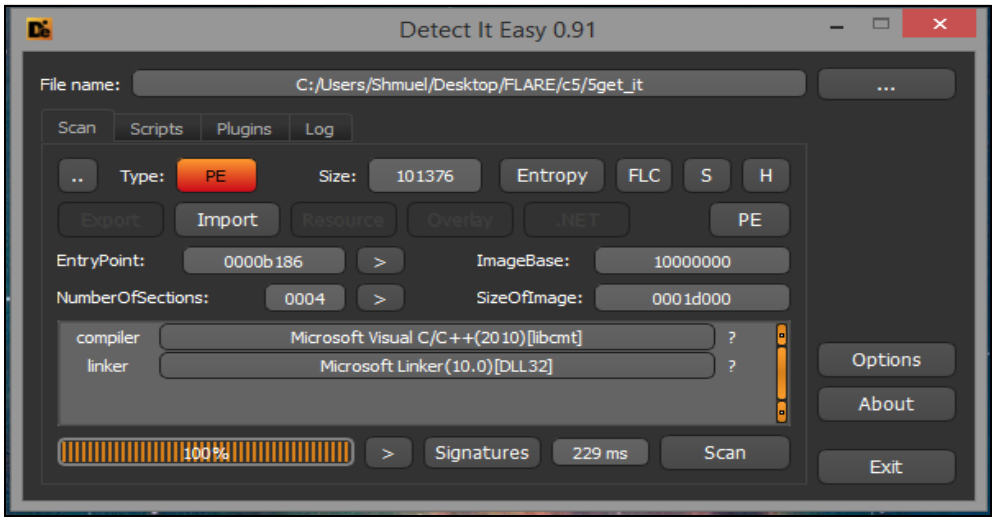
- Intel x86 Instruction Set:
  http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
- MSDN:
  http://msdn.microsoft.com/en-us/library/windows/desktop/ff818516%28v=vs.85%29.aspx

כלים נדרשים:

- DiE / PEiD:
  PEiD - http://www.aldeid.com/wiki/PEiD
  DiE - http://ntinfo.biz/
- IDA Pro/Demo:
  https://www.hex-rays.com/products/ida/support/download.shtml

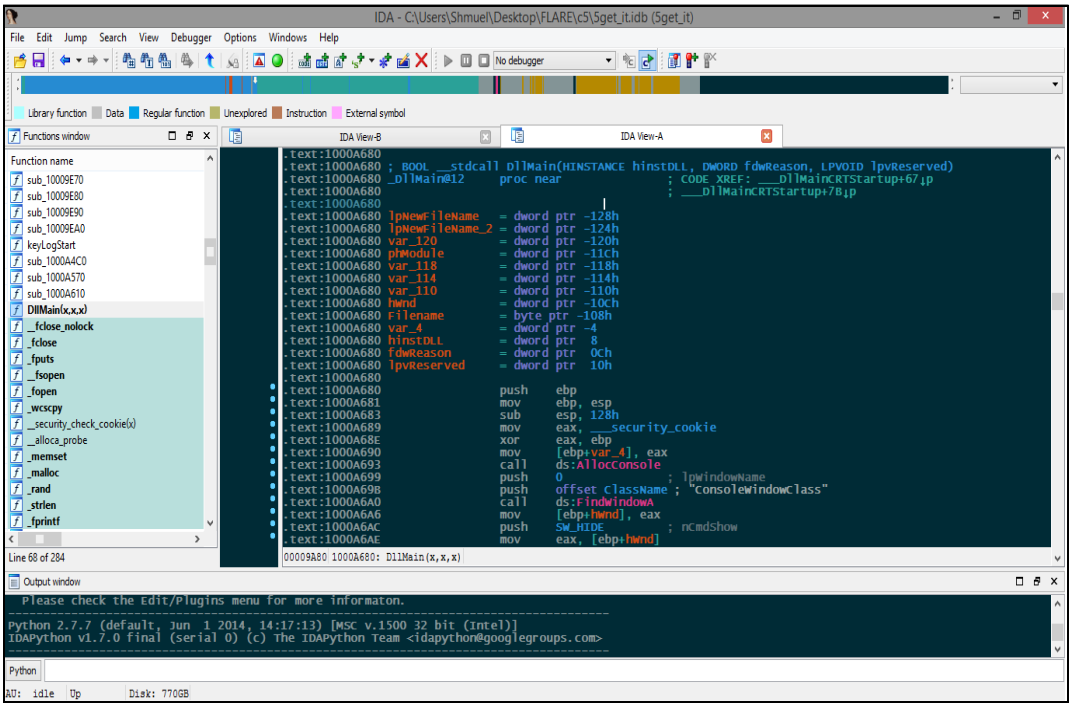
ניתוח התוכנית

לאחר ההורדה נחלץ את הקובץ (הסיסמא היא malware). נפתח אותה ב-DiE:



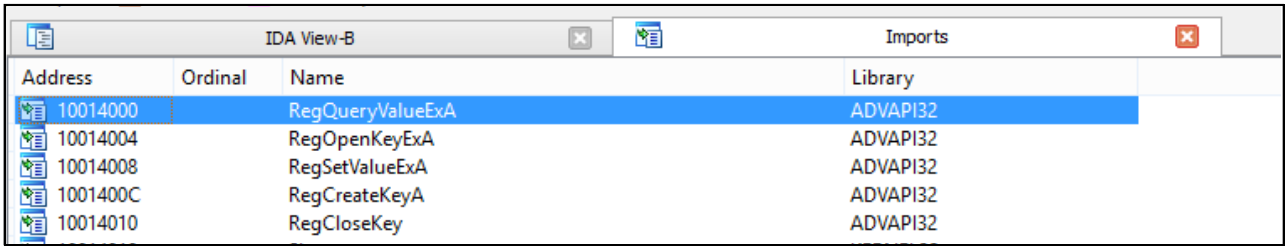
נעבור ל-IDA. לא אסביר את ההבדלים בין ניתוח סטטי לניתוח דינאמי, אבל תוכלו לקרוא עליהם באינטרנט. בחרתי להתחיל בניתוח סטטי, יש לניתוח סטטי כמה יתרונות על ניתוח דינאמי:

- אנחנו לא יודעים מה הקובץ מריץ בדיוק.
- IDA מזהה המון פונקציות ספרייה (ע"י חתימות) שדיבאגר בד"כ לא מצליח לזהות, כך זה חוסך לנו המון זמן בניתוח פונקציות מיותרות.
- יש ב-IDA המון פיצ'רים שעוזרים המון, נראה אותם בהמשך.



[תמונת מסך ה-IDA ב-main של הבינארי]

הדבר הראשון ששמנו לב אליו הוא שמדובר בקובץ DLL. מעולה, נסתכל על הפונקציות שהתוכנית מייבאת (חלון ה-import):



ונחפש קריאה לפונקציות חשודות:

- 1001400C RegCreateKeyA ADVAPI32
- 10014010 RegCloseKey ADVAPI32
- 10014020 CopyFileA KERNEL32
- 1001413C GetAsyncKeyState USER32

הפונקציה הכי חשודה בנתיים היא:

```
SHORT WINAPI GetAsyncKeyState(
  _In_  int vKey
);
```

הפונקציה מקבלת פרמטר אחד (vKey) - הקוד הוירטואלי של מקש מסויים, ומחזירה את המצב שלו (לחוץ או משוחרר). כל זה בנוסף לפונקציות שראינו בחלון ה import (כתיבה לרג'יסטרי ולקבצים) מעלה את החשד שהDLL הזה הוא key Logger.

נמשיך לשוטט בקובץ. בתחילת התוכנית נראה את הקריאות לפונקציות שמטרתם להסתיר את החלון:

```
.text:1000A693                  call    ds:AllocConsole
.text:1000A699                  push    0               ; lpWindowName
.text:1000A69B                  push    offset ClassName ;
"ConsoleWindowClass"
.text:1000A6A0                  call    ds:FindWindowA
.text:1000A6A6                  mov     [ebp+hWnd], eax
.text:1000A6AC                  push    SW_HIDE         ; nCmdShow
.text:1000A6AE                  mov     eax, [ebp+hWnd]
.text:1000A6B4                  push    eax             ; hWnd
.text:1000A6B5                  call    ds:ShowWindow
```

נעבור על הפונקציות:

- AllocConsole - מקצה חלון "קונסול" לתהליך.
- FindWindowA - מציאת החלון.
- ShowWindow - הסתרת החלון ע"י העברת הפרמטר SW_HIDE.

לאחר השורות הנ"ל נראה קריאה לפונקציה בלי פרמטרים:

```
.text:1000A6BB                  call    sub_1000A570
```

בהמשך ניתן לראות שהתוכנית ממשיכה בביצוע קריאות ל:

- GetModuleHandleExA - כדי לקבל ידית (handle) לקובץ.
- GetModuleFileNameA - כדי לקבל את שם הקובץ (כולל הנתיב המלא)
- CopyFileA - העתקת קובץ ממקום למקום

ואח"כ קריאה ל-2 פונקציות שנצטרך לנתח: sub_1000A610 ו-sub_1000A4C0

סיכום ביניים

עד עכשיו נתקלנו ב-3 פונקציות לא מוכרות:

- sub_1000A570
- sub_1000A610
- sub_1000A4C0

נעבור עליהם ונקשר הכל ביחד תוך כדי.

ניתוח sub_1000A570:

sub_1000A570 היא הפונקציה הראשונה, במבט זריז על הפונקציה ניתן לומר עליה כמה דברים:

- קריאה לפונקציה RegOpenKeyExA עם הערך:

```
SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

- קריאה לפונקציה RegQueryValueExA עם הערך svchost.
- משווה את הערך החוזר (lpcbData) עם 0x50 לאחר הפונקציות בשביל לוודא שהכל כמו שצריך (על הערכים עצמם אתם יכולים לקרוא בMSDN)

לא מעניין במיוחד..

נחזור לאיפה שהפסקנו. קודם לכן, הזכרנו את הפונקציה CopyFileA, אבל לא שמנו לב מה הפרמטרים שהיא מקבלת:

```
.text:1000A747                  push    0               ; bFailIfExists
.text:1000A749                  mov     edx, [ebp+lpNewFileName]
.text:1000A74F                  push    edx             ; lpNewFileName
.text:1000A750                  lea     eax, [ebp+Filename]
.text:1000A756                  push    eax             ;
lpExistingFileName
```

```
.text:1000A757                  call    ds:CopyFileA
```

את FileName קיבלנו מ-GetModuleFileNameA, (זוכרים?), אבל מה מכיל lpNewFileName? מעט למעלה נמצאת התשובה:

```
.text:1000A733    mov     [ebp+lpNewFileName], offset aCWindowsSystem ;
"c:\\windows\\system32\\svchost.dll"
```

כבר נתקלנו בשם הקובץ הזה...

ניתוח sub_1000A610:

גם כאן יש פונקציות שקשורות לרג'יסטרי:

- קריאה ל-RegCreateKeyA עם הערך המוכר SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- קריאה ל-RegSetValueExA עם שם הערך scvhost שמכיל c:\windows\system32\rundll32.exe c:\windows\system32\svchost.dll (זה בטח מובן, אבל rundll32.exe מריץ DLL שמועבר כפרמטר [במקרה הזה scvhost.dll]).

RegCreateKeyA - יוצרת מפתח בנתיב שהועבר כפרמטר.

RegSetValueExA - יוצרת ערך+נתונים בנתיב שהועבר כפרמטר

כל מה שנכתב ל-SOFTWARE\Microsoft\Windows\CurrentVersion\Run יורץ כאשר מערכת ההפעלה עולה. במקרה הזה, rundll32.exe עם הפרמטר svchost.dll.

עכשיו הכל מתחיל להתבהר...

עד עכשיו הבנו בערך מה הפונקציה עושה:

1. מעתיקה את עצמה לנתיב c:\windows\system32 עם השם svchost.dll

יוצרת ערך חדש ברג'יסטרי בנתיב SOFTWARE\Microsoft\Windows\CurrentVersion\Run בשם svchost עם הערך c:\windows\system32\rundll32.exe c:\windows\system32\svchost.dll

ניתוח sub_1000A4C0:

בגדול, אפשר להסתבך פה עם כל מיני תנאים ולולאות, אבל אני אסביר בקצרה מה הולך פה (לא בא לי לגמור לכם את הדיו בבית... ☺)

- יש פה לולאה אינסופית שמתחילה ב: loc_1000A4C6 (זה לא משנה באמת איך הוא נוצרה שם)
- לאחר מכן יש קריאה ל-sub_10009EB0
- בדיקת שהערך החוזר גדול מ-0
- ולאחר מכן קריאה ל-sub_10001000

ננתח את הפונקציות בדיוק כמו מקודם.

הערה קטנה, סדר ניתוח הפונקציות לא חייב להתבצע ע"פ סדר הקריאה, אלא גם לפי נוחות. הכוונה: אם יש פונקציה שהיא לא ארוכה/מובנת מאוד עדיף להסתכל עליה לפני הכל, לכן - אני אתחיל ב sub_10001000

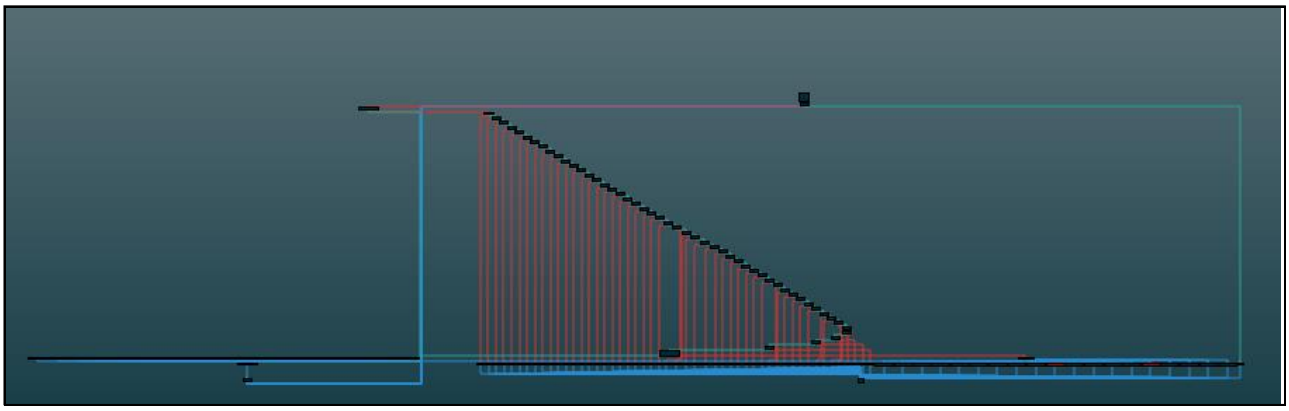
ניתוח sub_10001000:

אני מקווה שהבנתם את העסק. הפונקציה כותבת את הערך (מן הסתם ערך ASCII) שמועבר אליה - לקובץ svchost.log.

ניתוח sub_10009EB0:

אם עד עכשיו לא נתקענו, כנראה ש-sub_10009EB0 היא האתגר האמיתי.

sub_10009EB0 נראית אולי טיפה מפחיד בהתחלה, אבל סה"כ היא באמת בסדר...



[מפה כבר ניתן לראות שזה רצף של משפטי switch case או if else]

נתחיל מנק' הכניסה: כבר בהתחלה אנו רואים שיש כאן לולאה שרצה מ-0x80 ל-0x0DE. נמשיך, ונוכל לראות את קטע הקוד המיוחל:

```
.text:10009EDD                  movsx   eax, [ebp+vkCode]
.text:10009EE1                  push    eax             ; vKey
.text:10009EE2                  call    ds:GetAsyncKeyState
```

קצת על GetAsyncKeyState:

GetAsyncKeyState מחזירה לנו את המצב של המקש (לחוץ או משוחרר), להלן דוגמא:

```
#include <iostream>
#include <windows.h>

int main() {
      int j;
      while(1) {
            j = GetAsyncKeyState(0x47); //0x47 is vkCode for G key
            if (j != 0) std::cout << "G key Pressed " << std::hex << j;
            Sleep(100);
      }
      return 0;
}
```

אם הערך החוזר הוא 0, המקש לא לחוץ, אם הערך החוזר הוא 0FFFF8001h המקש לחוץ.

נמשיך...

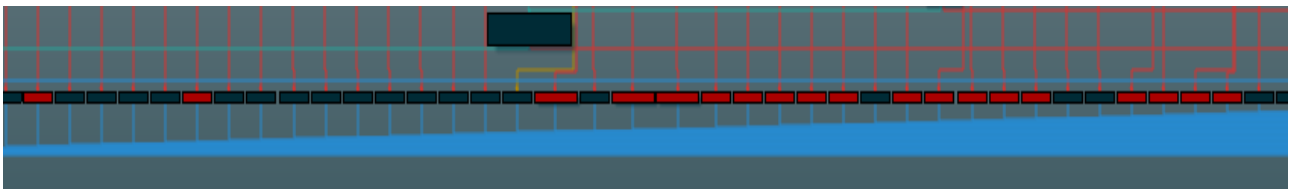
```
.text:10009EF7                  movsx   edx, [ebp+vkCode]
.text:10009EFB                  cmp     edx, 27h        ; ASCII '
.text:10009EFE                  jnz     short loc_10009F0A
.text:10009F00                  call    sub_100093B0
.text:10009F05                  jmp     loc_1000A3A6
.text:10009F0A ; ---------------------------------------------------------
--------------------
.text:10009F0A
.text:10009F0A loc_10009F0A:                            ; CODE XREF:
sub_10009EB0+4E#j
.text:10009F0A                  movsx   eax, [ebp+vkCode]
.text:10009F0E                  cmp     eax, '('
.text:10009F11                  jnz     short loc_10009F1D
.text:10009F13                  call    sub_100093C0
.text:10009F18                  jmp     loc_1000A3A6
```

בהנחה שהערך שחזר הוא 0FFFF8001h, אנחנו נכנסים לרצף של משפטי if else (כתובת היציאה היא loc_1000A3A6, נפרט על כך בהמשך).

מבט קצר ב-sub_100093B0 מראה שהפונקציה מחזירה (ב eax) את ערך ה-ASCII של אותו קוד וירטואלי (vkCode). ולאחר מכן קופצת לסוף הפונקציה (loc_1000A3A6), מה שמביא אותנו ל-sub_10001000 (כתיבה לקובץ זוכרים?). כל זה בתוך הלולאה האינסופית.

בשלב זה התחלתי לחשוש, הרי כל המטרה של האתגר הזה הוא מציאת הסיסמא. הדבר המתבקש מזה שאנחנו נצטרך לנתח כל קריאה לכל תו אפשרי ע"מ לדעת באיזה תו חל שינוי בפונקציה (מה שהתברר לאחר מכן כצ'יפס ☺)

מעבר זריז על הפונקציות שנקראות לאחר הבדיקה כדוגמא: sub_100093B0, sub_100093C0. מראה שיש כמה כאלה שהן קצת שונות:



[הפונקציות השונות מסומנות באדום]

ניקח לדוגמא את הפונקציה שנקראת לאחר לחיצה על האות C:

```
.text:10009850 sub_10009850    proc near               ; CODE XREF:
sub_10009EB0+251#p
.text:10009850                 push    ebp
.text:10009851                 mov     ebp, esp
.text:10009853                 cmp     dword_100194F4, 0
.text:1000985A                 jle     short loc_10009872
.text:1000985C                 mov     dword_100194F4, 0
.text:10009866                 mov     dword_100194F8, 1
.text:10009870                 jmp     short loc_10009877
.text:10009872 ; ---------------------------------------------------------
--------------------
.text:10009872
.text:10009872 loc_10009872:                           ; CODE XREF:
sub_10009850+A#j
.text:10009872                 call    __cfltcvt_init
.text:10009877
.text:10009877 loc_10009877:                           ; CODE XREF:
sub_10009850+20#j
.text:10009877                 mov     eax, offset aC_0 ; "c"
.text:1000987C                 pop     ebp
.text:1000987D                 retn
.text:1000987D sub_10009850    endp
```

טוב, נראה דיי מובן, אבל מה זה __cfltcvt_init? פתיחה זריזה מראה ש-__cfltcvt_init מאתחלת מערך (או רצף משתנים איך שבא לכם).

הצצה בעוד כמה כאלה, מראה שצריך להקליד את האותיות לפי הסדר ע"מ לקבל את הפונקציה האחרונה, אחרת הכל מתאפס שוב ע"י cfltcvt_init__...

```
.text:10009AF0 sub_10009AF0    proc near               ; CODE XREF:
sub_10009EB0+30F#p
.text:10009AF0                 push    ebp
.text:10009AF1                 mov     ebp, esp
.text:10009AF3                 cmp     dword_100194FC, 0
.text:10009AFA                 jle     short loc_10009B06
.text:10009AFC                 call    __cfltcvt_init
.text:10009B01                 call    sub_10001240    ; init var and
print someting to the screen
.text:10009B06
.text:10009B06 loc_10009B06:                           ; CODE XREF:
sub_10009AF0+A#j
.text:10009B06                 mov     eax, offset aM_0 ; "m"
.text:10009B0B                 pop     ebp
.text:10009B0C                 retn
.text:10009B0C sub_10009AF0    endp
```

[נשים לב ש-sub_10009AF0 שונה מהאחרות בגלל הקריאה ל-sub_10001240, שמדפיסה על המסך נתונים]

קודם לכן, שמנו לב שלחיצה על האות גורמת לפונקציה מסוימת להיקרא ובאותה פונקציה משווים משתנה מסוים ל-0, אם הוא שווה אז ממשיכים כרגיל. אך אם הוא שונה - מגדירים משתנה אחר ל-1 ואת הראשון ל-0...

זאת אומרת שכל לחיצה על מקש גורמת להשוואה של ערך מסוים ל-0 אם הוא שונה - הגדרה של משתנה אחר ל-1. ואותה פונקציה שבודקת את אותו הערך השני(שהפך ל-1 לפני שניה) נקראת ע"י לחיצה על מקש אחר, שגורמת לקוד אחר שנקרא ע"י לחיצת מקש אחר לבדוק את אותו הערך וכך הלאה. נשמע מסובך, ונשמע שקשה למצוא את זה בכל ערימות הקוד הזה... ☹

אבל הצצה חוזרת ב-cfltcvt_init__ פתרה את הבעיה במהירות.

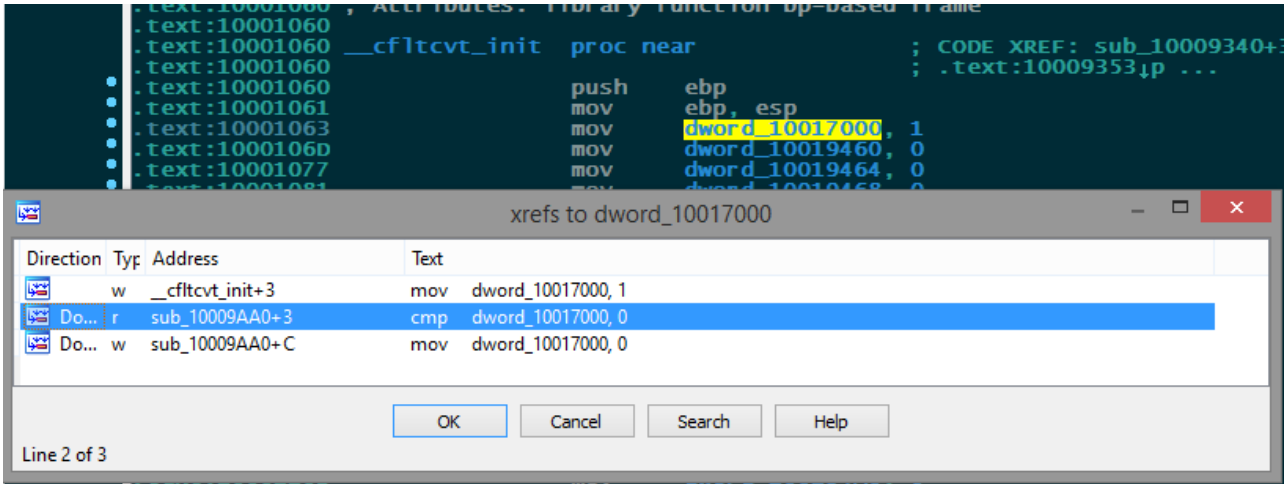
XREF - Cross reference

XREF - בעזרת ה-XREF אפשר למצוא איפה יש התייחסות לאותו אובייקט בכל הבינארי, זה יכול להיות פונקציה או תא זיכרון למשל (את הרעיון הזה אפשר גם למצוא בכלים אחרים כמו ollydbg). אם נשים לב כל המשתנים מאותחלים ל-0 חוץ מהראשון:

```
.text:10001060 __cfltcvt_init  proc near               ; CODE XREF:
sub_10009340+3#p
.text:10001060                                         ;
.text:10009353#p ...
.text:10001060                 push    ebp
.text:10001061                 mov     ebp, esp
.text:10001063                 mov     dword_10017000, 1
.text:1000106D                 mov     dword_10019460, 0
.text:10001077                 mov     dword_10019464, 0
```

אני מקווה שאתם יודעים מה צריך לעשות...

למי שלא מכיר את פונקציית ה-XREF אפשר להפעיל ע"י לחיצה על הערך ולחיצה על מקש הX או בקליק ימני בעכבר:

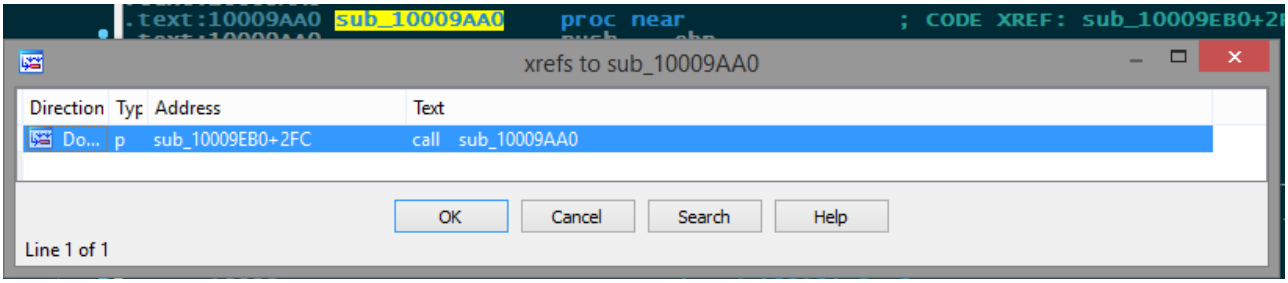


[XREF to dword_10017000]

נלחץ על השורה השניה, (אנחנו מחפשים את ההשוואה זוכרים?), וזה לוקח אותנו לפה:

```
.text:10009AA0 sub_10009AA0    proc near               ; CODE XREF:
sub_10009EB0+2FC#p
.text:10009AA0                 push    ebp
.text:10009AA1                 mov     ebp, esp
.text:10009AA3                 cmp     dword 10017000, 0
.text:10009AAA                 jle     short loc_10009AC2
```

נראה מעניין, אבל מאיפה הפונקציה הזאת נקראת?

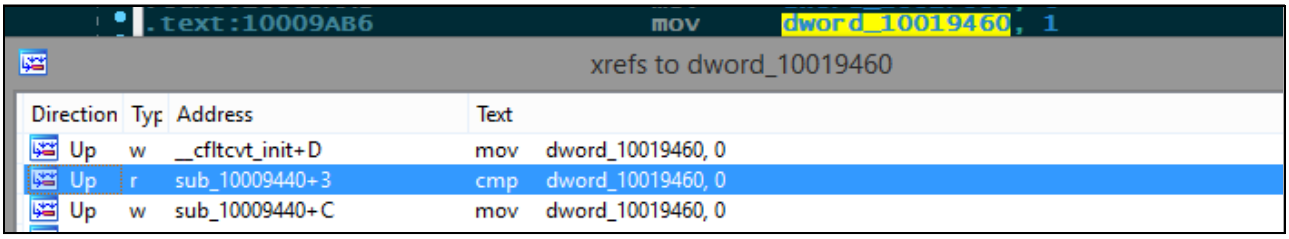


[XREF to sub_10009AA0]

יפה, sub_10009AA0 נקראת ע"י לחיצה על האות L.

```
sub_10009AA0
      :
.text:10009AAA                 jle     short loc_10009AC2
.text:10009AAC                 mov     dword_10017000, 0
.text:10009AB6                 mov     dword_10019460, 1
.text:10009AC0                 jmp     short loc_10009AE6
```

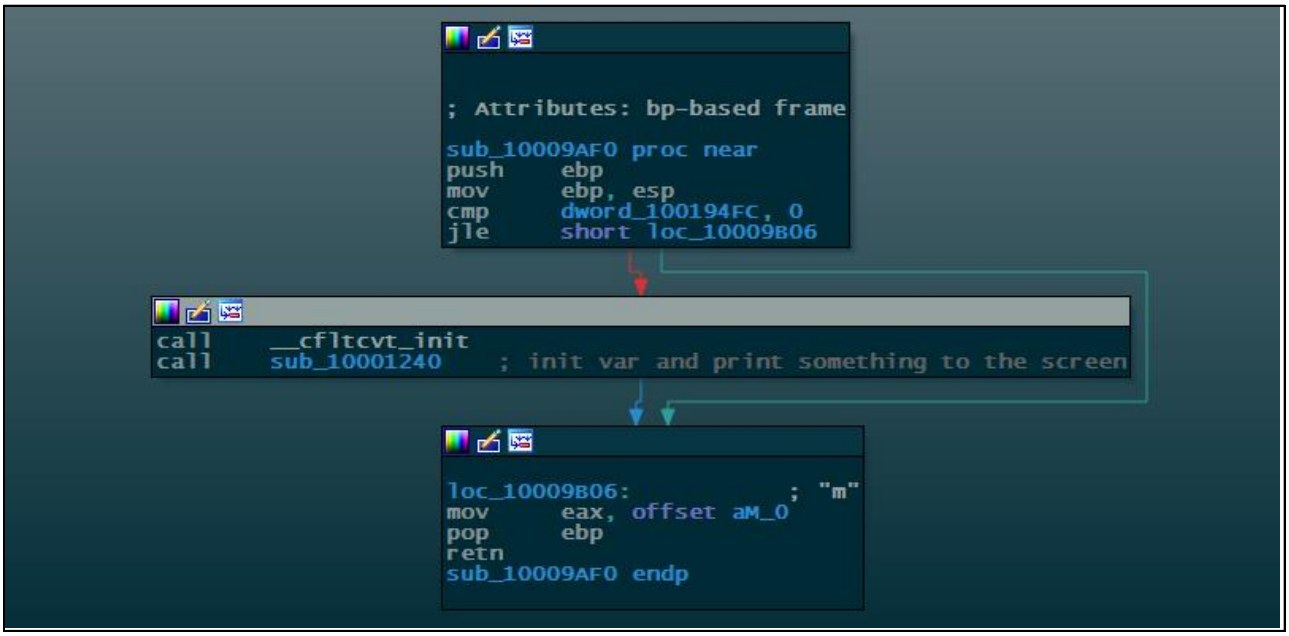
אני מקווה שאתם יודעים מה ההמשך...



[XREF to dword_10019460]

```
text:10009440 sub_10009440    proc near               ; CODE XREF:
sub_10009EB0+FB#p
.text:10009440                                         ;
sub_10009EB0:loc_1000A347#p
.text:10009440                 push    ebp
.text:10009441                 mov     ebp, esp
.text:10009443                 cmp     dword_10019460, 0
.text:1000944A                 jle     short loc_10009462
.text:1000944C                 mov     dword_10019460, 0
.text:10009456                 mov     dword_10019464, 1
.text:10009460                 jmp     short loc_10009486
```

כל זה נקרא ע"י לחיצה על מקש ה-0. הבנתם את הקטע... כאשר נגיע אל התו האחרון - נראה את זה:



(sub_10001240 לא מעניינת במיוחד, היא סה"כ מדפיסה הודעה על המסך אם הקלדנו הכל לפי הסדר בעזרת DialogBoxIndirectParamW, בזמנכם הפנויי אתם יכולים לנתח אותה). עם עשיתם הכל כמו שצריך, תקבלו כתובת מייל שנראית כך:

L0GGingdotUrdot5tr0ke5atFLAREdasOndotcom = L0gging.Ur.5tr0ke5@flare-on.com

סיכום

ניתחנו DLL שכותב ערך מסוים (svchost) עם נתונים (rundll32.exe c:\.....\svchost.dll) כאשר svchost.dll הוא העתק של הקובץ שלנו, לנתיב SOFTWARE\Microsoft\Windows\CurrentVersion\Run ברג'יסטרי, כך שהוא יופעל גם לאחר כיבוי והדלקת המחשב.

ניתחנו את הפונקציות, וראינו איך GetAsyncKeyState מנטרת המקלדת.

לבסוף עלינו על הטריק, ופתרנו את האתגר.

לא התעכבתי על הפונקציות של windows, הסבר מפורט של כל אחת ניתן למצוא ב-MSDN.

אם כבר הגענו עד לכאן אני רוצה להודות לכמה חברה: Antartic, electron ו-deadly (d4d) על כל העזרה בדרך.

ניתן ליצור עימי קשר בערוץ #reversing או בפורום.

קישורים ללמידה עצמית:

- Lena's Reversing
- R4ndom tuts
- Reddit - Reverse engineering

Happy Reversing!