Digital Whisper Technologic Papers

אתגרי ה-CrackMe של רפאל 2015

מאת דימה פשול

הקדמה

המאמר בא לתאר ולתעד את הדרך בה ניגשתי ופתרתי את אתגרי ה-CrackMe של רפאל במאי השנה. האתגרים פורסמו באתר של רפאל, בקישור הבא:

http://portal.rafael.co.il/rechallenge15/Documents/rechallenge15/index.html

האתגרים פורסמו באותו פורמט של האתגרים של השב"כ משנת 2009 ונקראים באותם השמות (stage0.exe - stage3.exe). בכל אתגר על הפותר למצוא סיסמא המדפיסה הודעת ניצחון ו-E-mail איתו ניתן ליצור קשר ולהירשם כפותר האתגר הספציפי. לאורך תיעוד הפיתרון לאתגרים ניסיתי להביא כמה שיותר דוגמאות ותמונות מ-IDA והכלים בהם השתמשתי. בנוסף ניסיתי להביא פתרונות כמה שיותר מקצועיים הבאים לידי ביטוי בפתרון ע"י כתיבת קוד שפותר את האתגרים - כמובן שלא בכולם זה אפשרי. בזמן קריאת המאמר יש לשים לב לכמה דברים:

כתובות ספציפיות

לאורך המאמר נקובות כתובות ספציפיות בקבצים המנותחים אשר משתנות מריצה לריצה. יש לקחת בחשבון שתי משמעויות שנובעות מכך:

1. אם ומי מקוראי המאמר שירצה לשחזר את פתרון ה-Crackme עלול להיתקל בכתובות לא תואמות לאלה המצוינות במאמר.
2. לאורך המאמר ובפתרון כל Crackme התמונות המצורפות להמחשה יכולות להיות משתי הרצות שונות ב-IDA ולכן הכתובות לא יהיו תואמות בין שתי תמונות או יותר.

מה שכן אפשר לעשות בשביל לעקוב אחרי הכתובות הוא להשתמש ב-4 ספרות האחרונות כסימניות, למשל הכתובות הבאות כנראה יכילו את אותו המידע\קוד בין 3 הרצות שונות:

```
0x001360C2
0x00FD60C2
0x00C060C2
```

מושגים וקיצורים

- **BP** - Breakpoint.

- **XREF** - Cross Reference (פונקציה ב-IDA המראה איפה הפונקציה / אובייקט / כתובת מאוזכרת לאורך הקוד).
- **RUR** - Run until execution returns from the current function (אפשרות ריצה ב-IDA המבצעת ריצה עד אחרי ה-retn הראשון).

קישורים

לאורך הקוד יש מילים מודגשות המכילות קישורים לרשימת פונקציות / מושגים הנמצאת בתחתית המאמר, עשו בהם שימוש.

אתגר 1 - stage0.exe:

ידע דרוש לפתרון:

- Registry
- אלגוריתמי קידוד מוכרים
- Windows API

חימום:

נפתח את הקובץ ב-IDA, נגיע ל-Main שנראה ככה:

```
01262040
01262040
01262040 ; Attributes: bp-based frame
01262040
01262040 ; int __cdecl main(int argc, const char **argv, const char **envp)
01262040 _main proc near
01262040
01262040 phkResult= dword ptr -0B0h
01262040 pcchString= dword ptr -0ACh
01262040 cbData= dword ptr -0A8h
01262040 pszString= byte ptr -0A4h
01262040 Data= byte ptr -54h
01262040 var_4= dword ptr -4
01262040 argc= dword ptr  8
01262040 argv= dword ptr  0Ch
01262040 envp= dword ptr  10h
01262040
01262040 push    ebp
01262041 mov     ebp, esp
01262043 sub     esp, 0B0h
01262049 mov     eax, ___security_cookie
0126204E xor     eax, ebp
01262050 mov     [ebp+var_4], eax
01262053 mov     eax, 50h
01262058 mov     [ebp+cbData], eax
0126205E mov     [ebp+pcchString], eax
01262064 lea     eax, [ebp+phkResult]
0126206A push    eax             ; phkResult
0126206B push    20019h          ; samDesired
01262070 push    0               ; ulOptions
01262072 push    offset SubKey   ; "Software\\rafael"
01262077 push    80000001h       ; hKey CurrentUser
0126207C call    ds:RegOpenKeyExA
01262082 test    eax, eax
01262084 jnz     loc_126210B
```

הקוד מנסה לפתוח את המפתח "HKEY_CURRENT_USER\Software\rafael" של ה-Registry בעזרת **RegOpenKeyExA**, במידה והוא לא מצליח הוא יקפוץ להודעת השגיאה, במידה והוא מצליח נגיע לקטע הקוד הבא:

```
0126208A lea     ecx, [ebp+cbData]
01262090 push    ecx             ; lpcbData
01262091 lea     edx, [ebp+Data]
01262094 push    edx             ; lpData
01262095 push    eax             ; lpType
01262096 push    eax             ; lpReserved
01262097 mov     eax, [ebp+phkResult]
0126209D push    offset ValueName ; "Password"
012620A2 push    eax             ; hKey
012620A3 call    ds:RegQueryValueExA
012620A9 test    eax, eax
012620AB jnz     short loc_126210B
```

הקוד יבצע תשאול לגבי ה-Value של "Password" בעזרת **RegQueryValueExA**, לאחר מכן יקודד אותו בעזרת הפונקציה **CryptBinaryToStringA** עם קידוד של **Base64**:

```
012620AD mov     eax, [ebp+cbData]
012620B3 lea     ecx, [ebp+pcchString]
012620B9 push    ecx             ; pcchString
012620BA lea     edx, [ebp+pszString]
012620C0 push    edx             ; pszString
012620C1 push    40000001h       ; dwFlags
012620C6 push    eax             ; cbBinary
012620C7 lea     ecx, [ebp+Data]
012620CA push    ecx             ; pbBinary
012620CB call    ds:CryptBinaryToStringA
012620D1 test    eax, eax
012620D3 jz      short loc_126210B
```

לאחר מכן התכנית תשווה את מה שקיבלה לאחר הקידוד עם ה-String **"U3RhZ2UjMHBhc3N3b3JkIQA="** בעזרת הפונקציה **strncmp**. בשביל לפתור את האתגר הזה עלינו לבצע Decode ל-String הנ"ל - נקבל **"Stage#0password!"** את ה-String הזה עלינו להכניס לתוך ערך בשם "Password" בנתיב ה-Registry הבא - HKEY_CURRENT_USER\Software\rafael. לאחר הפעולות האלה ניתן להריץ את הקובץ:

```
C:\Windows\system32\cmd.exe

C:\Users\Alien\Downloads\All4Stages>Stage0.exe
Congratulations! You have solved this challenge!
To get the next challenge send an e-mail to the following address: iFv2h1ZnmNG@g
mail.com .
To get the points for the raffle you must explain in the mail how you solved thi
s challenge.
C:\Users\Alien\Downloads\All4Stages>
```

כעת הקובץ ייגש לערך שהוספנו, יקודד אותו בעזרת base64 ויעשה השוואה בין הערך המקודד ל-String הקבוע בקוד (ההשוואה כמובן תיהיה נכונה כעת).

אתגר 2 - stage1.exe

ידע דרוש לפתרון:

- x86 Assembly
- Anti-Debugging מבוסס זמן

ספרינט 100 מטר:

החלק הראשון של ה-Main נראה ככה:

```
00282000
00282000
00282000 ; Attributes: bp-based frame
00282000
00282000 ; int __cdecl main(int argc, const char **argv, const char **envp)
00282000 _main proc near
00282000
00282000 var_200= byte ptr -200h
00282000 var_1FF= byte ptr -1FFh
00282000 var_100= byte ptr -100h
00282000 var_FF= byte ptr -0FFh
00282000 argc= dword ptr  8
00282000 argv= dword ptr  0Ch
00282000 envp= dword ptr  10h
00282000
00282000 push    ebp
00282001 mov     ebp, esp
00282003 sub     esp, 200h
00282009 push    0FFh            ; size_t
0028200E lea     eax, [ebp+var_FF]
00282014 push    0               ; int
00282016 push    eax             ; void *
00282017 mov     [ebp+var_100], 0
0028201E call    _memset
00282023 push    0FFh            ; size_t
00282028 lea     ecx, [ebp+var_1FF]
0028202E push    0               ; int
00282030 push    ecx             ; void *
00282031 mov     [ebp+var_200], 0
00282038 call    _memset
0028203D push    offset aPleaseEnterAUs ; "Please enter a user-name: ...\n"
00282042 call    _printf
00282047 push    100h
0028204C lea     edx, [ebp+var_100]
00282052 push    edx
00282053 push    offset Format   ; "%s"
00282058 call    _scanf_s
0028205D lea     eax, [ebp+var_100]
00282063 push    eax             ; char *
00282064 call    sub_281B90
00282069 add     esp, 2Ch
0028206C test    eax, eax
0028206E jnz     short loc_28208D
```

הקוד יבקש מאיתנו Username, לאחר מכן יעבור ל-**sub_281B90** ויבצע **strncmp** של ה-Username שהכנסנו עם ה-String "root". ולכן נריץ מחדש וה-String שנכניס לו ב-Username יהיה root.

במידה וה-Username נכון נעבור לקטע הקוד הבא המבקש לקבל את הסיסמא למשתמש, ולאחר מכן עובר ל-sub_281DE0 בשביל לאמת את הסיסמא:

```
0028208D
0028208D loc_28208D:
0028208D lea     edx, [ebp+var_100]
00282093 push    edx
00282094 push    offset aHelloSPleaseEn ; "Hello %s, please enter the password: ..."...
00282099 call    _printf
0028209E push    100h
002820A3 lea     eax, [ebp+var_200]
002820A9 push    eax
002820AA push    offset Format   ; "%s"
002820AF call    _scanf_s
002820B4 add     esp, 14h
002820B7 lea     ecx, [ebp+var_100]
002820BD push    ecx             ; char *
002820BE lea     edx, [ebp+var_200]
002820C4 push    edx             ; void *
002820C5 call    sub_281DE0
002820CA neg     eax
002820CC sbb     eax, eax
002820CE and     eax, 0FFFFFFFEh
002820D1 add     eax, 2
002820D4 mov     esp, ebp
002820D6 pop     ebp
002820D7 retn
002820D7 _main endp
002820D7
```

שרשרת הטעיות

תחילת הפונקציה sub_281DE0 נראית ככה:

```
00281DE0
00281DE0
00281DE0 ; Attributes: bp-based frame
00281DE0
00281DE0 ; int __stdcall sub_281DE0(void *, char *)
00281DE0 sub_281DE0 proc near
00281DE0
00281DE0 var_34= dword ptr -34h
00281DE0 var_30= dword ptr -30h
00281DE0 var_2C= dword ptr -2Ch
00281DE0 var_28= dword ptr -28h
00281DE0 flOldProtect= dword ptr -24h
00281DE0 var_20= dword ptr -20h
00281DE0 var_1C= dword ptr -1Ch
00281DE0 Time= qword ptr -18h
00281DE0 var_C= dword ptr -0Ch
00281DE0 var_8= dword ptr -8
00281DE0 var_4= dword ptr -4
00281DE0 arg_0= dword ptr  8
00281DE0 arg_4= dword ptr  0Ch
00281DE0
00281DE0 push    ebp
00281DE1 mov     ebp, esp
00281DE3 and     esp, 0FFFFFFF8h
00281DE6 sub     esp, 34h
00281DE9 push    ebx
00281DEA push    esi
00281DEB push    edi
00281DEC xor     edi, edi
00281DEE push    edi             ; Time
00281DEF mov     [esp+44h+var_20], 32CCFA3Ah
00281DF7 mov     [esp+44h+var_1C], 0A750D6B0h
00281DFF call    __time64
00281E04 mov     dword ptr [esp+44h+Time], eax
00281E08 lea     eax, [esp+44h+Time]
00281E0C add     esp, 4
00281E0F push    eax             ; Time
00281E10 mov     dword ptr [esp+44h+Time+4], edx
00281E14 call    __localtime64
00281E19 add     esp, 4
00281E1C mov     [esp+40h+var_28], eax
00281E20 call    ds:GetTickCount
00281E26 mov     [esp+40h+flOldProtect], eax
00281E2A xor     ecx, ecx
00281E2C mov     [esp+40h+var_30], edi
00281E30 mov     [esp+40h+var_34], edi
00281E34 mov     [esp+40h+var_2C], edi
00281E38 jmp     short loc_281E40
```

הפונקציה מבצעת שלל קריאות לפונקציות שאינן רלוונטיות לפיתרון האתגר ואינן משמשות אותו בשום צורה, למעט 2 פונקציות והן VirtualProtect ו-GetTickCount. לאחר הקריאה הראשונה ל-GetTickCount תבצע sub_281DE0 מספר חישובים שאינם רלוונטיים, לאחר מכן תעבור לקריאה שניה ל-GetTickCount, אחרי קריאה זאת יתבצע חיסור בין הזמן הראשון שהתקבל והזמן השני שהתקבל מ-GetTickCount, במידה ותוצאת החיסור אינה 0, הקוד יברח לנו ולא יבצע בדיקה של הסיסמא, בדיקה זאת חוזרת על עצמה לאורך הקוד 3 פעמים בעזרת 4 קריאות ל-GetTickCount. בשלב הבא יתבצע שימוש ב-OutputDebugStringA, לעיתים פונקציה זאת יכולה לשמש כ-Anti-Debugging, המקרה הזה אינו כזה ומודפס לנו ה-String: "Every Good Boy Does Fine", לאחר ההדפסה שוב מתבצעת בדיקת זמנים ע"י GetTickCount:

```
00281EF9 mov     edi, ds:GetTickCount
00281EFF call    edi ; GetTickCount
00281F01 mov     ebx, [esp+40h+flOldProtect]
00281F05 sub     eax, ebx
00281F07 jnz     short loc_281F5A

00281F09 push    offset OutputString ; "Every Good Boy Does Fine"
00281F0E call    ds:OutputDebugStringA
00281F14 mov     esi, ds:GetLastError
00281F1A call    esi ; GetLastError
00281F1C cmp     eax, 30Fh
00281F21 jz      loc_281FE3

00281F27 lea     edx, [esp+40h+flOldProtect]
00281F2B push    edx             ; lpflOldProtect
00281F2C push    40h             ; flNewProtect
00281F2E push    0C8h            ; dwSize
00281F33 push    offset sub_281AB0 ; lpAddress
00281F38 mov     [esp+50h+flOldProtect], 0
00281F40 call    ds:VirtualProtect
00281F46 test    eax, eax
00281F48 jnz     short loc_281F65
```

בחלק הבא יתבצע VirtualProtect אשר ישנה את הרשאות הזכרון לפונקציה של sub_281AB0 ל-RW (Read Write) בשביל שהקוד יוכל בעתיד לפענח ולשנות את הפונקציה בזמן הריצה מאחר וברירת המחדל היא Read (אם נציץ בתוכה נראה שהיא אינה נראית תקינה).

נעבור לקטע הקוד הבא:

```
00281F65
00281F65 loc_281F65:
00281F65 call    sub 281B30
00281F6A mov     eax, [ebp+arg_4]
00281F6D push    eax             ; char *
00281F6E call    sub 281B90
00281F73 add     esp, 4
00281F76 test    eax, eax
00281F78 jz      short loc_281FE3

00281F7A mov     esi, [ebp+arg_0]
00281F7D lea     ecx, [esp+40h+var_20]
00281F81 mov     [esp+40h+var_C], esi
00281F85 mov     [esp+40h+var_8], ecx
00281F89 mov     [esp+40h+var_4], 8
00281F91 call    edi ; GetTickCount
00281F93 sub     eax, ebx
00281F95 jnz     short loc_281F5A

00281F97 lea     edx, [esp+40h+var_C]
00281F9B push    edx
00281F9C call    sub 281AB0
00281FA1 add     esp, 4
00281FA4 test    al, al
00281FA6 jnz     short loc_281FC4

00281FC4
00281FC4 loc_281FC4:
00281FC4 call    edi ; GetTickCount
00281FC6 sub     eax, ebx
00281FC8 jnz     short loc_281F5A
```

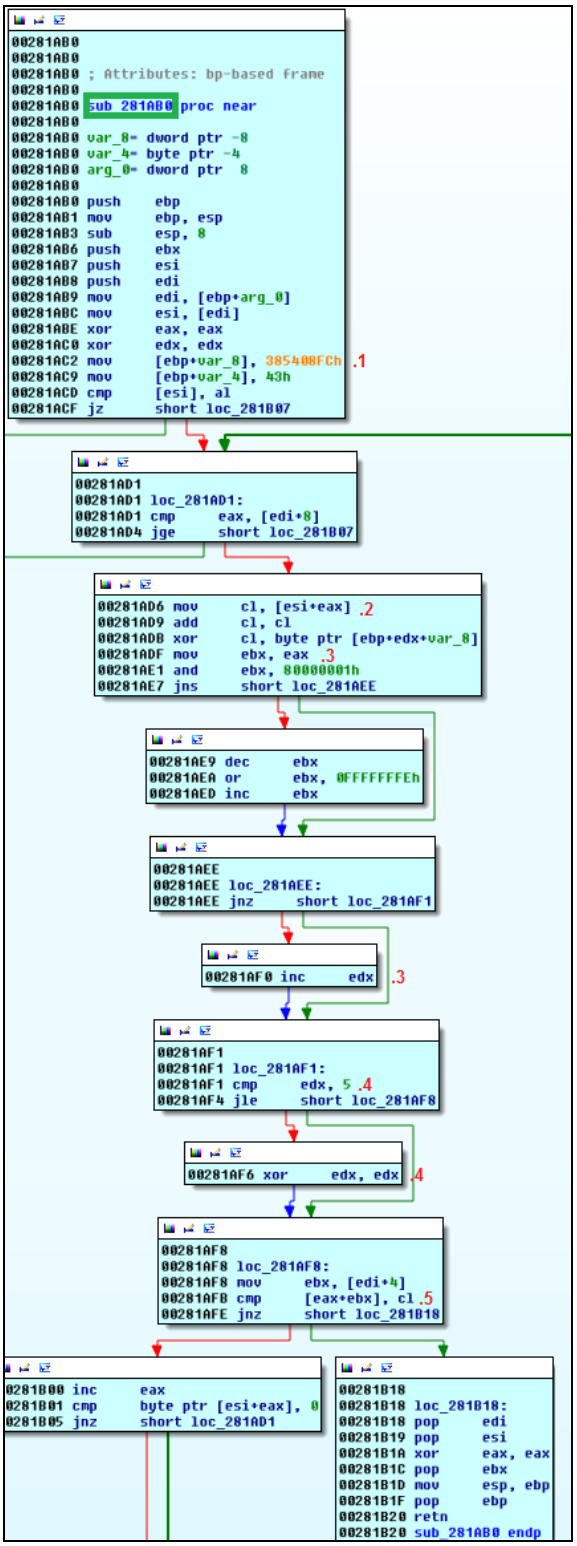
הפונקציה sub_281B30 תבצע פיענוח ל-sub_281AB0:

```
00281AB0
00281AB0
00281AB0 ; Attributes: bp-based frame
00281AB0
00281AB0 sub_281AB0 proc near
00281AB0
00281AB0 var_8= dword ptr -8
00281AB0 var_4= byte ptr -4
00281AB0 arg_0= dword ptr  8
00281AB0
00281AB0 push    ebp
00281AB1 mov     ebp, esp
00281AB3 sub     esp, 8
00281AB6 push    ebx
00281AB7 push    esi
00281AB8 push    edi
00281AB9 mov     edi, [ebp+arg_0]
00281ABC mov     esi, [edi]
00281ABE xor     eax, eax
00281AC0 xor     edx, edx
00281AC2 mov     [ebp+var_8], 385408FCh
00281AC9 mov     [ebp+var_4], 43h
00281ACD cmp     [esi], al
00281ACF jz      short loc_281B07

00281AD1
00281AD1 loc_281AD1:
00281AD1 cmp     eax, [edi+8]
00281AD4 jge     short loc_281B07

00281AD6 mov     cl, [esi+eax]
00281AD9 add     cl, cl
00281ADB xor     cl, byte ptr [ebp+edx+var_8]
00281ADF mov     ebx, eax
```

```
00281AB0
00281AB0
00281AB0
00281AB0 sub_281AB0 proc near
00281AB0 sbb     eax, 22216BE8h
00281AB5 inc     eax
00281AB6 xor     cl, dl
00281AB8 cmc
00281AB9 inc     ebp
00281ABA xor     eax, 0FD950C6Bh
00281ABF mov     [eax+55h], dl
00281AC2 mov     esi, gs:[eax-9097061h]
00281AC9 mov     fs, dword ptr [esi]
00281ACB jnp     short near ptr byte_281AAE

00281ACD test    byte ptr [esi+17h], 0B1h
00281AD1 cdq
00281AD2 mov     [eax+1Eh], eax
00281AD5 mov     dh, 28h
00281AD7 retn    614Eh
00281AD7 sub_281AB0 endp
00281AD7
```

לפני

הפונקציה sub_281B90 מבצעת השוואה נוספת של "root" אל מול ה-Username שהוכנס. לפני הכניסה ל-sub_281AB0 תתבצע עוד בדיקת השוואת זמנים ואם נעבור אותה ניכנס לפונקציה sub_281AB0 שהיא פונקציית בדיקת הסיסמא. הפונקציה נראית ככה:



הפונקציה כוללת 5 נקודות ציון ובמהלך ריצתה עוברת על ה-Byte-ים של הסיסמא שהוכנסה אחד-אחד:

1. אתחול המפתח (0xFC08543843)
2. האלגוריתם המתמטי:
   א. mov cl, [esi+eax] - העברת ה-byte שנבדק ל-cl
   ב. cl = cl * 2 - add cl, cl
   ג. xor cl, byte ptr [ebp+edx+var_8] - ביצוע xor של cl עם ה-Byte עליו מצביע המפתח edx הוא ה-Offset שלנו.
3. במידה ומספר לולאת הריצה שלנו מתחלק ב-2 נגדיל את ה-offset אשר מצביע למפתח (edx).
4. במידה וה-Offset שווה ל-5 הוא יתאפס.
5. השוואה של cl מול תא במערך של Byte-ים ב-offset של מספר הריצות:

cl == Byte_array[Counter].

מערך ה-Byte-ים:

```
Stack[00000510]:0046F5D8 db  3Ah ; :
Stack[00000510]:0046F5D9 db 0FAh ; ·
Stack[00000510]:0046F5DA db 0CCh ; ¦
Stack[00000510]:0046F5DB db  32h ; 2
Stack[00000510]:0046F5DC db 0B0h ; ¦
Stack[00000510]:0046F5DD db 0D6h ; +
Stack[00000510]:0046F5DE db  50h ; P
Stack[00000510]:0046F5DF db 0A7h ; º
Stack[00000510]:0046F5E0 db 0BFh ; +
Stack[00000510]:0046F5E1 db 0C7h ; ¦
Stack[00000510]:0046F5E2 db 0FDh ; ²
Stack[00000510]:0046F5E3 db  55h ; U
```

הסקריפט הבא ב-Python מוצא את הסיסמא הדרושה:

```
xorkey = "\xFC\x08\x54\x38\x43"
cmpkey = "\x3a\xfa\xcc\x32\xb0\xd6\x50\xa7"

decoded = ""
i = 0
run_count = 0
for c in cmpkey:
    xored = 0
    xored = ord(c) ^ ord(xorkey[i])
    xored = xored / 2
    decoded += chr(xored)
    if (run_count % 2) == 0:
        i += 1
    if i == 5:
        i = 0
    run_count += 1

print decoded
```

שמירתו לקובץ והרצתו, תניב את הפלט הבא:

```
C:\Windows\system32\cmd.exe
C:\Users\Alien\Desktop>decode-stage1-rafael.py
cyb3rw4r

C:\Users\Alien\Desktop>_
```

לשם הספורט אני אציין כי יש עוד בדיקת **GetTickCount** לאחר בדיקת הסיסמא ולפני הדפסת הודעת ההצלחה.

אם כך, נראה שסיימנו...

```
C:\Windows\system32\cmd.exe
C:\Users\Alien\Downloads\All4Stages>stage1.exe
Please enter a user-name: ...
root
Hello root, please enter the password: ...
cyb3rw4r

Good news!


(C) Rafael Advanced Defense Systems Ltd. 2015
Congratulations!
You have solved this challenge!
For the next challenge, please send an e-mail to the following address: 0s8C4Ds7
MKA@gmail.com.
To be eligible to participate in the raffle, you must explain in the mail, how y
ou solved this challenge.


C:\Users\Alien\Downloads\All4Stages>
```

אתגר 3 - stage2.exe:

ידע דרוש לפתרון:

- פורמט קובץ PE
- Exception Handling
- Exception Anti-Debugging
- x86 Assembly

מאוורר Hash-ים או Hashing Fans

נצלול ישר ל-main שנראה כך:

```
01362980
01362980
01362980 ; Attributes: bp-based frame
01362980
01362980 ; int __cdecl main(int argc, const char **argv, const char **envp)
01362980 _main proc near
01362980
01362980 argc= dword ptr  8
01362980 argv= dword ptr  0Ch
01362980 envp= dword ptr  10h
01362980
01362980 push    ebp
01362981 mov     ebp, esp
01362983 push    offset unk_139BEC8
01362988 push    offset aEnterPassword ; "Enter password:"
0136298D call    _printf
01362992 add     esp, 8
01362995 call    sub_136CDA6
0136299A push    eax             ; FILE *
0136299B push    4Fh             ; int
0136299D push    offset byte_139BF18 ; char *
013629A2 call    _fgets
013629A7 add     esp, 0Ch
013629AA call    sub_1362780
013629AF push    5F048AF0h
013629B4 call    sub_13628B0
013629B9 add     esp, 4
013629BC mov     dword_139BF6C, eax
013629C1 push    offset sub_1362450
013629C6 call    dword_139BF6C
013629CC int     2Dh             ; Windows NT - debugging services: eax = type
013629CE xor     eax, eax
013629D0 pop     ebp
013629D1 retn
013629D1 _main endp
013629D1
```

sub_13628B0 הוא מימוש ידני של **GetProcAddress** אשר מקבל Hash של שם הפונקציה ועל פיו מוצא את כתובת הפונקציה (נקרא לו **MGPA** מעכשיו - ר"ת של **Manual GetProcAddress**[1]). ניתן לדלג על הפונקציה הזאת בכל פעם ורק להתבונן בערך ההחזר (eax) של הפונקציה בשביל לבדוק איזה מצביע (לפונקציה) הוחזר מהפונקציה.

[1] https://books.google.co.il/books?id=FQC8EPYy834C&pg=PA413&lpg=PA413&dq=manual+getprocaddress&source=bl&ots=BtknkBIfah&sig=YCE3B2G3pOm_vS_resSJXLq_XSk&hl=en&sa=X&ved=0CCgQ6AEwAjgKahUKEwjAx6yG5PnHAhWFxxQKHT0CCxM#v=onepage&q=manual%20getprocaddress&f=false

בקריאה הראשונה לפונקציה מוחזר המצביע ל-SetUnhandledExceptionFilter הקריאה ל-dword_139BF6C היא בעצם קריאה ל-SetUnhandledExceptionFilter לפי המצביע שזה עתה הוחזר מ-MGPA, הפונקציה שמותקנת כ-Handler של ה-Exception היא sub_1362450, זאת אומרת שברגע שיתבצע exception הפונקציה הזאת היא שמטפלת ב-Exception. ואכן אנו רואים את הפסיקה int 0x2D מיד לאחר התקנת ה-Exception Handler.[2]

SetUnhandledExceptionFilter עובד בצורה כזאת שלפני שהוא מעביר את ה-Exception ל-Handler שלו הוא בודק האם ה-Process רץ תחת Debugger, ואם כן אז הוא לא יעביר את השליטה ל-Handler אלא ל-Debugger, התנהגות שמבחינתנו אינה רצויה מכיוון שאנחנו לא מעוניינים לגרום להתערבות חיצונית כלשהי, אלא רוצים לתת לקובץ לרוץ בצורה טבעית.

איך SetUnhandledExceptionFilter בודק אם הוא רץ תחת Debugger? ברגע שמתבצע Exception מתבצעת קריאה ל-NtQueryInformationProcess עם הקבוע 0x7 (**ProcessDebugPort**) בצורה הבאה:

```
NtQueryInformationProcess(x, y, ProcessDebugPort, z);
```

במידה והקוד יבין כי אין Debugger הוא יעביר את ה-Exception ל-Exception Handler, במקרה שלנו: sub_1362450. בשביל לגרום לקוד להגיע ל-sub_1362450 בצורה טבעית נשים BP בתחילת sub_1362450, ונשים BP בתחילת NtQueryInformationProcess. ניתן לקוד לרוץ ולבצע את int 0x2D. ניתקל במספר קריאות ל-NtQueryInformationProcess, עלינו למצוא את הקריאה הרלוונטית אלינו, הקריאה צריכה להכיל את הקבוע 0x7 (**ProcessDebugPort**) בפרמטר השלישי, ניתן לראות זאת גם במחסנית. ברגע שמצאנו את הקריאה נבצע RUR ונחליף את ערך ההחזר (eax) מ-0 ל-1. ניתן לקוד לרוץ עד שיעצר בתחילת sub_1362450. החלק הראשון של הפונקציה יוצר קובץ בשם **"tmp0.X"** לכתיבה. לאחר מכן נגיע לקטע קוד הבא:

```
013624F3
013624F3 loc_13624F3:
013624F3 movsx   cx, byte ptr aTestingThis_0+3
013624FB movzx   edx, cx
013624FE push    edx
013624FF movsx   ax, byte ptr aTestingThis_0+2
01362507 movzx   ecx, ax
0136250A push    ecx
0136250B movsx   dx, byte ptr aTestingThis_0+1
01362513 movzx   eax, dx
01362516 push    eax
01362517 movsx   cx, byte ptr aTestingThis_0 ; "testing-this\n"
0136251F movzx   edx, cx
01362522 push    edx
01362523 mov     eax, [ebp+var_C]
01362526 movzx   ecx, word_1396630[eax*2]
0136252E push    ecx
0136252F call    sub_1362650
01362534 add     esp, 14h
01362537 mov     [ebp+var_D], al
0136253A mov     edx, [ebp+var_8]
0136253D push    edx             ; FILE *
0136253E movsx   eax, [ebp+var_D]
01362542 push    eax             ; int
01362543 call    _fputc
01362548 add     esp, 8
```

[2] http://fumalwareanalysis.blogspot.co.il/2011/09/malware-analysis-3-int2d-anti-debugging.html

loc_13624F3 יחזור על עצמו 0x1C00 פעמים. בכל ריצה הוא יקח את ארבעת ה-Byte-ים הראשונים של הסיסמא שהוכנסה וידחוף אותם לפונקציה **sub_1362650** יחד עם Word ממערך של word-ים קבוע אשר נמצא ב-word_1396630. הקריאה לפונקציה נראית ככה:

```
sub_1362650(const_word_array[counter] , password[0], password[1], password[2],
password[3]);
```

sub_1362650 נראה ככה:

```
01362650 sub_1362650 proc near
01362650
01362650 var_8= dword ptr -8
01362650 var_4= dword ptr -4
01362650 arg_0= word ptr  8
01362650 arg_4= word ptr  0Ch
01362650 arg_8= word ptr  10h
01362650 arg_C= word ptr  14h
01362650 arg_10= word ptr  18h
01362650
01362650 push    ebp
01362651 mov     ebp, esp
01362653 sub     esp, 8
01362656 movsx   eax, [ebp+arg_C]
0136265A movsx   ecx, [ebp+arg_C]
0136265E imul    eax, ecx
01362661 xor     eax, 5
01362664 mov     [ebp+arg_8], ax
01362668 movsx   edx, [ebp+arg_C]
0136266C movsx   eax, [ebp+arg_8]
01362670 imul    edx, eax
01362673 sub     edx, 3
01362676 mov     [ebp+arg_C], dx
0136267A movsx   ecx, [ebp+arg_0]
0136267E movsx   edx, [ebp+arg_4]
01362682 sub     ecx, edx
01362684 shl     ecx, 2
01362687 movsx   eax, [ebp+arg_8]
0136268B shl     eax, 2
0136268E mov     edx, 10h
01362693 sub     edx, eax
01362695 movsx   eax, [ebp+arg_C]
01362699 lea     edx, [edx+eax*4]
0136269C imul    edx, 3
0136269F sub     ecx, edx
013626A1 movsx   eax, [ebp+arg_10]
013626A5 shl     eax, 2
013626A8 sub     ecx, eax
013626AA movsx   edx, [ebp+arg_8]
013626AE mov     eax, 4
013626B3 sub     eax, edx
013626B5 movsx   edx, [ebp+arg_C]
013626B9 add     eax, edx
013626BB imul    eax, 0Ch
013626BE add     ecx, eax
013626C0 mov     [ebp+var_4], ecx
013626C3 mov     eax, [ebp+var_4]
013626C6 cdq
013626C7 mov     ecx, 0Ch
013626CC idiv    ecx
013626CE mov     [ebp+var_8], eax
013626D1 mov     eax, [ebp+var_8]
013626D4 add     eax, 20h
013626D7 movsx   edx, [ebp+arg_4]
013626DB xor     eax, edx
013626DD mov     esp, ebp
013626DF pop     ebp
013626E0 retn
013626E0 sub_1362650 endp
013626E0
```

אפשר להיכנס לניתוח האלגוריתם ואח"כ להבין איך להתקדם הלאה אך אני מעדיף להראות דרך יותר יצירתית לפתרון ה-Crackme הזה. מה שחשוב להבין מהפונקציה הזאת היא שהיא מעבדת את המידע שהיא מקבלת ככה שלא ניתן לשחזר את המידע המקורי (פונקציית Hash) ועל כל ריצה היא מחזירה Byte. זאת אומרת שהגדרת הפונקציה נראית ככה:

```
BYTE hashFunc(WORD w, BYTE a, BYTE b, BYTE c, BYTE d);
```

נבצע זום אאוט לפונקציה loc_13624F3, קטע הקוד מקבל את ערך ההחזר של **sub_1362650** וכותב אותו לקובץ שנפתח בתחילת הפונקציה **sub_1362450** בעזרת **Fputc**.

נמשיך ונגיע לקטע הקוד **loc_1362550**:

```
01362550
01362550 loc_1362550:
01362550 mov     ecx, [ebp+var_8]
01362553 push    ecx             ; FILE *
01362554 call    _fclose
01362559 add     esp, 4
0136255C push    0A498EAB6h      ; SetErrorMode hash
01362561 call    manual_getProcAddress
01362566 add     esp, 4
01362569 mov     funcSetErrorMode, eax
0136256E push    1
01362570 call    funcSetErrorMode
01362576 push    0EC0E4E8Eh      ; LoadLibraryA hash
0136257B call    manual_getProcAddress
01362580 add     esp, 4
01362583 mov     funcLoadLibraryA, eax
01362588 push    offset aTmp0_x_0 ; "tmp0.X"
0136258D call    funcLoadLibraryA
01362593 mov     [ebp+var_4], eax
01362596 push    7C0DFCAAh       ; GetProcAddress hash
0136259B call    manual_getProcAddress
013625A0 add     esp, 4
013625A3 mov     funcGetProcAddress, eax
013625A8 push    offset aTmp0_x_1 ; "tmp0.X"
013625AD call    _remove
013625B2 add     esp, 4
013625B5 cmp     [ebp+var_4], 0
013625B9 jz      short loc_1362634
```

```
013625BB push    offset aQualify ; "qualify"
013625C0 mov     edx, [ebp+var_4]
013625C3 push    edx
013625C4 call    funcGetProcAddress
013625CA mov     funcQualify, eax
013625CF cmp     funcQualify, 0
013625D6 jz      short loc_1362634
```

קטע הקוד מבצע **MGPA** לפונקציות **SetErrorMode**, **LoadLibraryA**, **GetProcAddress** לאחר מכן קורא להן. הקריאות המעניינות אותנו הן **LoadLibraryA**, ו-**GetProcAddress**.

הקוד מעביר ל-**LoadLibraryA** את הקובץ **"tmp0.X"** - הפונקציה מצפה לקבל Dll, במידה והקובץ לא יהיה בפורמט של **PE** הפונקציה לא תתבצע בהצלחה. ניתן להבין כי הקובץ הנכתב צריך להיות בפורמט של **PE**. אחרי טעינת הקובץ הקוד מבצע קריאה ל-**GetProcAddress** ומבקש את הפונקציה "qualify" שכנראה מיוצאת ע"י **"tmp0.X"**.

אם כך, הרעיון הוא להכניס 4 byte-ים ככה שאחרי מעבר של הפונקציה **sub_1362650** על כל מערך ה-Word-ים נקבל קובץ **PE** המייצא את הפונקציה "qualify".

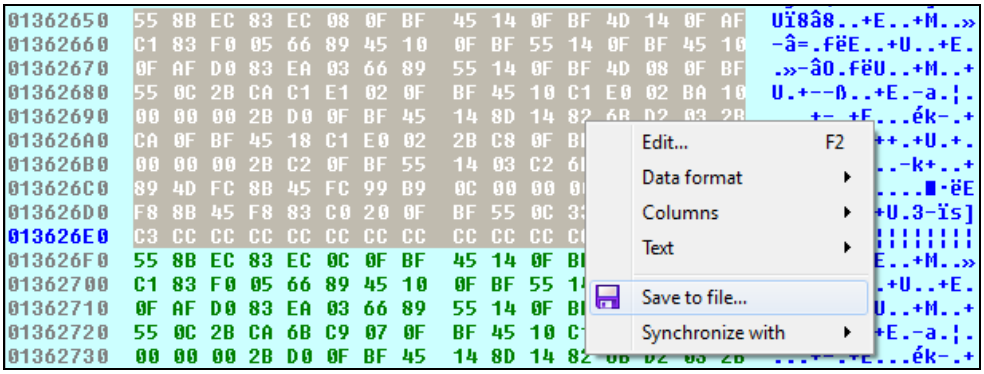
כל word במערך ה-word-ים מייצג byte של הקובץ הסופי, מהתבוננות קצרה ב-Header של קובץ **PE** ניתן לבצע המרה בין ה-Word הקיים במערך בתא במסוים לבין ב-Byte שהוא אמור לייצג. ההמרה נראית ככה (6 byte-ים ראשונים):

```
.data:01396630 word_1396630 dw 142h =0x4D
.data:01396630
.data:01396632 dw 181h =0x5A
.data:01396634 dw 223h =0x90
.data:01396636 dw 73h  =0x00
.data:01396638 dw 7Ch  =0x03
.data:0139663A dw 73h  =0x00
```

דרך הפעולה בה בחרתי היא ביצוע Bruteforce מחושב, הצעדים לביצוע הם כדלהלן:

1. שיחזור אלגוריתם ה-Hash.
2. מציאת ערכים קבועים במקומות קבועים ב-Format של PE עליהם אפשר להסתמך בשביל ה-Bruteforce.

הצעד הראשון: מה שנחמד מאוד בפונקציה **sub_1362650** הוא שהיא פונקציה שאינה תלויה בשום משתנה או מצביע בקוד. היא מקבלת ומבצעת בעזרת ה-Stack והאוגרים ולכן ניתן לבצע לה Dump לדיסק ולטעון אותה דרך תוכנית שאנחנו נכתוב מהדיסק, להעביר לה פרמטרים והיא תחזיר לנו תשובות, כמה נוח.



הצעד השני: כתיבת תכנית אשר בודקת סיסמאות ובדיקתה.

ביצוע כמה ריצות נסיון של תכנית ה-Bruteforce והוספת ערכים קבועים שערכם אחרי ההמרה ידוע מראש, למשל ארבעת ה-Byte-ים הראשונים:

```
0x0142 = 0x4D
0x0181 = 0x5A
0x0223 = 0x90
0x0073 = 0x00
```

אחרי הוספתם עדיין יתפסו הרבה False Positives ב-Bruteforce. לאחר התבוננות בתוצאות התכנית והשוואה מול קבצי PE אחרים ניתן לראות כי ה-Byte שמייצג space (0x20), למשל ב-String המוכר: "This program cannot be run in DOS mode", אינו נכון ומיוצג ע"י "!" (0x21). מכאן אפשר לקחת את ה-Offset של ה-byte 0x21 וללכת לאותו offset במערך ה-word-ים ולהציב את ההמרה כתנאי נוסף בתכנית - לאחר מעבר בפונקציית ה-Hash ה-Word 0x0013 צריך להיות שווה ל-0x20.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ .└...┘...ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	¸.....@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	F0	00	00	00	.....ð...
00000040	0E	1F	BA	0E	00	B4	09	CD	22	B8	01	4C	CD	22	54	68	♫ º♫.´.Í"¸.LÍ"Th
00000050	69	73	21	70	72	6F	67	72	61	6D	21	63	61	6E	6E	6F	is!program!canno
00000060	74	21	62	65	21	72	75	6E	21	69	6E	21	44	4F	53	21	t!be!run!in!DOS!
00000070	6D	6F	64	65	2F	0D	0D	0A	25	00	00	00	00	00	00	00	mode/...%.....

לא תקין

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ .└...┘...ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	¸.....@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00	.....à...
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	♫ º♫.´.Í!¸ LÍ!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is.program.canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t.be.run.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.....

תקין

מכאן נובע כי יש לנו 5 תנאים, 5 תנאים שהם מספקים את התכנית והופכות אותה למספיק חכמה, נשאר רק לכתוב פונקציה אשר מגדילה כל פעם את ה-dword המוכנס אליה, ושומרת עליו בין הערכים המתקבלים בקלט (רק תווים שיש להם קידוד ASCII ומתקבלים בקלט של קונסול, 0x20 - 0x7E).

Flow התוכנית:

1. קבלת סיסמת התחלה (0x20202020).
2. הרצת הסיסמא בפונקציית ה-Hash אל מול 5 התנאים שהצבנו.
3. במידה והסיסמא עומדת בכל התנאים - יש לנו סיסמא.
4. במידה ולא עומדת בתנאי - מעבר לסיסמא הבאה ע"י הגדלת הסיסמא ב-1 תוך כדי שמירתה בגבולות תווי ASCII. (אחרי 0x20202020 תבוא 0x20202021 ואחרי 0x2020207E תבוא 0x20202120 עד הסיסמא המקסימלית שהיא 0x7E7E7E7E)

התכנית שלנו שתבצע את ה-Bruteforce נראית ככה (כתוב ב-C++):

```
#include "stdafx.h"
#include "Windows.h"

typedef DWORD(*hashFunc)(WORD w, BYTE a, BYTE b, BYTE c, BYTE d);

#define ASCII_MIN                                   32
#define ASCII_MAX                                   126
#define PASS_INIT                                   0x20202020
#define PASS_MAX                                    0x7E7E7E7E
#define PASS_NO_MORE_AVAILABLE                      0xFFFFFFFF
#define FUNC_FILE                                   "func_raw"

DWORD incPassword(DWORD dwPass)
{
	if ((dwPass & 0xff) < ASCII_MAX)
	{
	dwPass += 1;
	}
	else if (((dwPass >> 8) & 0xff) < ASCII_MAX)
	{
		dwPass += 0x100;
		dwPass -= 94;
	}
	else if (((dwPass >> 16) & 0xff) < ASCII_MAX)
	{
		dwPass += 0x10000;
		dwPass -= 11776;
		dwPass -= 94;
	}
	else if (((dwPass >> 24) & 0xff) < ASCII_MAX)
	{
		dwPass += 0x1000000;
		dwPass -= 3014656;
		dwPass -= 11776;
```

```
                dwPass -= 94;
        }
        else if (dwPass == PASS_MAX)
        {
                return PASS_NO_MORE_AVAILABLE;
        }
        return dwPass;
}

int _tmain(int argc, _TCHAR* argv[])
{
        DWORD dwPasswordCounter = 0;
        HANDLE hFunc = CreateFileA(FUNC_FILE, FILE_READ_ACCESS, NULL, NULL, OPEN_EXISTING, NULL,
NULL);
        if (hFunc != INVALID_HANDLE_VALUE)
        {
                DWORD dwFileSize = GetFileSize(hFunc, NULL);
                LPVOID lpBuffer = VirtualAlloc(NULL, dwFileSize, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
                DWORD dwBytesRead;
                if (ReadFile(hFunc, lpBuffer, dwFileSize, &dwBytesRead, NULL))
                {

                        DWORD dwPass = PASS_INIT;
                        DWORD dwResult;
                        hashFunc hash = (hashFunc)lpBuffer;

                        while (dwPass != PASS_NO_MORE_AVAILABLE)
                        {
              dwResult = hash(0x0142, (dwPass >> 24) & 0xff, (dwPass >> 16) & 0xff, (dwPass >> 8)

                                                                                & 0xff, dwPass &
0xff);
                                dwResult &= 0xff;
                                if (dwResult == 0x4D)
                                {
                  dwResult = hash(0x0181, (dwPass >> 24) & 0xff, (dwPass >> 16) & 0xff, (dwPass >> 8)

                                                                        & 0xff, dwPass & 0xff)
                                        dwResult &= 0xff;
                                        if (dwResult == 0x5A)
                                        {
                    dwResult = hash(0x0223, (dwPass >> 24) & 0xff, (dwPass >> 16) & 0xff, (dwPass >>

                                                                        8) & 0xff, dwPass & 0xff);
                                                dwResult &= 0xff;
                                                if (dwResult == 0x90)
                                                {
                        dwResult = hash(0x0073, (dwPass >> 24) & 0xff, (dwPass >> 16) & 0xff, (dwPass >>

                                                                                8) & 0xff, dwPass
& 0xff);
                                                        dwResult &= 0xff;
                                                        if (dwResult == 0x00)
                                                        {
                                                                dwResult = hash(0x0013, (dwPass >> 24) &
0xff, (dwPass >> 16) & 0xff,

                                                dwPass >> 8) & 0xff, dwPass & 0xff);
                                                                dwResult &= 0xff;
                                                                if (dwResult == 0x20)
                                                                {
                                                                        printf("this is it - 0x%4x\n",
dwPass);
                                                                        dwPasswordCounter++;
                                                                }
                                                        }
                                                }
                                        }
                                }
                                dwPass = incPassword(dwPass);
```

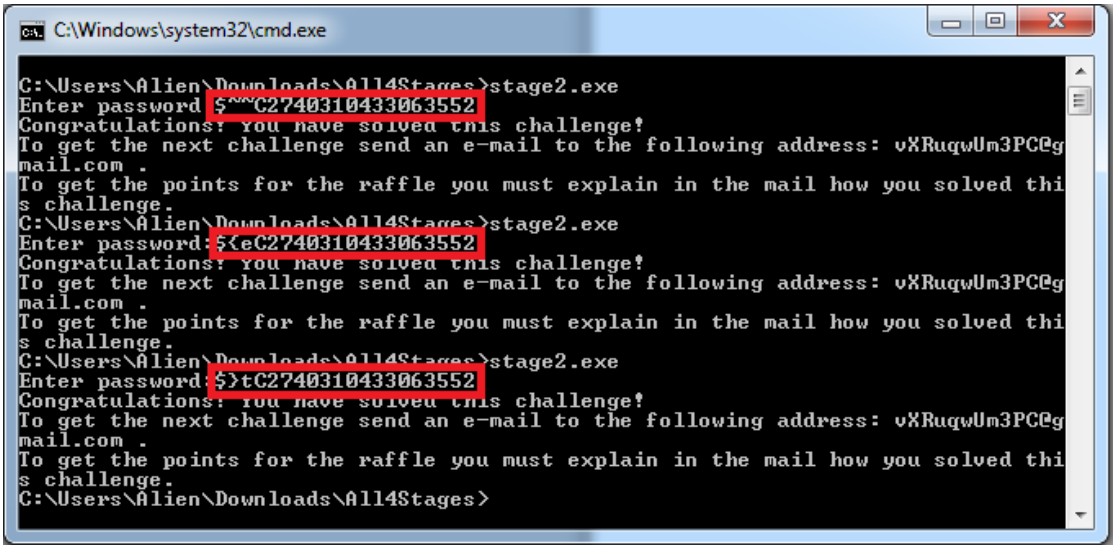
```
                        }
                        printf("%d passwords found", dwPasswordCounter);
                        getchar();
                }
        }
        return 0;
}
```

התכנית מוצאת 2209 סיסמאות שונות.

נשאר החלק האחרון, הטעינה של **"tmp0.X"** ע"י **LoadLibraryA** והקריאה ל-"qualify" מה-Export-ים של **"tmp0.X"**. ניתן לקוד לטעון את **"tmp0.X"** ונשים BP בכתובת שהתקבלה ע"י **GetProcAddress** ממציאת הפונקציה "qualify" כמה צעדים בתוך הפונקציה יראו לנו שיש ביצוע של **strncmp** בין הסיסמא שהכנסנו ב-Offset של 4 byte לבין ה-string "2740310433063552" ניתן להסיק כי הסיסמא שלנו תיהיה בנויה ככה: XXXX2740310433063552 כאשר XXXX הם ארבעת התוים שיגרמו ל-**"tmp0.X"** להפוך ל-Dll (ארבעת תוים אלה הם המשתנים היחידים בסיסמא, התכנית הנ"ל מבצעת Bruteforce ומציאה שלהם).

מדגמית נבדקו 3 סיסמאות בצורה מוצלחת (יש לשים לב כי מחליפים Byte שהוא אינו ה-Byte ה-3 בסיסמא כי אין לו השפעה באלגוריתם ה-Hash לכן הסיסמא הבאה: "$~?C" תתפוס לגבי כל ערך של "?" בין 0x20 ל-0x7E):

```
$~~C2740310433063552
${eC2740310433063552
$}tC2740310433063552
```



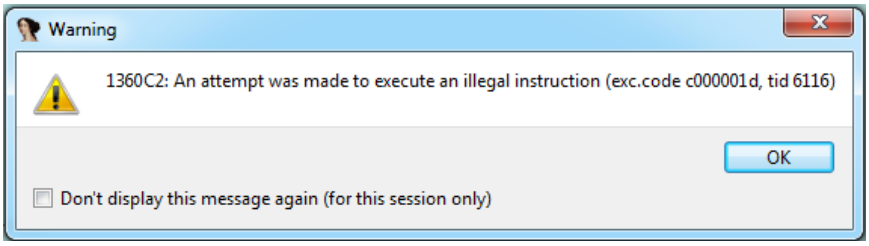
אתגר 4 - stage3.exe:

ידע דרוש לפתרון

- Exception Handling
- תהליך קימפול וצורת קימפול של Visual Studio
- Thread-ים ושליטה ב-Thread-ים ב-IDA
- x86 Assembly
- Windows API
- Exception Anti-Debugging
- Timing Anti-Debugging
- General Anti-Debugging
- Dynamic Function Pointers

איך להיאבד בדרך

נתחיל מלשים BP בתחילת הפונקציה לה-IDA קורא main_ ונריץ את הקובץ, נראה כי IDA אינו מגיע בכלל לפונקציה וזורק Exception הרבה לפני.



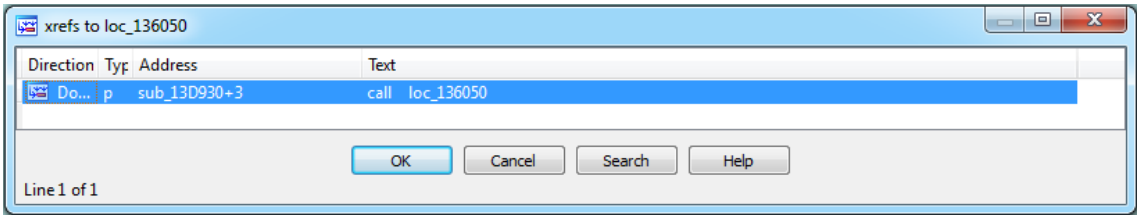
אכן כשנגיע ל-Instruction נראה כי הוא אינו תקין:

```
.text:001360B3 loc_1360B3:                             ; CODE XREF: .text:0013609F↑j
.text:001360B3                                         ; .text:001360AB↑j ...
.text:001360B3 movzx   ecx, byte ptr [ebp-1Dh]
.text:001360B7 cmp     ecx, 1
.text:001360BA jnz     short loc_1360BD
.text:001360BA ; ---------------------------------------------------------------------------
.text:001360BC db 2Dh
.text:001360BD ; ---------------------------------------------------------------------------
.text:001360BD
.text:001360BD loc_1360BD:                             ; CODE XREF: .text:001360BA↑j
.text:001360BD mov     edx, [ebp-8]
.text:001360BD ; ---------------------------------------------------------------------------
.text:001360C0 db 89h
.text:001360C1 db  55h ; U
.text:001360C2 ; ---------------------------------------------------------------------------
.text:001360C2 lock jmp short sub_1360CE
.text:001360C5 ; ---------------------------------------------------------------------------
.text:001360C5 ; START OF FUNCTION CHUNK FOR sub_1360CE
.text:001360C5
.text:001360C5 loc_1360C5:                             ; CODE XREF: sub_1360CE+26↓j
.text:001360C5 mov     eax, [ebp-10h]
.text:001360C8 add     eax, 1
.text:001360CB mov     [ebp-10h], eax
.text:001360CB ; END OF FUNCTION CHUNK FOR sub_1360CE
.text:001360CE
```

נתחיל בלהבין איך הגענו לקטע הקוד (loc_1360BD) הזה בכלל. נבצע XREF-ים עד שנבין איזה הסתעפות מביאה אותנו ל-Instruction הלא תקין.

```
.text:00136050 loc_136050:                          ; CODE XREF: sub_13D930+3↓p
.text:00136050 push    ebp
.text:00136051 mov     ebp, esp
.text:00136053 sub     esp, 20h
.text:00136056 mov     dword ptr [ebp-8], 0
.text:0013605D mov     dword ptr [ebp-14h], 0
.text:00136064 push    offset aNtqueryinforma       ; "NtQueryInformationProcess"
.text:00136069 push    offset ModuleName            ; "ntdll.dll"
.text:0013606E call    ds:GetModuleHandleA
.text:00136074 push    eax
.text:00136075 call    ds:GetProcAddress
.text:0013607B mov     [ebp-1Ch], eax
.text:0013607E push    0
.text:00136080 push    4
.text:00136082 lea     eax, [ebp-14h]
.text:00136085 push    eax
.text:00136086 push    1Fh
.text:00136088 call    ds:GetCurrentProcess
.text:0013608E push    eax
.text:0013608F call    dword ptr [ebp-1Ch]
.text:00136092 mov     [ebp-18h], eax
.text:00136095 cmp     dword ptr [ebp-18h], 0
.text:00136099 jz      short loc_1360A1
.text:0013609B mov     byte ptr [ebp-1Dh], 0
.text:0013609F jmp     short loc_1360B3
.text:001360A1 ; ---------------------------------------------------------------
.text:001360A1
.text:001360A1 loc_1360A1:                          ; CODE XREF: .text:00136099↑j
.text:001360A1 cmp     dword ptr [ebp-14h], 0
.text:001360A5 jnz     short loc_1360AF
.text:001360A7 mov     byte ptr [ebp-1Dh], 1
.text:001360AB jmp     short loc_1360B3
.text:001360AD ; ---------------------------------------------------------------
.text:001360AD jmp     short loc_1360B3
.text:001360AF ; ---------------------------------------------------------------
.text:001360AF
.text:001360AF loc_1360AF:                          ; CODE XREF: .text:001360A5↑j
.text:001360AF mov     byte ptr [ebp-1Dh], 0
.text:001360B3
.text:001360B3 loc_1360B3:                          ; CODE XREF: .text:0013609F↑j
.text:001360B3                                      ; .text:001360AB↑j ...
.text:001360B3 movzx   ecx, byte ptr [ebp-1Dh]
.text:001360B7 cmp     ecx, 1
.text:001360BA jnz     short loc_1360BD
.text:001360BA ; ---------------------------------------------------------------
.text:001360BC db 2Dh
.text:001360BD ; ---------------------------------------------------------------
.text:001360BD
.text:001360BD loc_1360BD:                          ; CODE XREF: .text:001360BA↑j
.text:001360BD mov     eax, [ebp-8]
.text:001360BD ; ---------------------------------------------------------------
.text:001360C0 db 89h
.text:001360C1 db  55h ; U
.text:001360C2 ; ---------------------------------------------------------------
.text:001360C2 lock jmp short sub_1360CE
```

נראה שהגענו ל-Instruction הזה מהפונקציה **loc_136050**. יש להבין עכשיו איך הגענו אליה, בשביל זה נבצע XREF לראות מאיפה היא נקראה.



לפונקציה sub_13D930 נבצע גם כן XREF, נגיע לטבלת פונקציות שנראית ככה:

```
.rdata:0013E16C dd offset _pre_cpp_init
.rdata:0013E170 dd offset sub_13D9FF
.rdata:0013E174 dd offset sub_13DA2D
.rdata:0013E178 dd offset sub_13DA43
.rdata:0013E17C dd offset sub_13DA15
.rdata:0013E180 dd offset sub_13DA21
.rdata:0013E184 dd offset sub_13D9DB
.rdata:0013E188 dd offset sub_13D9E7
.rdata:0013E18C dd offset sub_13D9F3
.rdata:0013E190 dd offset sub_13D8D0
.rdata:0013E194 dd offset sub_13D8F0
.rdata:0013E198 dd offset sub_13D910
.rdata:0013E19C dd offset sub_13D930
.rdata:0013E1A0 dd offset sub_13D940
.rdata:0013E1A4 dd offset sub_13D950
.rdata:0013E1A8 dd offset sub_13D970
.rdata:0013E1AC dd offset sub_13D990
.rdata:0013E1B0 dd offset sub_13D9D0
```

טבלת הפונקציות הזאת היא חלק מרוטינת האיתחול של קוד המקומפל ב-Visual Studio הממולאת בידי המשתמש ע"י פונקציות שהוא מגדיר כפונקציות אתחול הצריכות לרוץ לפני הקוד העיקרי. הטבלה נקראת מהפונקציה **_initterm**.[3]

אם אנחנו יוצאים מנקודת הנחה שאיננו מכירים את התנהגות זאת יש שתי דרכים להבין מאיפה הפונקציות נקראות:

1. הנחה של BP בתוך הפונקציה הראשונה בטבלה, הגעה ל-Entry Point של הקובץ (CTRL+E), הנחת BP, הרצה אל ה-BP בתחילת הקובץ, סימון Toggle Function Tracing והרצה עד שניתקל ב-BP השני. במעבר למסך ה-Trace יתקבל משהו בסגנון הזה:

```
00001D3C  .text:__security_init_cookie+37   call  ds:GetSystemTimeAsFileTime   __security_init_cookie call kernel32.dll:kernel32_GetSystemTimeAsFileTime
00001D3C  .text:__security_init_cookie+43   call  ds:GetCurrentProcessId       __security_init_cookie call kernel32.dll:kernel32_GetCurrentProcessId
00001D3C  .text:__security_init_cookie+4B   call  ds:GetCurrentThreadId        __security_init_cookie call kernel32.dll:kernel32_GetCurrentThreadId
00001D3C  .text:__security_init_cookie+53   call  ds:GetTickCount              __security_init_cookie call kernel32.dll:kernel32_GetTickCount
00001D3C  .text:__security_init_cookie+5F   call  ds:QueryPerformanceCounter   __security_init_cookie call kernel32.dll:kernel32_QueryPerformanceCounter
00001D3C  .text:__security_init_cookie+9A   retn                               __security_init_cookie returned to start+5
00001D3C  .text:__tmainCRTStartup+7         call  __SEH_prolog4                __tmainCRTStartup call __SEH_prolog4
00001D3C  .text:__SEH_prolog4+44            retn                               __SEH_prolog4 returned to __tmainCRTStartup+C
00001D3C  .text:__tmainCRTStartup+1B        call  ds:HeapSetInformation        __tmainCRTStartup call kernel32.dll:kernel32_HeapSetInformation
00001D3C  .text:__tmainCRTStartup+38        call  ds:InterlockedCompareExchange __tmainCRTStartup call kernel32.dll:kernel32_InterlockedCompareExchange
00001D3C  .text:__tmainCRTStartup+8A        call  _initterm_e                  __tmainCRTStartup call _initterm_e
00001D3C  .text:__tmainCRTStartup+BF        call  _initterm                    __tmainCRTStartup call _initterm
```

ניתן לראות בבירור את הקריאה האחרונה ל-**_initterm**.

2. הנחה של BP בתוך הפונקציה הראשונה בטבלה וביצוע RUR מיד ננחת בקטע קוד המסומן ע"י IDA כקטע קוד של **_initterm**.

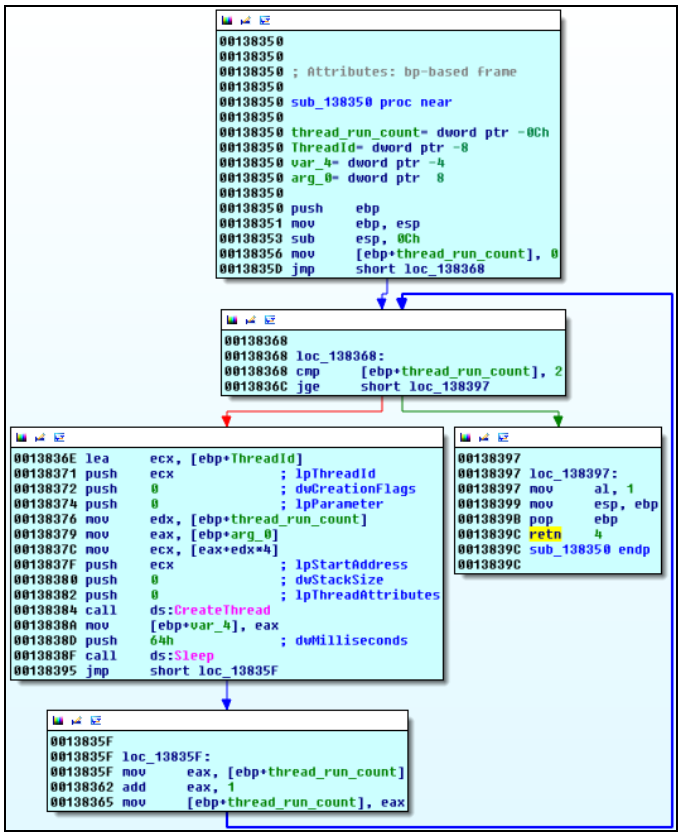
```
msvcr100.dll:71052630 loc_71052630:                    ; CODE XREF: msvcr100.dll:msvcr100__initterm+19↓j
msvcr100.dll:71052630 cmp     esi, [ebp+0Ch]
msvcr100.dll:71052633 jnb     short loc_71052642
msvcr100.dll:71052635 mov     eax, [esi]
msvcr100.dll:71052637 test    eax, eax
msvcr100.dll:71052639 jz      short loc_7105263D
msvcr100.dll:7105263B call    eax
msvcr100.dll:7105263D
msvcr100.dll:7105263D loc_7105263D:                    ; CODE XREF: msvcr100.dll:msvcr100__initterm+12↑j
msvcr100.dll:7105263D add     esi, 4
msvcr100.dll:71052640 jmp     short loc_71052630
msvcr100.dll:71052642 ; ---------------------------------------------------------------------------
msvcr100.dll:71052642
msvcr100.dll:71052642 loc_71052642:                    ; CODE XREF: msvcr100.dll:msvcr100__initterm+C↑j
msvcr100.dll:71052642 pop     esi
msvcr100.dll:71052643 pop     ebp
msvcr100.dll:71052644 retn
```

[3] http://www.codeguru.com/cpp/misc/misc/applicationcontrol/article.php/c6945/Running-Code-Before-and-After-Main.htm

איך לא להיאבד בדרך - שרשרת האתחול

פונקציית אתחול 1 (sub_136100):

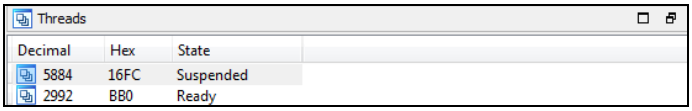
אז אחלה, הבנו איפה נפלנו ואיך הגענו לנפילה. עכשיו אפשר להניח BP-ים בכל הפונקציות הנמצאות בטבלת הפונקציות ולהתחיל להריץ. נריץ כל פונקציה וננסה להבין האם היא רלוונטית בשבילנו, אפשר להבין כי הפונקציה הרלוונטית הראשונה בשבילנו היא פונקציה sub_13D8F0 בתוכה נמצאת קריאה לפונקציה נוספת, אם נעקוב אחריה נגיע לפונקציה הבאה:



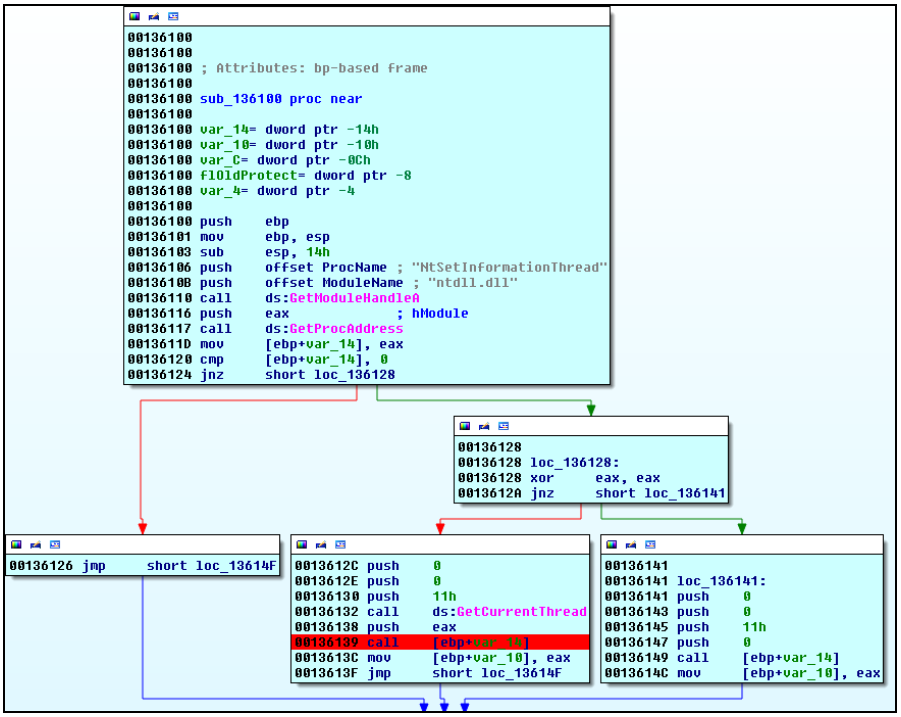
הפונקציה מקבלת מערך של מצביעים לפונקציות ומריצה את 2 הפונקציות הראשונות במערך. ניתן לחזור לפונקציה הקודמת ולעקוב אחרי הOffset שנדחף למחסנית (off_142E98) בשביל למצוא את המערך המועבר.

```
.data:00142E98 off_142E98 dd offset sub_136100          ; DATA XREF: sub_13D8F0+3↑o
.data:00142E9C dd offset loc_134E40
```

לשם שליטה ב-Thread-ים שיווצרו ע"י הקוד נניח BP בתחילת כל Thread חדש שיווצר וכאשר יווצר ה-Thread נעבור אליו ונבצע השהייה (Suspend) ל-Thread העיקרי ככה שלא יברח לנו ויאתחל עוד Thread-ים מה שעלול לשלוח אותנו לארץ האבדון בה Thread-ים אינם נשלטים ע"י IDA.



לצורך העניין התמונה הנ"ל ממחישה את העניין, ה-Thread העיקרי הרץ מ-sub_138350, בעל הערך 5884, וה-Thread שאני מבצע לו Debug כרגע והוא רץ מהפונקציה sub_136100 בעל הערך 2992. ניכנס לפונקציה הראשונה (sub_136100):



הפונקציה מתחילה בקבלת הכתובת של הפונקציה **NtSetInformationThread** ע"י קריאה ל-**GetProcAddress**. לאחר מכן אם **GetProcAddress** החזירה תשובה חיובית הפונקציה **NtSetInformationThread** תיקרא בצורה הבאה:

```
NtSetInformationThread(GetCurrentThread(), 0x11, 0, 0);
```

מתוך **ntnative.h**:

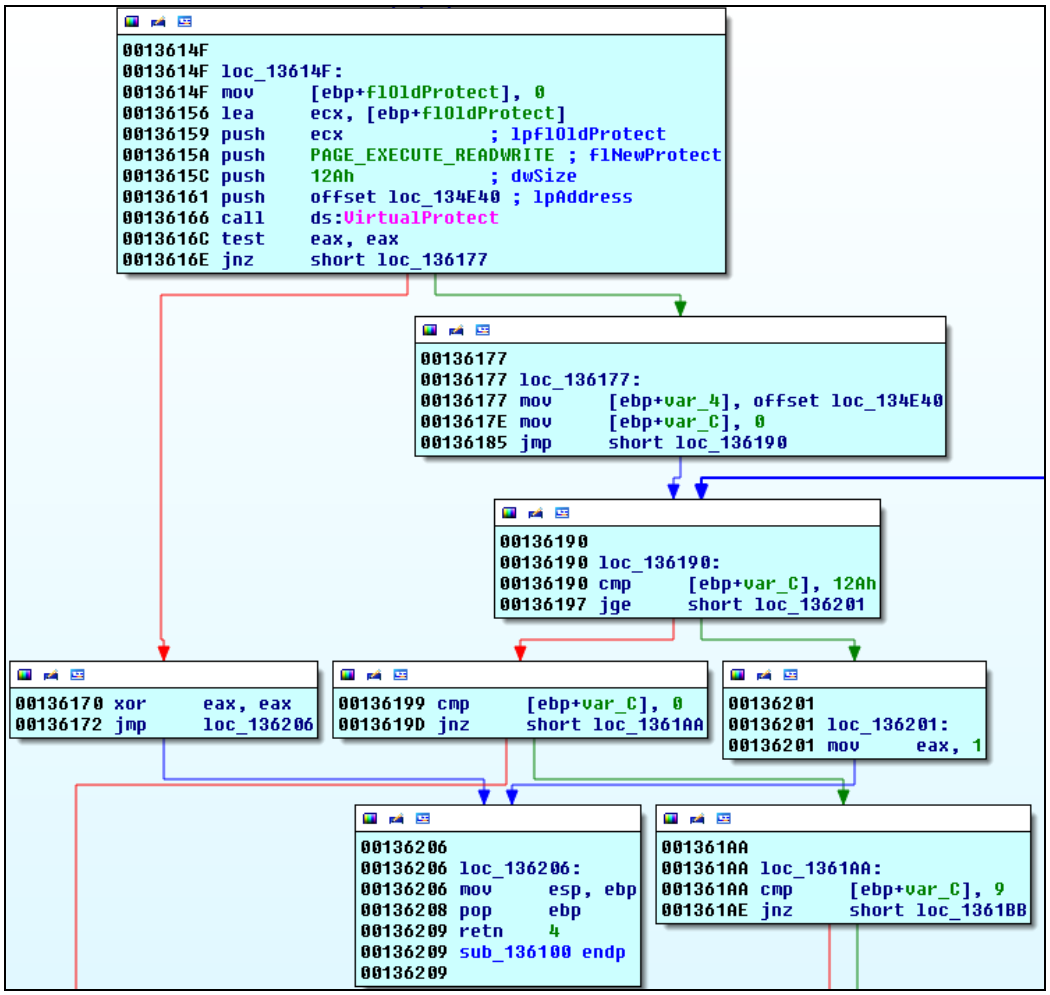
```
typedef enum  THREADINFOCLASS {       //Query Set
    ThreadBasicInformation,           // 0  Y N
    ThreadTimes,                      // 1  Y N
    ThreadPriority,                   // 2  N Y
    ThreadBasePriority,               // 3  N Y
    ThreadAffinityMask,               // 4  N Y
    ThreadImpersonationToken,         // 5  N Y
    ThreadDescriptorTableEntry,       // 6  Y N
    ThreadEnableAlignmentFaultFixup,  // 7  N Y
    ThreadEventPair,                  // 8  N Y
    ThreadQuerySetWin32StartAddress,  // 9  Y Y
    ThreadZeroTlsCell,                // 10 N Y
    ThreadPerformanceCount,           // 11 Y N
    ThreadAmILastThread,              // 12 Y N
    ThreadIdealProcessor,             // 13 N Y
    ThreadPriorityBoost,              // 14 Y Y
    ThreadSetTlsArrayAddress,         // 15 N Y
    ThreadIsIoPending,                // 16 Y N
    ThreadHideFromDebugger,           // 17 N Y
    ThreadIsCriticalInformation       // 18 Y Y //RtlSetThreadIsCritical, sizeof = 4, XP
} THREADINFOCLASS;
```

כשנמיר את הפרמטר 0x11 עפ"י ה-Enum הנתון, נקבל שהפרמטר המועבר הוא **ThreadHideFromDebugger** (0x11 = 17) מה שהפונקציה תעשה כאשר היא מקבלת פרמטר כזה הוא ניתוק ה-Context של ה-Thread מה-Debugger ככה שה-Thread ירוץ אך לא נוכל לעשות איתו כל מה ש-Debugger מאפשר לנו (למשל עצירה, שינוי מידע, התבוננות באוגרים וכו') וה-Debugger לא יכיר ב-Thread ולא יראה אותו ברשימת ה-Thread-ים. בשביל שה-Thread לא ינותק ניתן לשנות את הערך המוכנס ל-Stack מ-0x11 ל-0x2 למשל. ה-Stack שלנו יראה כך:

```
007EF7DC  FFFFFFFE
007EF7E0  00000002
007EF7E4  00000000
007EF7E8  00000000
```

בצורה כזאת ה-Thread לא יתנתק מה-Debugger.

בהמשך נבצע את אותה הפעולה בכל פעם שתיהיה קריאה לפונקציה **NtSetInformationThread** עם הפרמטר 0x11. לאחר שנפרדנו מה-Anti-debugging הראשון שהקוד הציע נמשיך עם ריצת הקוד ונגיע לקריאה ל-**VirtualProtect** שנראית ככה:



הפרמטר הראשון ש-VirtualProtect מקבלת הוא פרמטר של כתובת בזיכרון אותה הפונקציה תשנה, הכתובת שמועברת היא הכתובת של הפונקציה השניה שמועברת במערך הפונקציות לפונקציה sub_138350 שנמצאת ב-loc_134E40.

הפרמטר השלישי אשר מועבר ל-VirtualProtect הוא הקבוע 0x40 (המתפרש כ-PAGE_EXECUTE_READWRITE) בפועל מה שקורה הוא שהקוד משנה את ההגנה על מקטע הזיכרון ממה שיש לו כרגע ל-RWX(execute, read, write) (התנהגות זאת נפוצה בקרב Packer-ים וקוד אשר משנה את עצמו מכיוון שSection-ים של מידע של PE לרוב מתקמפלים עם הרשאות R או RW על ה-Section לעומת -Section-ים של קוד אשר מתקמפלים לרוב עם הרשאות RX) אפשר לנחש כי קטע הקוד מפה והלאה מבצע פיענוח של הפונקציה בשביל שיוכל להריץ אותה ב-Thread הבא.

נשחרר את ה-Thread לדרכו ואכן נראה כי הוא מבצע פיענוח לקוד שנמצא בכתובת המועברת בפרמטר ל-VirtualProtect.

```
.text:01104E40 loc_1104E40:
.text:01104E40
.text:01104E40 and     cl, [ebx+6A006AECh]
.text:01104E46 add     [edx+0], ebp
.text:01104E49 adc     [edi+38h], esi
.text:01104E4C loopne  near ptr loc_1104E5D+1
.text:01104E4E add     Handles[ebx], esp
.text:01104E54 int     3
.text:01104E55 sub     eax, 1113564h
.text:01104E5A rol     byte ptr [esi+10h], 1
.text:01104E5D
.text:01104E5D loc_1104E5D:
.text:01104E5D add     esp, ecx
.text:01104E5F sub     eax, 1113534h
.text:01104E64 jo      short near ptr loc_1104EAF+3
.text:01104E66 adc     [ecx], al
.text:01104E68 int     3
.text:01104E69 sub     eax, 11135D0h
.text:01104E6E jo      short loc_1104EBF
.text:01104E70 adc     [ecx], al
.text:01104E72 int     3
.text:01104E73 sub     eax, 1113540h
.text:01104E78 ror     byte ptr [edx+10h], 1
.text:01104E7B add     esp, ecx
.text:01104E7D sub     eax, 111357Ch
.text:01104E82 inc     eax
.text:01104E83 inc     ebp
.text:01104E84 adc     [ecx], al
.text:01104E86 int     3
.text:01104E87 sub     eax, 1113544h
.text:01104E8C lock push esi
.text:01104E8E adc     [ecx], al
.text:01104E90 int     3
.text:01104E91 sub     eax, 1113600h
.text:01104E96 inc     eax
.text:01104E97 push    esi
.text:01104E98 adc     [ecx], al
.text:01104E9A int     3
.text:01104E9B sub     eax, 11135F8h
```

```
.text:01104E40 loc_1104E40:                          ;
.text:01104E40                                       ;
.text:01104E40 push    ebp
.text:01104E41 mov     ebp, esp
.text:01104E43 push    0
.text:01104E45 push    1
.text:01104E47 push    0
.text:01104E49 call    ds:CreateMutexA
.text:01104E4F mov     Handles, eax
.text:01104E54 mov     dword_1113564, offset sub_11046D0
.text:01104E5E mov     dword_1113534, offset sub_1104C70
.text:01104E68 mov     dword_11135D0, offset sub_1104F70
.text:01104E72 mov     dword_1113540, offset sub_1104AD0
.text:01104E7C mov     dword_111357C, offset sub_1104540
.text:01104E86 mov     dword_1113544, offset sub_11056F0
.text:01104E90 mov     dword_1113600, offset sub_1105640
.text:01104E9A mov     dword_11135F8, offset sub_1102B20
.text:01104EA4 mov     dword_1113584, offset sub_11043D0
.text:01104EAE mov     dword_1113570, offset sub_11041A0
.text:01104EB8 mov     dword_11135F0, offset sub_1105750
.text:01104EC2 mov     dword_111354C, offset sub_1103A50
.text:01104ECC mov     dword_1113560, offset sub_1104000
.text:01104ED6 mov     dword_1113554, offset sub_1103900
.text:01104EE0 mov     dword_111355C, offset sub_1102DC0
.text:01104EEA mov     dword_1113530, offset sub_1103E10
.text:01104EF4 mov     dword_1113558, offset sub_1102CA0
.text:01104EFE mov     dword_1113568, offset sub_1103680
.text:01104F08 mov     dword_1113574, offset sub_11033C0
.text:01104F12 mov     dword_111353C, offset sub_1105120
.text:01104F1C mov     dword_1113594, offset sub_1102930
.text:01104F26 mov     dword_1113580, offset sub_1105000
.text:01104F30 mov     dword_1113578, offset sub_1103530
.text:01104F3A mov     dword_1113588, offset sub_1102A70
.text:01104F44 mov     dword_11135C8, offset sub_11044E0
.text:01104F4E mov     dword_1113590, offset sub_11028B0
.text:01104F58 mov     eax, Handles
.text:01104F5D push    eax
.text:01104F5E call    ds:ReleaseMutex
.text:01104F64 xor     eax, eax
```

אחרי

תכלית הפונקציה היא אתחול מערך של פונקציות, לפני ההתחלה היא קוראת ל-CreateMutexA ובסיום ל-ReleaseMutex, את הסיבה נבין רק בהמשך הניתוח אבל אפשר להסיק שזאת דרך לסנכרן את הקוד. ניתן לקוד לרוץ עד שיעצר ב-BP הבא הנמצא בפונקציה הבאה (sub_13D910) בטבלת פונקציות האתחול של **_initterm**.

פונקציית אתחול 2 (sub_13D910):

הפונקציה מבצעת את אותן הפעולות כמו פונקציית האתחול הראשונה עם שוני בפונקציה השניה שמועברת אליה בפרמטרים אך תכליתה זהה לקודמת לה - והיא השלמת אתחול אותו מערך של פונקציות.

פונקציית אתחול 3 (sub_13D930):

פונקציה זאת קוראת ל-loc_136050 הנראית ככה:

```
.text:00136050 loc_136050:                          ; CODE XREF: sub_13D930+3↓p
.text:00136050 push    ebp
.text:00136051 mov     ebp, esp
.text:00136053 sub     esp, 20h
.text:00136056 mov     dword ptr [ebp-8], 0
.text:0013605D mov     dword ptr [ebp-14h], 0
.text:00136064 push    offset aNtqueryinforma       ; "NtQueryInformationProcess"
.text:00136069 push    offset ModuleName            ; "ntdll.dll"
.text:0013606E call    ds:GetModuleHandleA
.text:00136074 push    eax
.text:00136075 call    ds:GetProcAddress
.text:0013607B mov     [ebp-1Ch], eax
.text:0013607E push    0
.text:00136080 push    4
.text:00136082 lea     eax, [ebp-14h]
.text:00136085 push    eax
.text:00136086 push    1Fh
.text:00136088 call    ds:GetCurrentProcess
.text:0013608E push    eax
.text:0013608F call    dword ptr [ebp-1Ch]
.text:00136092 mov     [ebp-18h], eax
.text:00136095 cmp     dword ptr [ebp-18h], 0
.text:00136099 jz      short loc_1360A1
.text:0013609B mov     byte ptr [ebp-1Dh], 0
.text:0013609F jmp     short loc_1360B3
.text:001360A1 ; ---------------------------------------------------------------
.text:001360A1
.text:001360A1 loc_1360A1:                          ; CODE XREF: .text:00136099↑j
.text:001360A1 cmp     dword ptr [ebp-14h], 0
.text:001360A5 jnz     short loc_1360AF
.text:001360A7 mov     byte ptr [ebp-1Dh], 1
.text:001360AB jmp     short loc_1360B3
.text:001360AD ; ---------------------------------------------------------------
.text:001360AD jmp     short loc_1360B3
.text:001360AF ; ---------------------------------------------------------------
.text:001360AF
.text:001360AF loc_1360AF:                          ; CODE XREF: .text:001360A5↑j
.text:001360AF mov     byte ptr [ebp-1Dh], 0
.text:001360B3
.text:001360B3 loc_1360B3:                          ; CODE XREF: .text:0013609F↑j
.text:001360B3                                      ; .text:001360AB↑j ...
.text:001360B3 movzx   ecx, byte ptr [ebp-1Dh]
.text:001360B7 cmp     ecx, 1
.text:001360BA jnz     short loc_1360BD
```

נראה מוכר? הרי זאת הפונקציה שגרמה לנו ל-Exception בקוד שאחריו לא יכולנו להמשיך בריצה הראשונה. הפונקצייה מוצאת את כתובת הפונקציה NtQueryInformationProcess ולאחר מכן קוראת לה בצורה הבאה:

```
NtQueryInformationProcess(GetCurrentProcess(), 0x1F, &out_buffer, 4, 0);
```

ה-Enum המועבר לפונקציה אינו מתועד ע"י Microsoft ולא נמצא ב-ntnative.h, אבל אפשר למצוא אותו בכל מני מדריכים על איך לבצע Anti-debugging[4]. ה-Enum נקרא **ProcessDebugFlags** וכאשר הוא מועבר לפונקציה היא מתשאלת את EPROCESS!NoDebugInherit ומחזירה את ערכו. נריץ את הקוד עד

[4] http://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide#NtQueryObject

אחרי הקריאה לפונקציה ונשנה את ערך ההחזר (eax) מ-0 ל-1. נמשיך בהרצה, נראה שהקוד לא יצור Exception ונגיע לקטע קוד אשר משתמש ב-**CreateThread**.

מנגנון ההגנה:

הדרך בה ממומש מנגנון הגנה במידה ונמצא Debugger היא כזאת:

במידה ונמצא Debugger תתבצע קפיצה לכתובת 0x001360C2 שממנה לא תתבצע ריצה תקינה מפני ש-assembly של intel x86 עובד בצורה כזאת שכל Instruction הוא בעל אורך משתנה בהתאם ל-Instruction. ריצה תקינה תתבצע מהכתובת 0x001360BD ובריצה זאת הכתובת 0x001360C2 לא תתפרש כתחילתה של פקודה אלא חלק מהפקודה שמתחילה ב-0x11360C0 (mov [ebp-0x10], edx) ובאורך 3 Byte-ים לכן כאשר תתבצע קפיצה ל-0x001360C2 המעבד יפרש את המידע הממוקם בכתובת בצורת Instruction שאי אפשר לבצע ולכן יבצע Exception.

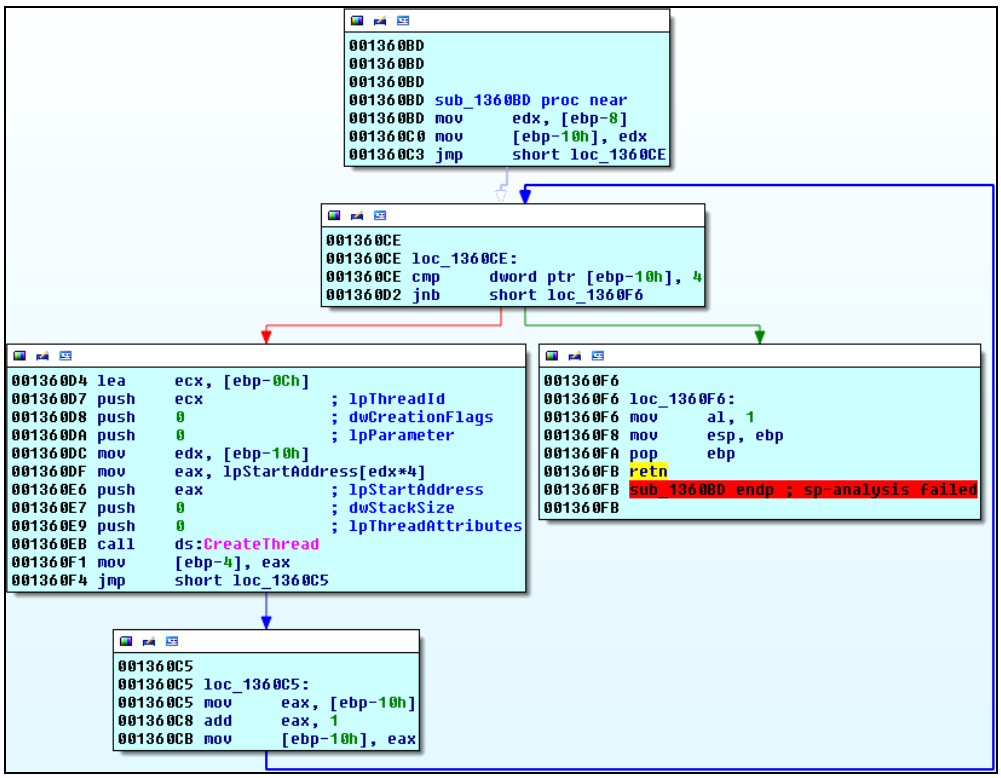
מצד ימין Disassembly תקין, מצד שמאל לא תקין.

```
.text:001360C2 lock jmp short sub_1360CE
```

```
.text:001360BD loc_1360BD:
.text:001360BD mov     edx, [ebp-8]
.text:001360C0 mov     [ebp-10h], edx
.text:001360C3 jmp     short sub_1360CE
```

תקין

הפונקציה אליה הגענו בשלב זה נראית ככה:

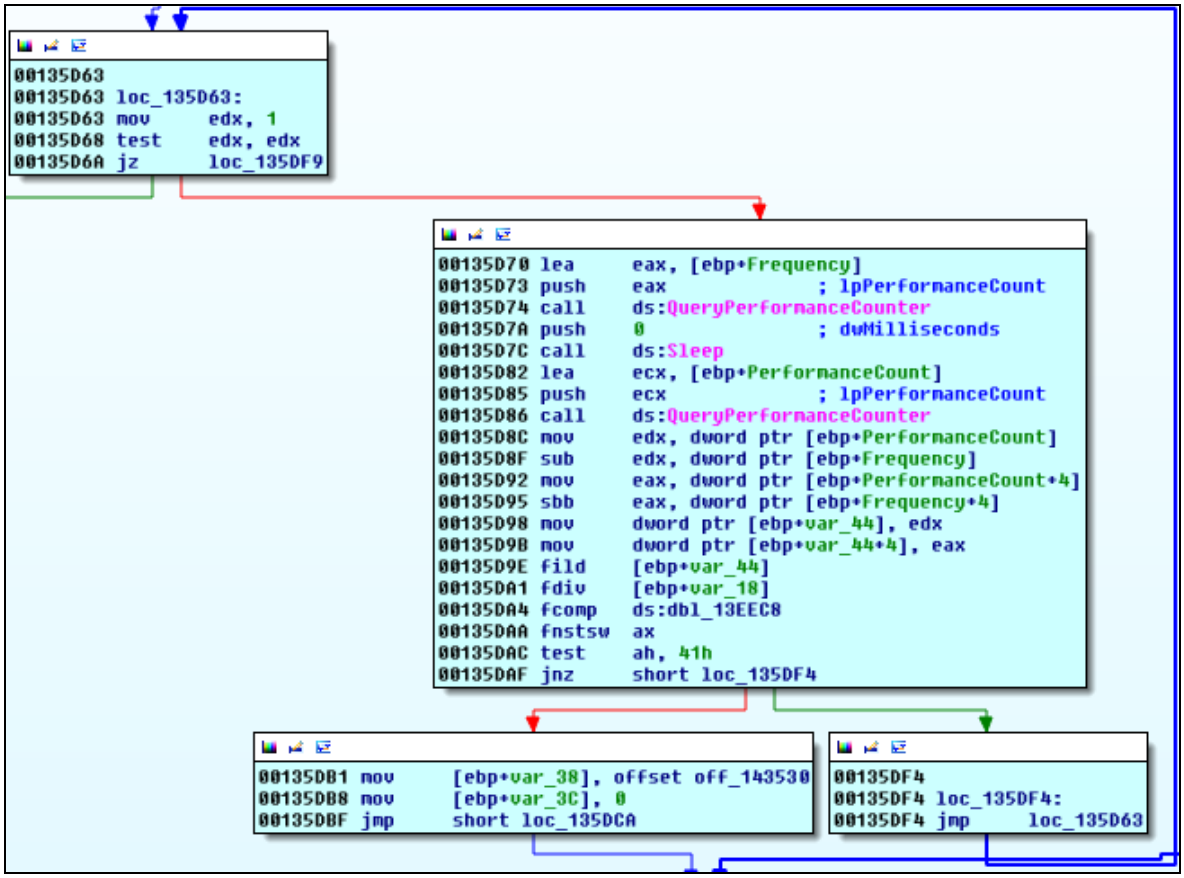


פעם שלישית **CreateThread** - גלידה? פחות. הפונקציה תיצור 4 Thread-ים, כל Thread בתבנית קבועה התואמת את ה-Pseudocode הבא:

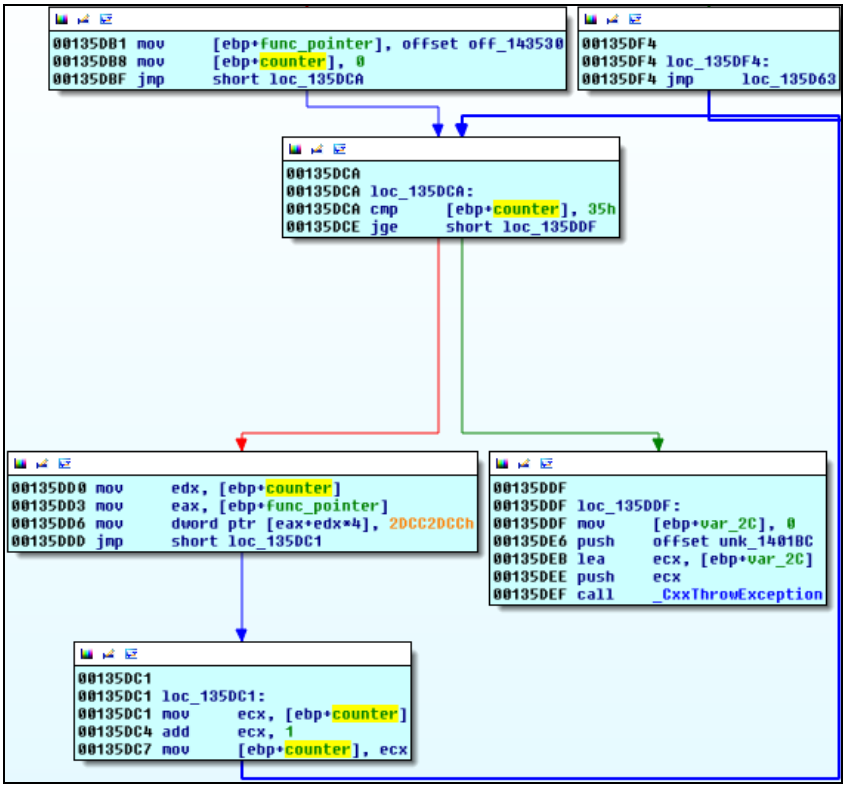
```
NtSetInformationThread(GetCurrentThread(), 0x11, 0, 0);
while(!isDebugged())
{
}
destroyFunction();
```

כאשר isDebugged הוא אחד מארבעת טכניקות למציאת Debugging עליהן יורחב בהמשך ו-destroyFunc הוא מנגנון ההגנה השני של הקובץ וכאשר אחת מטכניקות מציאת ה-Debugging מדווחת כי נמצא Debugger ה-Thread יעבור למצב בו הוא מפעיל את destroyFunc והורס (ע"י שכתוב) פונקציה הדרושה בשביל לפתור את האתגר.

דוגמא לאיך isDebugged נראה:



ו-destroyFunc (off_143530 הוא תא במערך שאותחל קודם המכיל מצביע לפונקציה - במידה וימצא Debugger הפונקציה הזאת היא שתיהרס):



מה שקורה בפועל אם התוכנית אינה רצה תחת Debugger כל ה-Thread-ים ירוצו בלולאה עד שימצאו Debugger, בשביל לעקוף את המנגנון ניתן לבצע Suspend לכל Thread לאחר יצירתו ולהמשיך הלאה.

4 טכניקות ל-isDebugged:

- **QueryPerformanceCounter** - הפונקציה מחזירה זמן מעבד ברזולוציה גבוהה. בגדול זאת טכניקת השוואת זמנים, הטכניקה תמצא את ה-Debugger רק אם נעצרים בין קריאה אחת לשניה ונותנים לזמן לעבור. לצורך התחמקות מהטכניקה הזאת נרוץ מהקריאה הראשונה עד אחרי השניה ללא עצירה.

- **GetTickCount** - הפונקציה מחזירה את הזמן במילישניות מאז הפעלת מערכת ההפעלה. שוב, זאת היא טכניקת השוואת זמנים, והטכניקה תמצא את ה-Debugger רק אם נעצרים בין קריאה אחת לשניה. לצורך התחמקות מהטכניקה הזאת נרוץ מהקריאה הראשונה עד אחרי השניה ללא עצירה.

- **IsDebuggerPresent** - הפונקציה מחזירה את ערך ה-**PEB!BeingDebugged**. לצורך התחמקות מהטכניקה הזאת אפשר לבחור בכל אחת מהדרכים הבאות:
  - ✓ לשנות את ערך ה-**PEB!BeingDebugged**
  - ✓ לשנות את ערך ההחזר (eax) מ-1 ל-0.
  - ✓ לשנות את ה-Zero Flag בהסתעפות הקוד הרלוונטית

- **NtQueryInformationProcess** - הפונקציה הוזכרה קודם לכן ולא יורחב עליה פה. לצורך התחמקות מהטכניקה הזאת אפשר לבחור בכל אחת מהדרכים הבאות:
  - ✓ לשנות את ערך ההחזר (eax) מ-0 ל-1.
  - ✓ לשנות את ה-Zero Flag בהסתעפות הקוד הרלוונטית

פונקציית אתחול 4 (sub_13D940):

הפונקציה קוראת ל-sub_132020 אשר נראית ככה (הפונקציה קצת יותר גדולה, רק החלק הרלוונטי נגזר):

```
0013205C call    ds:GetModuleHandleA
00132062 mov     [ebp+lpAddress], eax
00132065 lea     eax, [ebp+flOldProtect]
00132068 push    eax             ; lpflOldProtect
00132069 push    PAGE_READWRITE  ; flNewProtect
0013206B push    1000h           ; dwSize
00132070 mov     ecx, [ebp+lpAddress]
00132073 push    ecx             ; lpAddress
00132074 call    ds:VirtualProtect
0013207A push    1000h           ; Size
0013207F push    0               ; Val
00132081 mov     edx, [ebp+lpAddress]
00132084 push    edx             ; Dst
00132085 call    memset
0013208A add     esp, 0Ch
0013208D mov     al, 1
0013208F jmp     short loc_1320B4
```

מקטע הקוד מקבל את ה-BaseAddress שלו ע"י קריאה ל-**GetModuleHandle**, מבצע שינוי הרשאות מ-BaseAddress עד BaseAddress+0x1000 להרשאות RW ולאחר מכן רושם למקטע הזיכרון אפסים, בגדול פשוט מוחק את ה-Header **PE** וכל מה שאחריו מהזכרון. לא מצאתי סיבה מיוחדת לבצע את הפעולה הזאת.

פונקציות האתחול הנותרות אינן רלוונטיות אלינו ולכן מפה אפשר לקפוץ ל-main ולהתחיל לנתח אותה.

Main

תחילת הפונקציה נראית ככה:

```
00136216
00136216 loc_136216:
00136216 cmp     Handles, 0
0013621D jz      short loc_136228

0013621F cmp     hMutex, 0
00136226 jnz     short loc_13622A

0013622A
0013622A loc_13622A:                ; dwMilliseconds
0013622A push    0FFFFFFFFh
0013622C push    1                  ; bWaitAll
0013622E push    offset Handles     ; lpHandles
00136233 push    2                  ; nCount
00136235 call    ds:WaitForMultipleObjects
0013623B mov     eax, [ebp+argc]
0013623E sub     eax, 1
00136241 xor     ecx, ecx
00136243 mov     edx, 4
00136248 mul     edx
0013624A setc    cl
0013624D neg     ecx
0013624F or      ecx, eax
00136251 push    ecx
00136252 call    unknown_libname_14 ; Microsoft VisualC 2-11/net runtime
00136257 add     esp, 4
0013625A mov     [ebp+var_10], eax
0013625D mov     eax, [ebp+var_10]
00136260 mov     [ebp+var_4], eax
00136263 mov     [ebp+var_8], 0
0013626A jmp     short loc_136275

00136228
00136228 loc_136228:
00136228 jmp     short loc_136216
```

[WaitForMultipleObjects](#) מקבלת מערך של ה-Handle-ים ל-Mutex-ים שנוצרו קודם לכן ומחכה עד שיסגרו, בשלב הזה שניהם אמורים להיות כבר סגורים, תכלית מקטע הקוד הזה הוא לחכות עד שמערך הפונקציות שראינו קודם לכן יהיה מאותחל (ובמילה אחת - סינכרוניזציה).

החלק השני של הפונקציה נראה ככה:

```
00136275
00136275 loc_136275:
00136275 mov     edx, [ebp+argc]
00136278 sub     edx, 1
0013627B cmp     [ebp+var_8], edx
0013627E jge     short loc_13629F

0013629F
0013629F loc_13629F:
0013629F mov     off_14352C, esp
001362A5 mov     [ebp+var_C], 4
001362AC jmp     short loc_1362B7

00136280 mov     eax, [ebp+var_8]
00136283 mov     ecx, [ebp+argv]
00136286 mov     edx, [ecx+eax*4+4]
0013628A push    edx             ; Str
0013628B call    ds:atoi
00136291 add     esp, 4
00136294 mov     ecx, [ebp+var_8]
00136297 mov     edx, [ebp+var_4]
0013629A mov     [edx+ecx*4], eax
0013629D jmp     short loc_13626C

001362B7
001362B7 loc_1362B7:
001362B7 cmp     [ebp+var_C], 200h
001362BE jge     short loc_1362D1

0013626C
0013626C loc_13626C:
0013626C mov     ecx, [ebp+var_8]
0013626F add     ecx, 1
00136272 mov     [ebp+var_8], ecx

001362C0 mov     ecx, off_14352C
001362C6 sub     ecx, [ebp+var_C]
001362C9 mov     dword ptr [ecx], offset sub_1322D0
001362CF jmp     short loc_1362AE

001362D1
001362D1 loc_1362D1:
001362D1 mov     edx, off_14352C
001362D7 mov     dword ptr [edx-160h], offset sub_1320D0
001362E1 mov     eax, dword_1447D4
001362E6 add     eax, 1FFCh
001362EB mov     dword_1447D4, eax
001362F0 mov     ecx, [ebp+var_4]
001362F3 push    ecx
001362F4 call    off_140538
001362FA xor     eax, eax
001362FC mov     esp, ebp
001362FE pop     ebp
001362FF retn
001362FF _main endp
001362FF

001362AE
001362AE loc_1362AE:
001362AE mov     eax, [ebp+var_C]
001362B1 add     eax, 4
001362B4 mov     [ebp+var_C], eax
```

נחלק את הפונקציה לשתי לולאות:

loc_136275 - הלולאה בונה מערך של מספרים (בגודל של DWORD) המתקבלים בהעברה דרך **argv** בעזרת הפונקציה **atoi**.

loc_1362B7 - הלולאה בונה מערך (שתמיד יבנה באותה הצורה) בגודל של 0x200, כל תא בגודל של DWORD (128 תאים). את כלל התאים במערך הלולאה מאכלסת באותה הפונקציה (sub_1322D0) בהצצה מהירה - הפונקציה דומה בתכלית שלה ל-destroyFunc (הפונקציה הלוגית שהגדרנו קודם לכן).

בסיום ריצת הלולאה מוכנסת הפונקציה **sub_1320D0** לכתובת (endOfArray - 0x160) (המערך מאותחל מכתובת גבוהה וגודל לכתובות נמוכות), לאחר מכן נדחף מערך המספרים שנוצר ע"י **loc_136275** ויש מעבר למקטע הלוגי האחרון המתחיל עם הפונקציה ב-**off_143538**.

במעבר זריז על **sub_1320D0** ניתן להניח כי בשביל לסיים את האתגר יש להגיע לפונקציה הזאת שכן היא כוללת בתוכה את הפונקציה **puts**, הפונקציה בנוסף קוראת לפונקציה **RemoveVectoredExceptionHandler** ניתן להניח שתיהיה קריאה ל-**AddVectoredExceptionHandler** לפני הכניסה לפונקציה.

פונקציות ניתוב:

בשלב זה מועברת הריצה לטבלה בת 53 פונקציות, המצביע לאחת מהן הוא ב-**off_143538**:

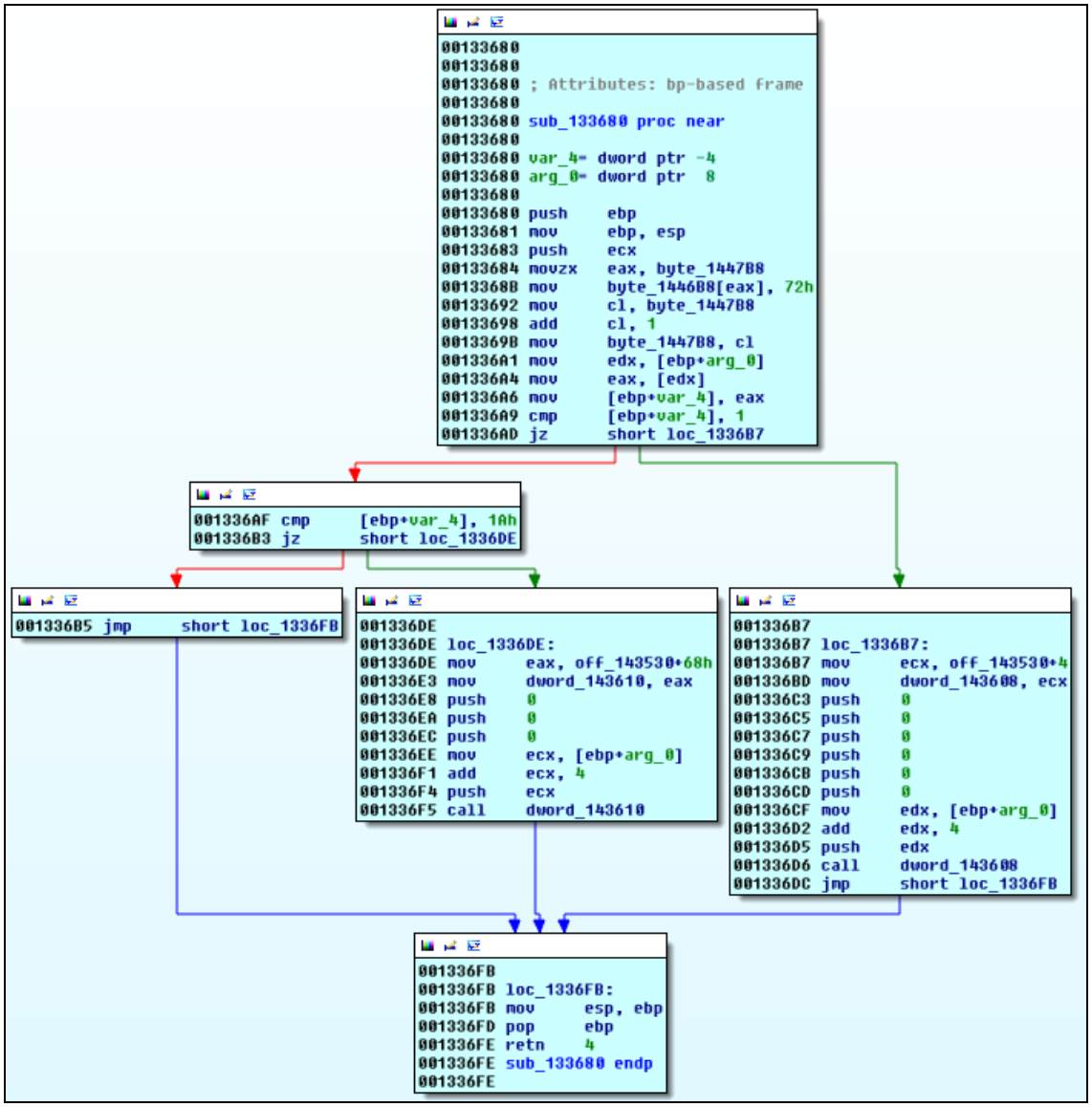
```
.data:00143530 off_143530 dd offset sub_133E10,offset sub_134C70,offset rf_argv_check_1,offset sub_135120
.data:00143530                                          ; DATA XREF: sub_1322D0+6↑o
.data:00143530                                          ; NtQueryInformationProcess_anti_dbg+E7↑o ...
.data:00143530 dd offset sub_134AD0,offset sub_1356F0,offset sub_133010,offset sub_133A50
.data:00143530 dd offset rf_vectored_exception_handler,offset sub_133900,offset sub_132CA0
.data:00143530 dd offset sub_132DC0,offset sub_134000,offset sub_1346D0,offset sub_133680
.data:00143530 dd offset sub_135530,offset sub_1341A0,offset sub_1333C0,offset sub_133530
.data:00143530 dd offset sub_134540,offset sub_135000,offset sub_1343D0,offset sub_132A70
.data:00143530 dd offset sub_132EC0,offset sub_1328B0,offset sub_132930,offset sub_1332A0
.data:00143530 dd offset sub_133F50,offset sub_133C20,offset sub_1348D0,offset sub_132590
.data:00143530 dd offset sub_134650,offset sub_135A60,offset sub_133710,offset sub_1354A0
.data:00143530 dd offset sub_1358A0,offset sub_1326E0,offset sub_1342F0,offset sub_1344E0
.data:00143530 dd offset sub_133870,offset sub_134F70,offset sub_134780,offset rf_argv_check_3
.data:00143530 dd offset sub_133130,offset rf_argv_check_2,offset sub_1337C0,offset sub_1353E0
.data:00143530 dd offset sub_134320,offset rf_argv_check_min_1,offset sub_133D60,offset sub_132B20
.data:00143530 dd offset sub_134960,offset sub_135640
```

ע"י מעבר על הפונקציות בצורה ויזואלית ניתן לראות כי הפונקציה הכי שונה היא הפונקציה שנקראת בתמונה rf_vectored_exception_handler (sub_133AE0), ניתן להניח שהיא הפונקציה שתוביל אותנו להגעה לפונקציה המיוחלת **sub_1320D0**.

ע"י ניתוח כמה מהפונקציות במערך ניתן להסיק כי תבנית הפונקציות היא קבועה ונראית ככה (בנוסף להסבר האלגוריתמי צירפתי את הפונקציה sub_133680 בה ניתן לראות את האלגוריתם המתואר)

1. קבלת מספר (אותו מספר ממערך המספרים שהוכנס בעזרת **argv**).
2. הוספת אות ל-String גלובלי (נמצא ב-0x001446B8) (בפונקציה המדגמית: 0x13368B)

3. ביצוע מניפולציה על המספר (אופציונלי - מהפונקציות שראיתי המניפולציה מסתכמת בהחסרת מספר קבוע מהמספר שהוכנס) (בפונקציה המדגמית: לא קיים)
4. השוואת המספר מול מספר קבוע או מול Jump table (בפונקציה המדגמית: 0x1336A9)
5. דחיפת המספר הבא במערך המספרים כפרמטר (בפונקציה המדגמית: 0x1336EE/0x1336CF)
6. לפי התוצאה ניתוב לפונקציה הבאה במערך הפונקציות. (בפונקציה המדגמית: 0x1336D6/0x1336F5)



המשימה הבאה שלנו היא לבנות ניתוב בין הפונקציות כך שנגיע מהפונקציה הראשונה עד הפונקציה האחרונה (rf_vectored_exception_handler).

```
offset                function   byte    xrefs

off_143534 dd offset sub_134C70 0x64    sub_check_-5    sub_1358A0  sub_1341A0
off_14353C dd offset sub_135120         sub_133D60
off_143544 dd offset sub_1356F0
off_143548 dd offset sub_133010 0x30    sub_check_-5
off_14354C dd offset sub_133A50
off_143554 dd offset sub_133900         sub_133010  sub_1341A0
off_143558 dd offset sub_132CA0 0x36    sub_133010  sub_133D60  sub_132590
off_14355C dd offset sub_132DC0         sub_1341A0
off_143560 dd offset sub_134000         sub_135A60
off_143564 dd offset sub_1346D0
off_143568 dd offset sub_133680 0x72    sub_134C70  check_3rd_argv
off_14356C dd offset sub_135530         sub_133010  sub_133D60
off_143570 dd offset sub_1341A0 0x32    sub_135A60
off_143574 dd offset sub_1333C0         sub_133D60  sub_132590
off_143578 dd offset sub_133530         sub_133010  sub_1358A0
off_14357C dd offset sub_134540         sub_133D60
off_143580 dd offset sub_135000         sub_133010
off_143584 dd offset sub_1343D0         sub_check_-5    sub_132930  sub_132590
off_143588 dd offset sub_132A70
off_14358C dd offset sub_132EC0         sub_132930
off_143590 dd offset sub_1328B0         sub_1341A0
off_143594 dd offset sub_132930 0x58    sub_132CA0
off_143598 dd offset sub_1332A0
off_1435A0 dd offset sub_133C20         sub_1341A0
off_1435A4 dd offset sub_1348D0
off_1435A8 dd offset sub_132590 0x73    sub_133D60
off_1435AC dd offset sub_134650
off_1435B0 dd offset sub_135A60 0x39    sub_1358A0
off_1435B4 dd offset sub_133710
off_1435B8 dd offset sub_1354A0         sub_132590
off_1435BC dd offset sub_1358A0 0x76    sub_132590
off_1435C4 dd offset sub_1342F0
off_1435C8 dd offset sub_1344E0
off_1435CC dd offset sub_133870         sub_132930  sub_135A60
off_1435D0 dd offset sub_134F70         sub_133010
off_1435D4 dd offset check_min_5_argv
off_1435DC dd offset sub_133130         sub_1358A0
off_1435E4 dd offset sub_1337C0         sub_132590
off_1435E8 dd offset sub_1353E0
off_1435EC dd offset sub_134320         sub_133010      sub_1358A0
off_1435F4 dd offset sub_133D60 0x64    sub_check_-5    sub_132930
off_1435F8 dd offset sub_132B20
off_1435FC dd offset sub_134960         sub_1358A0
off_143600 dd offset sub_135640
```

נתחיל במיפוי הנתיבים שאי אפשר לשנות אותם - זאת אומרת פונקציות בשתי קצוות הניתוב, מהסוף או מההתחלה שאנו יודעים בוודאות לאיזה פונקציה לנתב אותן - או מאיזה פונקציה לקרוא להן (ע"י ביצוע XREF-ים ומיפוי הקריאות בפונקציות ניתן להבין - פונקציות ניתוב 1 עד 3 ופונקציות ניתוב 5- עד rf_vectored_exception_handler שהיא האחרונה הן פונקציות הנופלות תחת הקטגוריה).

בשלב זה אנחנו מכירים 8 פונקציות שניהיה מחויבים לנתב דרכן וחסרות לנו כמה חוליות ניתוב בדרך, עכשיו הגיע הזמן לבצע קצת עבודה שחורה. יש למפות מאיפה פונקציה 5- נקראת, בנוסף למפות את כל הפונקציות החסרות, ולהבין מאיפה כל אחת מהן נקראת עד שנוכל להשלים את החוליות החסרות בין פונקציה 3 ל-5-.

כל המיפוי נעשה ידנית ע"י XREF-ים, התוצר הוא הטבלה משמאל.

בשביל להשלים את החוליות החסרות נדרשתי למפות 10 פונקציות אחריהן הצלחתי לחבר בין ההתחלה לסוף ולבנות ניתוב.

עכשיו נותר רק לחשב עבור כל פונקציה שאנו עוברים דרכה את הערך עלינו להכניס אליה בשביל לעבור בהסתעפות המתאימה לנו. מתקבלת הטבלה הבאה:

פונקציה	ערך מעבר
0x1355E0	44
0x132530	42
0x1352A0	14
0x133680	1
0x134C70	41
0x134780	36
0x1326E0	0
0x133E10	27
0x133F50	48
0x135750	8
0x133AE0	-

ניתוב אחרון:

```
00133AE0
00133AE0
00133AE0 ; Attributes: bp-based frame
00133AE0
00133AE0 rf_vectored_exception_handler proc near
00133AE0
00133AE0 var_4= dword ptr -4
00133AE0 arg_0= dword ptr  8
00133AE0
00133AE0 push    ebp
00133AE1 mov     ebp, esp
00133AE3 push    ecx
00133AE4 movzx   eax, byte_1447B8
00133AEB mov     byte_1446B8[eax], 36h
00133AF2 mov     cl, byte_1447B8
00133AF8 add     cl, 1
00133AFB mov     byte_1447B8, cl
00133B01 mov     off_14360C, esp
00133B07 mov     esp, dword_1447D4
00133B0D push    offset Handler  ; Handler
00133B12 push    0               ; First
00133B14 call    ds:AddVectoredExceptionHandler
00133B1A call    sub_1324B0
00133B1F push    offset Handler  ; Handle
00133B24 call    ds:RemoveVectoredExceptionHandler
00133B2A mov     esp, off_14360C
00133B30 mov     edx, [ebp+arg_0]
00133B33 mov     eax, [edx]
00133B35 mov     [ebp+var_4], eax
00133B38 mov     ecx, [ebp+var_4]
00133B3B sub     ecx, 9
00133B3E mov     [ebp+var_4], ecx
00133B41 cmp     [ebp+var_4], 26h ; switch 39 cases
00133B45 ja      loc_133BDE       ; jumptable 00403B55 default case
```

ניתן לשים BP על הפונקציה **AddVectoredExceptionHandler** ולהריץ, נראה כי אנו באמת מגיעים אל ה-BP שלנו, ה-String שמתקבל הוא **"WMardD87Qy6"** - כנראה מפתח הפיענוח להודעת ההצלחה, הפונקציה מגדירה את sub_132480 כ-Exception Handler, IDA יודע לקרוא לו ככה בעצמו (Handler), לאחר מכן יש קריאה לפונקציה sub_13B24B0 אשר מבצעת Exception והריצה מועברת ל-Handler משם מתבצע מעבר (כמובן רק אם נעביר את ה-Exception לאפליקציה ולא נטפל בו בעצמנו) לפונקציה המיוחלת שלנו (**sub_1320D0**) עכשיו רק נותר לבדוק אם באמת הצלחנו.

```
C:\Windows\system32\cmd.exe

C:\Users\Alien\Downloads\All4Stages>stage3.exe 44 42 14 1 41 36 0 27 48 8
Congratulations! You have solved this challenge!Send an e-mail to the following
address: Da6ogI9bObMg@gmail.com.To get the points for the raffle you must explai
n in the mail how you solved this challenge.

C:\Users\Alien\Downloads\All4Stages>_
```

נראית כמו התוצאה הרצויה.

תודות

תודה לחברה שלי, שרון שתרמה מזמנה לקרוא את המאמר ותמכה בי לאורך כתיבתו.

תודה לשחק שלו שנתן מזמנו בשביל לקרוא טיוטה ארוכה מאוד.

תודה ל-Digital Whisper שפרסמו את המאמר וממשיכים להזרים מאמרים באיכות גבוהה לאורך השנים.

ליצירת קשר ניתן לפנות לכתובת: d.pshoul@gmail.com

נספח קישורים ופונקציות

- RegOpenKeyEx - הפונקציה מקבלת מפתח ל-Registry ופותחת אותו, מחזירה Handle למפתח:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms724897(v=vs.85).aspx

- RegQueryValueEx - הפונקציה מקבלת Handle למפתח פתוח, נתיב + ערך לתשאל ומחזירה את המידע בתוך הערך:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms724911(v=vs.85).aspx

- CryptBinaryToString - הפונקציה מקבלת מידע וצורת המרה, מחזירה מידע מומר:
https://msdn.microsoft.com/en-us/library/windows/desktop/aa379887(v=vs.85).aspx

- Base64 encoding - שיטת קידוד נפוצה, תומכת בקידוד מידע בינארי ל-ASCII:
https://en.wikipedia.org/wiki/Base64

- Strncmp - הפונקציה מקבלת 2 String-ים ומשווה ביניהם:
https://msdn.microsoft.com/en-us/library/eywx8zcx.aspx

- GetTickCount - הפונקציה מחזירה את הזמן במילישניות מאז הפעלת מערכת ההפעלה:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms724408(v=vs.85).aspx

- GetProcAddress - הפונקציה מקבלת Handle לDll ושם של פונקציה ומחזירה את הכתובת של הפונקציה:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms683212(v=vs.85).aspx

- Fputc - הפונקציה מקבלת Byte ו-Handle לקובץ וכותב את הByte- לקובץ:
https://msdn.microsoft.com/en-us/library/yah67377.aspx

- NtSetInformationThread - הפונקציה מקבלת Handle לThread ופרמטרים נוספים המורים לה על שינוים שעליה לבצע על ה-Thread, הפונקציה מתועדת בצורה חלקית ע"י Microsoft:
https://msdn.microsoft.com/en-us/library/windows/hardware/ff567101(v=vs.85).aspx

- VirtualProtect - הפונקציה מקבלת כתובת וקבוע הרשאות, הפונקציה מכילה את ההרשאות המבוקשות על ה-Page של הכתובת:
https://msdn.microsoft.com/en-us/library/windows/desktop/aa366898(v=vs.85).aspx

- NtQueryInformationProcess - הפונקציה מקבלת Handle ל-Process הנוכחי ופרמטרים נוספים המורים לה על מידע עליו נרצה לעשות שאילתה, הפונקציה מתועדת בצורה חלקית ע"י Microsoft:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms684280(v=vs.85).aspx

- IsDebuggerPresent - הפונקציה בודקת האם יש Debugger המסתכל על הProcess, מחזירה True או False):
https://msdn.microsoft.com/en-us/library/windows/desktop/ms680345(v=vs.85).aspx

- QueryPerformanceCounter - הפונקציה מחזירה את ערך הPerformance Counter הנוכחי:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms644904(v=vs.85).aspx

- GetModuleHandle - הפונקציה מקבל NULL או שם של מודול ומחזירה את כתובת הטעינה שלו - BaseAddress):
https://msdn.microsoft.com/en-us/library/windows/desktop/ms683199(v=vs.85).aspx
- WaitForMultipleObjects - הפונקציה מחכה זמן מסוים או זמן אינסופי לאחד או יותר מהאובייקטים שהועברו לה במערך):
https://msdn.microsoft.com/en-us/library/windows/desktop/ms687025(v=vs.85).aspx

- Atoi - הפונקציה מקבלת מידע מספרי ב-String וממירה אותו לערך מספרי, ר"ת של Ascii to int):
https://msdn.microsoft.com/en-us/library/yd5xkb5c.aspx

- SetUnhandledExceptionFilter - הפונקציה מקבלת פונקציה ומתקינה אותה כ-UnhandledExceptionFilter, במידה ויש Debugger הפונקציה לא תעביר את השליטה לפונקציה המותקנת):
https://msdn.microsoft.com/en-us/library/windows/desktop/ms680634(v=vs.85).aspx

- AddVectoredExceptionHandler - הפונקציה מקבלת פונקציה ורושמת אותה VectoredExceptionHandler:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms679274(v=vs.85).aspx

- RemoveVectoredExceptionHandler - הפונקציה מקבלת פונקציה ומורידה אותה מה-VectoredExceptionHandler:
https://msdn.microsoft.com/en-us/library/windows/desktop/ms680571(v=vs.85).aspx

LoadLibraryA - הפונקציה מקבלת נתיב של קובץ Dll וטוענת אותו:

https://msdn.microsoft.com/en-us/library/windows/desktop/ms684175(v=vs.85).aspx

- OutputDebugStringA - הפונקציה מדפיסה String ל-Debugger:

https://msdn.microsoft.com/en-us/library/windows/desktop/aa363362(v=vs.85).aspx

- CreateMutexA - הפונקציה יוצרת אובייקט של סינכרוניזציה - Mutex, ReleaseMutex - הפונקציה משחררת Mutex:

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682411(v=vs.85).aspx

- CreateThread - הפונקציה יוצרת Context ריצה נוסף המתבטא ביצירת Thread מקביל:

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682453(v=vs.85).aspx

- PE - פורמט קבצי ההרצה ב-Windows:

https://en.wikipedia.org/wiki/Portable_Executable

- EPROCESS - מבנה נתונים של ה-Kernel המתאר Process:

http://www.nirsoft.net/kernel_struct/vista/EPROCESS.html

- PEB - מבנה נתונים הנמצא בכל Process השומר מידע שימושי על ה-Process:

https://msdn.microsoft.com/en-us/library/windows/desktop/aa813706(v=vs.85).aspx

- ntnative.h - קובץ Header המתאר מבני נתונים של פונקציות Native:

http://mytoybox.googlecode.com/svn/trunk/SetEvent/ntnative.h

- Argv - העברת משתנים לקובץ הרצה ע"י שורת פקודה - למשל בהרצה: 1.exe param_a param_b:

https://en.wikipedia.org/wiki/Entry_point#C_and_C.2B.2B