digital whisper
Technologic Papers

פתרון אתגר השב"כ 2017 - מחקר

מאת D4d ותומר זית.

הקדמה

ב-27.4.2017 השב"כ פרסם אתגרים בכדי לגייס אנשים למחקר ופיתוח האתגר התחלק לשני חלקים, מחקר ופיתוח, במאמר הזה נציג את הפתרונות לאתגרים של המחקר.

שלב מקדים - למצוא את הדרך לאתגר

בדומה לאתגרי המוסד, השלב הראשוני פורסם בעיתון וברשתות החברתיות והיינו צריכים להבין איך להגיע אליו. באתר של השב"כ פורסמה התמונה הבאה:

כפי שניתן לראות, בתמונה יש מחרוזת בקידוד של Base64 אשר מתורגמת בסופו של דבר לכתובת האתר שבו מאוכסן האתגר.

פיענוח המחרוזת ב-Linux:

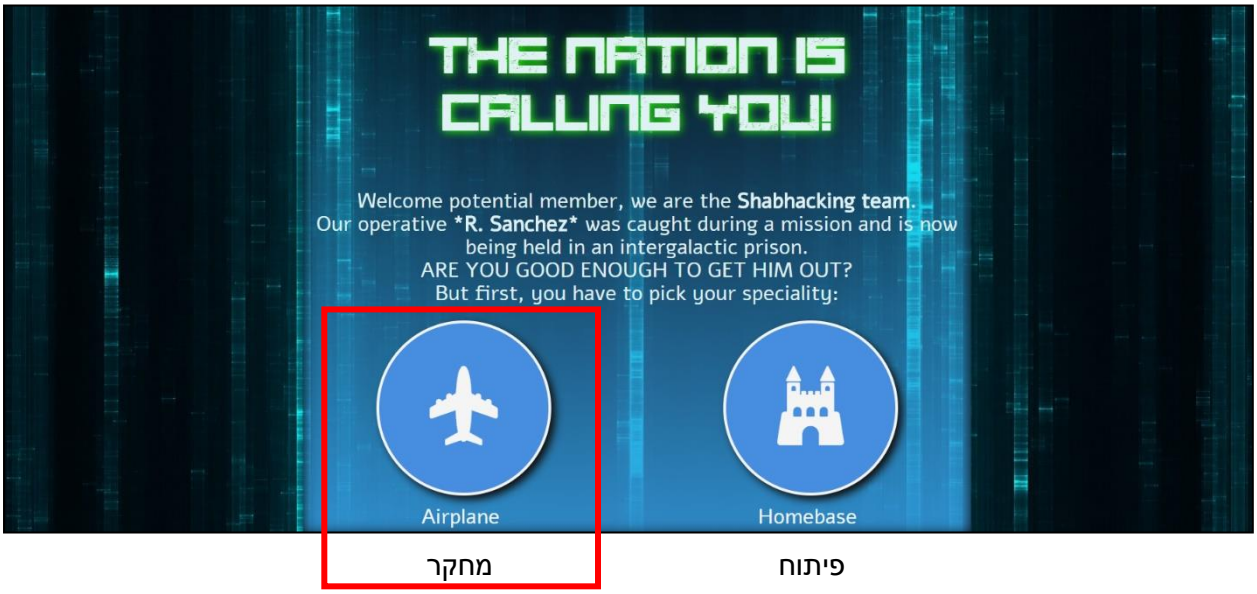
```
base64 -d <<< MTAxMDAxMTAxMTAxMDAwMDExMDAwMDEwMTEwMDAxMDAxMTAwMDAxMDExMDEwMTEuY29t
```

התוצאה:

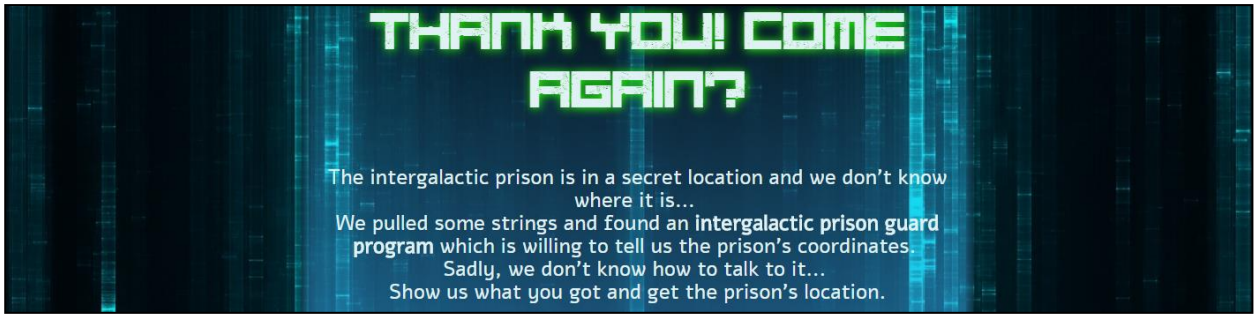
https://101001101101000011000010110001001100001011010 11.com

(שהיא גם הערך הבינארי של המחרוזת "Shabak").

באתר הזה מופיעים האתגרים של השב"כ בשני חלקים כמו שרואים בתמונה:



שלב ראשון - Roman Emperor



בהתחלה של הקובץ כשמגיעים לפונקציה main יש בדיקה האם יש תיקיה בשם **"meseeker inc"**. 0x10 זה הקבוע שבודק האם מדובר בתיקיה.

```
.text:00E212F1                 lea     eax, [esp+4F8h+filePath]
.text:00E212F5                 push    104h            ; nSize
.text:00E212FA                 push    eax             ; lpDst
.text:00E212FB                 push    offset Src      ; "%PROGRAMFILES%\\meseeker inc"
.text:00E21300                 call    ds:ExpandEnvironmentStringsA
.text:00E21306
.text:00E21306
.text:00E21306                 lea     eax, [esp+4F8h+FileInformation]
.text:00E2130A                 push    eax             ; lpFileInformation
.text:00E2130B                 push    0               ; fInfoLevelId
.text:00E2130D                 lea     eax, [esp+500h+filePath]
.text:00E21311                 push    eax             ; lpFileName
.text:00E21312                 call    ds:GetFileAttributesExA
.text:00E21318                 test    eax, eax
.text:00E2131A                 jz      short loc_E21381
.text:00E2131C
.text:00E2131C
.text:00E2131C                 test    byte ptr [esp+4F8h+FileInformation.dwFileAttributes], 10h
.text:00E21321                 jz      short loc_E21381
```

לאחר הבדיקה הזו בודקים את הזמן שפעם אחרונה ניגשו לקובץ.

```
.text:00E21323
.text:00E21323                 mov     ecx, [esp+4F8h+FileInformation.ftLastAccessTime.dwHighDateTime]
.text:00E21327                 mov     esi, ds:highDateTime
.text:00E2132D                 mov     eax, [esp+4F8h+FileInformation.ftLastAccessTime.dwLowDateTime]
.text:00E21331                 mov     edx, ds:lowDateTime
```

בודקים את ההפרש בין ה-low time ש-hardcoded בקובץ לבין ה-low time של הפעם אחרונה שניגשו לקובץ וגם את ההפרש של ה-high time ש-hardcoded לבין ה-high time של הפעם האחרונה שניגשו לקובץ.

```
.text:00E21347 loc_E21347:                                     ;
.text:00E21347                                                 ;
.text:00E21347                 sub     edx, eax
.text:00E21349                 mov     eax, edx
.text:00E2134B                 sbb     esi, ecx
.text:00E2134D                 mov     ecx, esi
.text:00E2134F
.text:00E2134F loc_E2134F:                                     ;
.text:00E2134F                 cmp     ecx, 201
.text:00E21355                 jb      short loc_E21360
.text:00E21357
.text:00E21357
.text:00E21357                 ja      short loc_E21381
.text:00E21359
.text:00E21359
.text:00E21359                 cmp     eax, 2A69C000h
.text:00E2135E                 jnb     short loc_E21381
```

על מנת למצוא את הסיסמא לא צריך לשנות את השדה שפעם אחרונה ניגשו לקובץ זה סתם מיותר. בתרגיל הזה מספיק לשנות את הקטע בקוד ולשים patch בקוד על מנת לגרום לתנאי הזה לעבור נשנה את הדגלים ב-runtime ב-x32dbg כדי שהתנאי יעבור.

```
    00E2134F   81 F9 C9 00 00 00   cmp ecx,C9
    00E21355   72 09               jb first.E21360
    00E21357   77 28               ja first.E21381
    00E21359   3D 00 C0 69 2A      cmp eax,2A69C000
    00E2135E   73 21               jae first.E21381
EIP 00E21360   8D 8C 24 30 01 00 00 lea ecx,dword ptr ss:[esp+130]
    00E21367   E8 24 00 00 00      call first.E21390
```

לאחר מכן מגיעים לקטע קוד שמפענח את המחרוזת שתביא את הסיסמא בתוך הפונקציה: d4d_decodePassOutput

```
.text:00E21360 loc_E21360:                             ; CODE XREF: _main+85Tj
.text:00E21360                 lea     ecx, [esp+4F8h+passOutput]
.text:00E21367                 call    d4d_decodePassOutput
.text:00E2136C
.text:00E2136C
.text:00E2136C                 lea     eax, [esp+4F8h+passOutput]
.text:00E21373                 push    eax
.text:00E21374                 push    offset aS       ; "%s"
.text:00E21379                 call    d4d_printf
.text:00E2137E                 add     esp, 8
```

אם נכנסים לתוך הפונקציה הזו מגיעים לפעולות של xmm שיבצעו פיענוח של המחרוזת עם פעולות XOR שיעבדו על 16 בתים במקביל שתדפיס את הסיסמא למסך.

Xmm1 מכיל את המפתח ל-XOR שהוא hardcoded בקובץ:

```
.text:00E213C0                                  decryptBuf:                             ; CODE XREF: d4d_decodePassOutput+55↓j
.text:00E213C0 0F 10 04 96                                      movups  xmm0, xmmword ptr [esi+edx*4]
.text:00E213C4 83 C2 08                                         add     edx, 8
.text:00E213C7 8D 40 20                                         lea     eax, [eax+20h]
.text:00E213CA 66 0F EF C1                                      pxor    xmm0, xmm1
.text:00E213CE 0F 11 40 D0                                      movups  xmmword ptr [eax-30h], xmm0
.text:00E213D2 0F 10 44 01 E0                                   movups  xmm0, xmmword ptr [ecx+eax-20h]
.text:00E213D7 66 0F EF C1                                      pxor    xmm0, xmm1
.text:00E213DB 0F 11 40 E0                                      movups  xmmword ptr [eax-20h], xmm0
.text:00E213DF 81 FA F0 00 00 00                                cmp     edx, 0F0h
.text:00E213E5 72 D9                                            jb      short decryptBuf
.text:00E213E7
.text:00E213E7
.text:00E213E7 81 FA F1 00 00 00                                cmp     edx, 0F1h
.text:00E213ED 73 25                                            jnb     short loc_E21414
```

ובסוף מגיעים לתוצאה הזו:

```
Great Job!
Your password is:

Never_Clean_DNA_Vials_With_Your_Spit
```

שלב שני - The Song

נתחיל לבדוק מה עושה הפונקציה main ש-IDA PRO מצא אוטומטית:

```
.text:009110B7 6A 32                                            push    32h             ; size_t
.text:009110B9 8D 45 C0                                         lea     eax, [ebp+myPass]
.text:009110BC 6A 00                                            push    0               ; int
.text:009110BE 50                                               push    eax             ; void *
.text:009110BF E8 CC 35 01 00                                   call    _memset
.text:009110C4
.text:009110C4
.text:009110C4 6A 00                                            push    0
.text:009110C6 E8 24 3E 01 00                                   call    ___acrt_iob_func
.text:009110CB
.text:009110CB
.text:009110CB 50                                               push    eax             ; FILE *
.text:009110CC 8D 45 C0                                         lea     eax, [ebp+myPass]
.text:009110CF 6A 32                                            push    32h             ; int
.text:009110D1 50                                               push    eax             ; char *
.text:009110D2 E8 FD 52 01 00                                   call    _fgets
.text:009110D7
.text:009110D7
.text:009110D7 8D 4D C0                                         lea     ecx, [ebp+myPass]
.text:009110DA E8 71 00 00 00                                   call    j_d4d_calcHash
.text:009110DF
```

בסוף רואים את הפונקציה fgets שאיתה מכניסים קלט למשתמש שאמורה להיות סיסמא כלשהי. אם ממשיכים לרוץ פתאום התוכנית נסגרת, במידה וזה קורה צריך להסתכל מה גורם לתוכנה להיסגר. התוכנה נסגרת בגלל שיש כנראה Anti Debug שפספסנו בהתחלה, אז צריך לחזור לפונקציות שנקראו לפני ה-main ולבדוק מה קרה.

אם חוזרים אחורה בקוד רואים שיש פונקציה אשר אחראית לאתחול של קוד בשם init_term:

```
.text:00923ABE
.text:00923ABE                 loc_923ABE:                          ; CODE XREF: __scrt_common_main_seh(void)+59↑j
.text:00923ABE 68 34 41 93 00                 push    offset unk_934134
.text:00923AC3 68 24 41 93 00                 push    offset funcTable
.text:00923AC8 E8 1A 31 00 00                 call    __initterm
.text:00923ACD 59                             pop     ecx
.text:00923ACE 59                             pop     ecx
.text:00923ACF C7 05 B0 B8 93 00 02 00+       mov     dword_93B8B0, 2
.text:00923AD9 EB 05                          jmp     short loc_923AE0
```

הפונקציה הזו מקבלת שני פרמטרים: הפרמטר הראשון זה איפה שמתחילה הטבלה של הפונקציות, הפרמטר השני זה איפה שנגמרת הטבלה. אם ממשיכים להסתכל על הטבלה עצמה, רואים שם רשימה של פונקציות בטבלה, בואו נביט על הפונקציות האלה:

```
.rdata:00934124 00                 funcTable       db    0
.rdata:00934125 00                                 db    0
.rdata:00934126 00                                 db    0
.rdata:00934127 00                                 db    0
.rdata:00934128 40 3A 92 00                        dd offset sub_923A40
.rdata:0093412C F0 11 91 00                        dd offset d4d_overwriteFunc
.rdata:00934130 D0 38 92 00                        dd offset d4d_antidebugThread
```

מה שמעניין בטבלה הזו זה הפונקציה האחרונה שעושה אנטי דיבאג ועוד פונקציה שמשכתבת קטע קוד. נתחיל להסתכל על הפונקציה d4d_overwriteFunc

```
.text:009111F0                 ; =============== S U B R O U T I N E =======================================
.text:009111F0
.text:009111F0
.text:009111F0                 d4d_overwriteFunc proc near          ; DATA XREF: .rdata:0093412C↓o
.text:009111F0 B8 28 38 92 00                 mov     eax, offset d4d_decryptDll
.text:009111F5 B9 60 11 91 00                 mov     ecx, offset sub_911160
.text:009111FA 0F 10 00                       movups  xmm0, xmmword ptr [eax]
.text:009111FD 0F 11 01                       movups  xmmword ptr [ecx], xmm0
.text:00911200 0F 10 40 10                    movups  xmm0, xmmword ptr [eax+10h]
.text:00911204 0F 11 41 10                    movups  xmmword ptr [ecx+10h], xmm0
.text:00911208 0F 10 40 20                    movups  xmm0, xmmword ptr [eax+20h]
.text:0091120C 0F 11 41 20                    movups  xmmword ptr [ecx+20h], xmm0
.text:00911210 66 8B 40 30                    mov     ax, [eax+30h]
.text:00911214 66 89 41 30                    mov     [ecx+30h], ax
.text:00911218 C3                             retn
.text:00911218                 d4d_overwriteFunc endp
.text:00911218
```

הפונקציה הזו משנה פונקציה בקוד של ה-main וכותבת קוד אחר שיתבצע בפונקציה הזו. עכשיו נסתכל על מה הפונקציה d4d_antidebugThread עושה:

```
.text:009238D0                 d4d_antidebugThread proc near        ; DATA XREF: .rdata:00934130↓o
.text:009238D0 6A 00                          push    0               ; lpThreadId
.text:009238D2 6A 00                          push    0               ; dwCreationFlags
.text:009238D4 6A 00                          push    0               ; lpParameter
.text:009238D6 68 60 38 92 00                 push    offset d4d_antidebug ; lpStartAddress
.text:009238DB 6A 00                          push    0               ; dwStackSize
.text:009238DD 6A 00                          push    0               ; lpThreadAttributes
.text:009238DF FF 15 14 40 93 00              call    ds:CreateThread
.text:009238E5 C3                             retn
.text:009238E5                 d4d_antidebugThread endp
```

הפונקציה הזו קוראת ל-thread שיבדוק האם יש דיבאגר, במידה ונמצא דיבאגר התוכנית תיסגר באמצע, וזו הסיבה שהתוכנית שלנו נסגרה.

הפונקציה עצמה נראית כך:

```
.text:00923860                          ; DWORD __stdcall d4d_antidebug(LPVOID lpThreadParameter)
.text:00923860                          d4d_antidebug   proc near               ; DATA XREF: d4d_antidebugThread+6↓o
.text:00923860
.text:00923860                          ThreadId        = dword ptr -8
.text:00923860                          var_4           = dword ptr -4
.text:00923860                          lpThreadParameter= dword ptr  8
.text:00923860
.text:00923860 55                                       push    ebp
.text:00923861 8B EC                                    mov     ebp, esp
.text:00923863 83 E4 F8                                 and     esp, 0FFFFFFF8h
.text:00923866 83 EC 0C                                 sub     esp, 0Ch
.text:00923869
.text:00923869
.text:00923869 53                                       push    ebx
.text:0092386A 56                                       push    esi
.text:0092386B 8B 35 04 40 93 00                        mov     esi, ds:GetTickCount64
.text:00923871 57                                       push    edi
.text:00923872 FF D6                                    call    esi ; GetTickCount64
.text:00923874
.text:00923874                          loc_923874:                             ; CODE XREF: d4d_antidebug+6E↓j
.text:00923874 8B DA                                    mov     ebx, edx
.text:00923876 8B F8                                    mov     edi, eax
.text:00923878 FF D6                                    call    esi ; GetTickCount64
.text:0092387A 8B F0                                    mov     esi, eax
.text:0092387C 89 44 24 10                              mov     [esp+18h+ThreadId], eax
.text:00923880 2B F7                                    sub     esi, edi
.text:00923882 89 54 24 14                              mov     [esp+18h+var_4], edx
.text:00923886 8B CA                                    mov     ecx, edx
.text:00923888 1B CB                                    sbb     ecx, ebx
.text:0092388A 85 C9                                    test    ecx, ecx
.text:0092388C 72 3A                                    jb      short loc_9238C8
.text:0092388E
.text:0092388E
.text:0092388E 77 05                                    ja      short loc_923895
.text:00923890
.text:00923890
.text:00923890 83 FE 64                                 cmp     esi, 64h
.text:00923893 76 33                                    jbe     short loc_9238C8
.text:00923895
.text:00923895                          loc_923895:                             ; CODE XREF: d4d_antidebug+2E↑j
.text:00923895 68 6C 94 93 00                           push    offset aDebuggingAttem ; "Debugging attempt found... aborting!"
.text:0092389A 68 68 94 93 00                           push    offset unk_939468
.text:0092389F E8 BC 00 00 00                           call    d4d_printf
```

כפי שניתן לראות בתמונה ה-thread בודק עם הפונקציה GetTickCount64 כמה זמן עבר בין קטעי הקוד. אם עברו מעל 100 מילי שניות הפונקציה תפתח עוד thread שמטרתו תהיה לסגור את התוכנית עם הפונקציה ExitProcess.

בכדי לעקוף את האנטי דיבאג הזה אפשר לעשות שני דברים:

1. לגרום ל-thread להיות במצב suspend (מספר 4 ב-DwCreationFlags)

```
HANDLE WINAPI CreateThread(
  _In_opt_   LPSECURITY_ATTRIBUTES  lpThreadAttributes,
  _In_       SIZE_T                 dwStackSize,
  _In_       LPTHREAD_START_ROUTINE lpStartAddress,
  _In_opt_   LPVOID                 lpParameter,
  _In_       DWORD                  dwCreationFlags,
  _Out_opt_  LPDWORD                lpThreadId
);
```

2. לעשות Patch ולכתוב בשורה הראשונה של ה-thread את הפקודה ret או לעשות JMP שירוץ על עצמו.

הבחירה שהתקבלה היה לשים ב-thread את הפקודה ret:

```
00923860   C3                  ret
00923861   8B EC               mov ebp,esp
00923863   83 E4 F8            and esp,FFFFFFF8
00923866   83 EC 0C            sub esp,C
00923869   53                  push ebx
0092386A   56                  push esi
0092386B   8B 35 04 40 93 00   mov esi,dword ptr ds:[<&GetTickCount64>]
00923871   57                  push edi
00923872   FF D6               call esi
```

לאחר שסיימנו להתמודד עם ה-Anti Debugging נחזור לפונקציה main ממקודם. לאחר שנכניס סיסמא נגיע לקטע קוד שמחשב Hash על הסיסמא:

```
.text:009110D7
.text:009110D7
.text:009110D7 8D 4D C0                    lea     ecx, [ebp+myPass]
.text:009110DA E8 71 00 00 00              call    j_d4d_calcHash
```

פונקציה זו מחשבת Hash מסוג FNV, על ידי חיפוש של הקבוע 0x811c9dc5 בגוגל הגענו למסקנה שמדובר ב-FNV:

```
.text:00923900
.text:00923900                 d4d_calcHash    proc near               ; CODE XREF: j_d4d_calcHash↑j
.text:00923900 56                              push    esi
.text:00923901 BA C5 9D 1C 81                  mov     edx, 811C9DC5h
.text:00923906 41                              inc     ecx
.text:00923907 BE 0A 00 00 00                  mov     esi, 0Ah
.text:0092390C 0F 1F 40 00                     nop     dword ptr [eax+00h]
.text:00923910
.text:00923910                 loc_923910:                             ; CODE XREF: d4d_calcHash+52↓j
.text:00923910 0F B6 41 FF                     movzx   eax, byte ptr [ecx-1]
.text:00923914 8D 49 05                        lea     ecx, [ecx+5]
.text:00923917 33 C2                           xor     eax, edx
.text:00923919 69 D0 93 01 00 01               imul    edx, eax, 1000193h
.text:0092391F 0F B6 41 FB                     movzx   eax, byte ptr [ecx-5]
.text:00923923 33 D0                           xor     edx, eax
.text:00923925 0F B6 41 FC                     movzx   eax, byte ptr [ecx-4]
.text:00923929 69 D2 93 01 00 01               imul    edx, 1000193h
.text:0092392F 33 D0                           xor     edx, eax
.text:00923931 0F B6 41 FD                     movzx   eax, byte ptr [ecx-3]
.text:00923935 69 D2 93 01 00 01               imul    edx, 1000193h
.text:0092393B 33 D0                           xor     edx, eax
.text:0092393D 0F B6 41 FE                     movzx   eax, byte ptr [ecx-2]
.text:00923941 69 D2 93 01 00 01               imul    edx, 1000193h
.text:00923947 33 D0                           xor     edx, eax
.text:00923949 69 D2 93 01 00 01               imul    edx, 1000193h
.text:0092394F 83 EE 01                        sub     esi, 1
.text:00923952 75 BC                           jnz     short loc_923910
.text:00923954 8B C2                           mov     eax, edx
.text:00923956 5E                              pop     esi
.text:00923957 C3                              retn
.text:00923957                 d4d_calcHash    endp
```

לאחר חישוב ההאש יש הקצאת זיכרון וקריאה לפונקציה d4d_decryptDll

```
.text:009110DF
.text:009110DF
.text:009110DF 68 98 94 93 00                  push    offset aDecryptingPass ; "Decrypting password.\n"
.text:009110E4 68 68 94 93 00                  push    offset aS       ; "%s"
.text:009110E9 89 45 F8                        mov     [ebp+hashOfPass], eax
.text:009110EC E8 6F 28 01 00                  call    d4d_printf
.text:009110F1
.text:009110F1
.text:009110F1 FF 35 20 38 92 00               push    ds:nNumberOfBytesToWrite ; size_t
.text:009110F7 E8 E3 52 01 00                  call    _malloc
.text:009110FC 83 C4 30                        add     esp, 30h
.text:009110FF 89 45 F4                        mov     [ebp+var_C], eax
.text:00911102 85 C0                           test    eax, eax
.text:00911104 75 0B                           jnz     short loc_911111
.text:00911106
.text:00911106
.text:00911106 B8 20 00 00 00                  mov     eax, 20h
.text:0091110B 5F                              pop     edi
.text:0091110C 5E                              pop     esi
.text:0091110D 8B E5                           mov     esp, ebp
.text:0091110F 5D                              pop     ebp
.text:00911110 C3                              retn
.text:00911111                 ; ---------------------------------------------------------------------------
.text:00911111
.text:00911111                 loc_911111:                             ; CODE XREF: _main+64↑j
.text:00911111 C7 45 FC 00 00 00 00            mov     [ebp+var_4], 0
.text:00911118 8B 45 F8                        mov     eax, [ebp+hashOfPass]
.text:0091111B 8B 75 F4                        mov     esi, [ebp+var_C]
.text:0091111E 68 00 10 91 00                  push    offset d4d_loadDll
.text:00911123 8D 3D 20 12 91 00               lea     edi, encodedDll
.text:00911129 8B 0D 20 38 92 00               mov     ecx, ds:nNumberOfBytesToWrite
.text:0091112F E8 2C 00 00 00                  call    sub_911160      ; d4d_decryptDll
```

הסיבה שקראנו לפונקציה d4d_decryptDll זה בגלל שהפונקציה d4d_loadDll טוענת DLL לזיכרון ולוקחת ממנו איזה פונקציה שנזכיר עוד מעט. כפי שרואים בתמונה למטה מתבצע חישוב שמפענח את ה-encodedDLL:

```
.text:00923828
.text:00923828                          d4d_decryptDll:                         ; DATA XREF: d4d_overwriteFunc↑o
.text:00923828 85 F6                                    test    esi, esi
.text:0092382A 75 02                                    jnz     short loc_92382E
.text:0092382C 14 40                                    adc     al, 40h
.text:0092382E
.text:0092382E                          loc_92382E:                             ; CODE XREF: .text:0092382A↑j
.text:0092382E 33 07                                    xor     eax, [edi]
.text:00923830 89 06                                    mov     [esi], eax
.text:00923832 89 C2                                    mov     edx, eax
.text:00923834 B8 04 00 00 00                           mov     eax, 4
.text:00923839 75 02                                    jnz     short loc_92383D
.text:00923839                          ; ---------------------------------------------------------------------------
.text:0092383B 81                                       db 81h
.text:0092383C C2                                       db 0C2h ; _
.text:0092383D                          ; ---------------------------------------------------------------------------
.text:0092383D
.text:0092383D                          loc_92383D:                             ; CODE XREF: .text:00923839↑j
.text:0092383D                                                                  ; .text:00923851↓j
.text:0092383D 39 C8                                    cmp     eax, ecx
.text:0092383F 75 07                                    jnz     short loc_923848
.text:00923841 87 74 24 04                              xchg    esi, [esp+4]
.text:00923845 FF E6                                    jmp     esi             ; d4d_loadDll
.text:00923845                          ; ---------------------------------------------------------------------------
.text:00923847 B8                                       db 0B8h
.text:00923848                          ; ---------------------------------------------------------------------------
.text:00923848
.text:00923848                          loc_923848:                             ; CODE XREF: .text:0092383F↑j
.text:00923848 33 14 07                                 xor     edx, [edi+eax]
.text:0092384B 89 14 06                                 mov     [esi+eax], edx
.text:0092384E 83 C0 04                                 add     eax, 4
.text:00923851 EB EA                                    jmp     short loc_92383D
```

בפונקציה d4d_loadDll מנסים לטעון את הקובץ DLL בעזרת הפונקציה LoadLibraryA:

```
.text:0091105A
.text:0091105A                          loc_91105A:                             ; CODE XREF: d4d_loadDll+4D↑j
.text:0091105A 56                                       push    esi             ; hObject
.text:0091105B FF 15 10 40 93 00                        call    ds:CloseHandle
.text:00911061
.text:00911061
.text:00911061 68 F4 94 93 00                           push    offset LibFileName ; "GettingSchwifty.bat"
.text:00911066 FF 15 0C 40 93 00                        call    ds:LoadLibraryA
.text:0091106C 5E                                       pop     esi
.text:0091106D
.text:0091106D
.text:0091106D 85 C0                                    test    eax, eax
.text:0091106F 75 0B                                    jnz     short loc_91107C
.text:00911071
.text:00911071
.text:00911071 68 86 00 00 00                           push    86h             ; uExitCode
.text:00911076 FF 15 1C 40 93 00                        call    ds:ExitProcess
```

מכיוון שהסיסמא שהכנסנו לא הייתה נכונה, הקובץ DLL לא ייטען כי לא קיבלנו קובץ DLL. ה-Hash של הסיסמא שהכנסנו הוא המפתח כדי לפענח את הקובץ DLL. יש דרך לדעת מה היה ה-Hash.

אנחנו יודעים איך קובץ DLL מתחיל, הרי יש לנו גם את הבאפר המקודד, אז ניקח את ה-4 בתים מהבאפר המקודד ונבצע XOR עם ה-4 בתים הידועים שאיתם מתחיל ה-DLL וככה נגיע אל ההאש הנכון שצריכה להיות הסיסמא. ה-4 בתים ראשונים של הבאפר הם: 0xE80285B5. ה-4 בתים ראשונים של קובץ ה-DLL הם: 0x00905A4D. על ידי ביצוע:

0xE80285B5 ^ 0x00905A4D = 0xE892DFF8

נקבל את ה-Hash הנכון של הסיסמא.

על מנת לקבל את הקובץ DLL אנחנו יכולים לנסות לבצע ברוט פורס ולנסות להגיע להאש הזה או לעשות שינוי ב-runtime עם הדיבאגר כדי לקבל את ההאש הזה.

הדרך שנבחרה הייתה לשנות את ההאש בזמן ריצה ל-0xE892DFF8. לאחר שההאש הנכון התקבל, קיבלנו קובץ DLL כפי שניתן לראות בתמונה למטה:

```
00338970  4D 5A 90 00  03 00 00 00  04 00 00 00  FF FF 00 00  MZ.........ÿÿ..
00338980  B8 00 00 00  00 00 00 00  40 00 00 00  00 00 00 00  ........@.......
00338990  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
003389A0  00 00 00 00  00 00 00 00  00 00 00 00  08 01 00 00  ................
003389B0  0E 1F BA 0E  00 B4 09 CD  21 B8 01 4C  CD 21 54 68  ..º..´.Í!¸.LÍ!Th
003389C0  69 73 20 70  72 6F 67 72  61 6D 20 63  61 6E 6E 6F  is program canno
003389D0  74 20 62 65  20 72 75 6E  20 69 6E 20  44 4F 53 20  t be run in DOS 
003389E0  6D 6F 64 65  2E 0D 0D 0A  24 00 00 00  00 00 00 00  mode....$.......
003389F0  CC 21 09 84  88 40 67 D7  88 40 67 D7  88 40 67 D7  Ì!...@gx.@gx.@gx
00338A00  3C DC 96 D7  81 40 67 D7  3C DC 94 D7  FE 40 67 D7  <Ü.x.@gx<Ü.xþ@gx
00338A10  3C DC 95 D7  90 40 67 D7  B3 1E 64 D6  99 40 67 D7  <Ü.x.@gx³.dÖ.@gx
00338A20  B3 1E 62 D6  9E 40 67 D7  B3 1E 63 D6  87 40 67 D7  ³.bÖ.@gx³.cÖ.@gx
00338A30  55 BF AC D7  8B 40 67 D7  88 40 66 D7  DE 40 67 D7  U¿¬x.@gx.@fxÞ@gx
00338A40  1F 1E 6E D6  8A 40 67 D7  88 40 67 D7  89 40 67 D7  ..nÖ.@gx.@gx.@gx
00338A50  1F 1E 67 D6  89 40 67 D7  1A 1E 98 D7  89 40 67 D7  ..gÖ.@gx...x.@gx
00338A60  1F 1E 65 D6  89 40 67 D7  52 69 63 68  88 40 67 D7  ..eÖ.@gxRich.@gx
00338A70  00 00 00 00  00 00 00 00  50 45 00 00  4C 01 06 00  ........PE..L...
00338A80  EA BB 00 59  00 00 00 00  00 00 00 00  E0 00 02 21  ê».Y........à..!
00338A90  0B 01 0E 00  00 AC 00 00  00 82 00 00  00 00 00 00  .....¬..........
00338AA0  9D 17 00 00  00 10 00 00  00 C0 00 00  00 00 00 10  .........À......
00338AB0  00 10 00 00  00 02 00 00  06 00 00 00  00 00 00 00
```

אחרי שטוענים את הקובץ DLL, מגיעים לפונקציה exported בשם Piper_16

```
.text:737913C0                 public Piper_16
.text:737913C0 Piper_16        proc near               ; DATA XREF: .rdata:off_737A1178↓o
.text:737913C0
.text:737913C0 NumberOfBytesWritten= dword ptr -18h
.text:737913C0 var_14          = dword ptr -14h
.text:737913C0 NumberOfBytesRead= dword ptr -10h
.text:737913C0 Buffer          = byte ptr -0Ch
.text:737913C0 var_4           = byte ptr -4
.text:737913C0
.text:737913C0                 push    ebp
.text:737913C1                 mov     ebp, esp
.text:737913C3                 and     esp, 0FFFFFFF8h
.text:737913C6                 sub     esp, 1Ch
.text:737913C9                 push    esi
.text:737913CA                 push    0               ; hTemplateFile
.text:737913CC                 push    0               ; dwFlagsAndAttributes
.text:737913CE                 push    3               ; dwCreationDisposition
.text:737913D0                 push    0               ; lpSecurityAttributes
.text:737913D2                 push    3               ; dwShareMode
.text:737913D4                 push    GENERIC_WRITE or GENERIC_READ ; dwDesiredAccess
.text:737913D9                 push    offset FileName ; "\\\\.\\pipe\\flumbus_channel"
.text:737913DE                 call    ds:CreateFileA
```

הפונקציה הזו מנסה לפתוח PIPE אך אין באמת PIPE כזה, באתגר הזה מצפים שנחשוב מחוץ לקופסא כדי להגיע אל הפתרון, לאחר שממשיכים לעבור על הקוד מגיעים לקטע הבא:

```
.text:737913E4                 push    0               ; lpOverlapped
.text:737913E6                 mov     esi, eax
.text:737913E8                 mov     [esp+24h+NumberOfBytesWritten], 0
.text:737913F0                 lea     eax, [esp+24h+NumberOfBytesWritten]
.text:737913F4                 push    eax             ; lpNumberOfBytesWritten
.text:737913F5                 push    20h             ; nNumberOfBytesToWrite
.text:737913F7                 push    offset aWhatIsCoolerTh ; "What is cooler than being cool?"
.text:737913FC                 push    esi             ; hFile
.text:737913FD                 call    ds:WriteFile
.text:73791403
.text:73791403
.text:73791403                 xor     eax, eax
.text:73791405                 mov     [esp+20h+NumberOfBytesRead], 0
.text:7379140D                 push    eax             ; lpOverlapped
.text:7379140E                 mov     dword ptr [esp+24h+Buffer+1], eax
.text:73791412                 mov     word ptr [esp+24h+Buffer+5], ax
.text:73791417                 mov     [esp+24h+Buffer+7], al
.text:7379141B                 lea     eax, [esp+24h+NumberOfBytesRead]
.text:7379141F                 push    eax             ; lpNumberOfBytesRead
.text:73791420                 push    8               ; nNumberOfBytesToRead
.text:73791422                 lea     eax, [esp+2Ch+Buffer]
.text:73791426                 mov     [esp+2Ch+Buffer], 0
.text:7379142B                 push    eax             ; lpBuffer
.text:7379142C                 push    esi             ; hFile
.text:7379142D                 call    ds:ReadFile
```

אנחנו רואים שיש פה שאלה ואחרי זה מנסים לקרוא מה-pipe אך בגלל שאין באמת pipe ייקראו 0 בתים.

לפי הפונקציה ReadFile הוא מנסה לקרוא 8 בתים.

לאחר מכן הוא ממיר את האותיות הקטנות לאותיות גדולות:

```
.text:73791433                 lea     ecx, [esp+20h+Buffer]
.text:73791437
.text:73791437 loc_73791437:                           ; CODE XREF: Piper_16+8C↓j
.text:73791437                 mov     al, [ecx]
.text:73791439                 cmp     al, 61h
.text:7379143B                 jb      short loc_73791445
.text:7379143D
.text:7379143D
.text:7379143D                 cmp     al, 7Ah
.text:7379143F                 ja      short loc_73791445
.text:73791441
.text:73791441
.text:73791441                 and     al, 5Fh
.text:73791443                 mov     [ecx], al
.text:73791445
.text:73791445 loc_73791445:                           ; CODE XREF: Piper_16+7B↑j
.text:73791445                                         ; Piper_16+7F↑j
.text:73791445                 inc     ecx
.text:73791446                 lea     eax, [esp+20h+var_4]
.text:7379144A                 cmp     ecx, eax
.text:7379144C                 jnz     short loc_73791437
```

בסוף מנסים לפענח את הבלוק בזיכרון ומבצעים האש על הבלוק שפוענח, אם ההאש יהיה נכון נקבל את הסיסמא.

```
.text:73791451                 mov     [esp+28h+var_14], 8
.text:73791459                 lea     eax, [esp+28h+Buffer]
.text:7379145D                 lea     ecx, [esp+28h+NumberOfBytesWritten]
.text:73791461                 mov     [esp+28h+NumberOfBytesWritten], eax
.text:73791465                 call    d4d_decryptPassBuf
.text:7379146A
.text:7379146A
.text:7379146A                 call    d4d_calcHash
.text:7379146F                 cmp     eax, 55B8B000h
.text:73791474                 jz      short loc_7379148B
.text:73791476
.text:73791476
.text:73791476                 push    offset aOhManYouNeedTo ; "Oh Man, you need to work on work music "...
.text:7379147B                 call    _puts
.text:73791480                 add     esp, 4
.text:73791483                 push    0               ; uExitCode
.text:73791485                 call    ds:ExitProcess
.text:7379148B ; ---------------------------------------------------------------------------
.text:7379148B
.text:7379148B loc_7379148B:                           ; CODE XREF: Piper_16+B4↑j
.text:7379148B                 mov     eax, offset PassBuf
.text:73791490                 pop     esi
.text:73791491                 mov     esp, ebp
.text:73791493                 pop     ebp
.text:73791494                 retn
.text:73791494 Piper_16        endp
```

הבאפר שמכיל את הסיסמא שאנחנו צריכים מוצפן עם משהו שדומה ל-RC4, אבל זה לא RC4...

להלן המימוש של ההצפנה:

```
65  tmp = 0
66  pos = 0
67  newbuf = ""
68  box = []
69  for i in range(0x100):
70      box.append(i)
71
72  for i in range(0x100):
73      pos = (pos + ord(key[i%8]) + box[i]) &0xff
74      tmp = box[i]
75      box[i] = box[pos]
76      box[pos] = tmp
77
78  pos = 0
79  k = 0
80  for i in range(len(buf)):
81      tmp = (pos + 1) & 0xff
82      pos = (box[tmp] + k) & 0xff
83      k = tmp
84      pos1 = (box[pos] + box[tmp]) & 0xff
85      buf[i] ^=box[pos1]
86
87  print buf
```

בכדי לדעת מה הסיסמא יש שתי דרכים: או לבצע ברוט פורס בתקווה שנקלע לסיסמא הנכונה או לנסות להסתכל על כל המחרוזות שקיבלנו בקוד ולהבין מה הכוונה. אחרי חיפוש קצר בגוגל של המחרוזת:

"What is cooler than being cool?"

הסתבר שמדובר בשיר ישן שלא הכרתי והתשובה שם לפי השיר הייתה Ice Cold... בכדי לקבל את המפתח Ice cold היה צריך לבצע Patch בקוד למשתנה Buffer, אותו משתנה שמנסים לקרוא לתוכו 8 תווים.

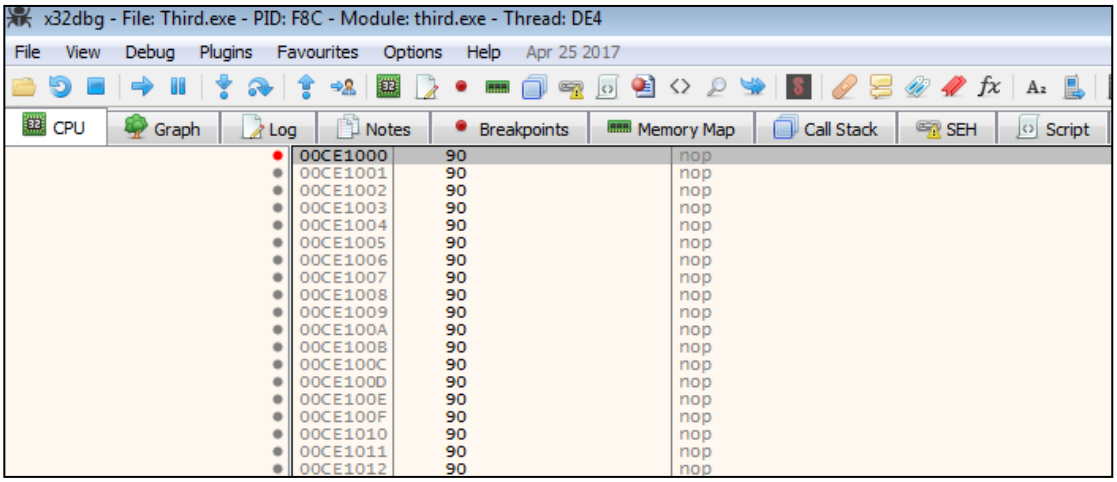
```
Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct
Address   Hex                                          ASCII
0018FAAC  49 63 65 20 43 6F 6C 64 85 10 91 00 C4 FA 18 00  Ice Cold....Äú..
```

לאחר מכן פוענח ה-output שמכיל את הסיסמא.

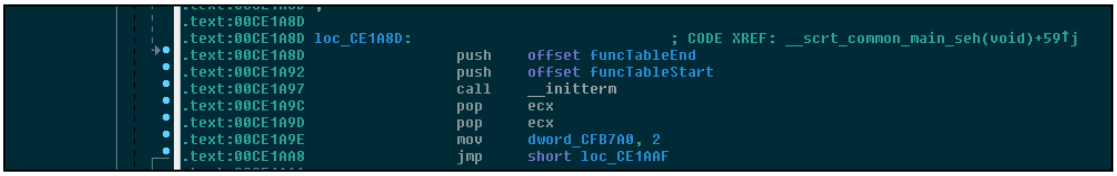
```
You are entering highly secure area, please enter your password.
1234
Decrypting password.
   *-----------------------------------------------------------*
  /                                                             \
  |                                                             |
  |                                                             |
  |    Great Job!                                               |
  |    Your password is:                                        |
  |                                                             |
  |    You_Pass_The_Butter                                      |
  |                                                             |
  |                                                             |
  |                                                             |
  |                                                             |
  \                                                             /
   *-----------------------------------------------------------*
```

שלב אחרון - The Hidden DLL

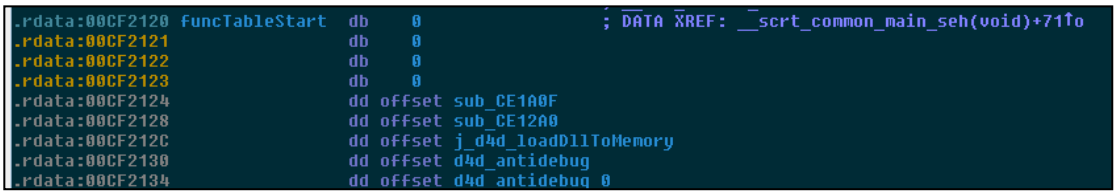
בשלב השלישי, אם ננסה להגיע לפונקציה main כפי שמופיע ב-IDA נקבל את המצב הבא:



זה אומר שגם בשלב הזה כמו בשלב השני, נעשה שימוש ב-init_term בכדי להפריע לנתח את הקוד.



בטבלה הבאה יש שלוש פונקציות מעניינות:



נתחיל מהפונקציות של ה-Anti Debugging: בפונקציה הראשונה יש בדיקה האם יש דיבאגר בעזרת ה-PEB, (קיצור של Process Environment Block) זה מבנה שיש לכל process. כפי שרואים בתמונה ניגשים ל-fs:[30]+2 במידה וזה 0 אין דיבאגר, במידה וזה 1 זה אומר שיש דיבאגר.

על מנת לעקוף את ה-Anti Debug הזה, אפשר לשנות את הביט ל-0 ככה שיחשוב שאין דיבאגר או כמו בשלב השני לעשות patch ל-ret.

כפי שרואים במידה ויש דיבאגר הוא כותב על ה-0x3000 בתים ראשון ב-main 0x90 שזה NOP:

```
.text:00CE17F0 ; =============== S U B R O U T I N E =======================================
.text:00CE17F0
.text:00CE17F0 ; Attributes: noreturn
.text:00CE17F0
.text:00CE17F0 ; DWORD __stdcall d4d_antiDebugThread(LPVOID lpThreadParameter)
.text:00CE17F0 d4d_antiDebugThread proc near           ; DATA XREF: d4d_antidebug+6↓o
.text:00CE17F0
.text:00CE17F0 lpThreadParameter= dword ptr  4
.text:00CE17F0
.text:00CE17F0                 push    esi
.text:00CE17F1                 mov     esi, large fs:30h
.text:00CE17F8
.text:00CE17F8 loc_CE17F8:                             ; CODE XREF: d4d_antiDebugThread+C↓j
.text:00CE17F8                                         ; d4d_antiDebugThread+25↓j
.text:00CE17F8                 cmp     byte ptr [esi+2], 0
.text:00CE17FC                 jz      short loc_CE17F8
.text:00CE17FE
.text:00CE17FE
.text:00CE17FE                 push    3000h           ; size_t
.text:00CE1803                 push    90h             ; int
.text:00CE1808                 push    offset _main    ; void *
.text:00CE180D                 call    _memset
.text:00CE1812                 add     esp, 0Ch
.text:00CE1815                 jmp     short loc_CE17F8
.text:00CE1815 d4d_antiDebugThread endp
.text:00CE1815
```

הפונקציה השניה של האנטי דיבאג בודקת את ה-global flags, ברגע שיש דיבאגר אז יש שלושה דגלים של heap שדולקים:

Flag	Value
FLG_HEAP_ENABLE_TAIL_CHECK	0x10
FLG_HEAP_ENABLE_FREE_CHECK	0x20
FLG_HEAP_VALIDATE_PARAMETERS	0x40
Total	0x70

וגם פה דורסים את ה-0x3000 בתים הראשונים בפונקציית main עם 0x90:

```
.text:00CE1840 d4d_antiDebugThread_0 proc near         ; DATA XREF: d4d_antidebug_0+6↓o
.text:00CE1840
.text:00CE1840 lpThreadParameter= dword ptr  4
.text:00CE1840
.text:00CE1840                 push    esi
.text:00CE1841                 mov     esi, large fs:30h
.text:00CE1848
.text:00CE1848 loc_CE1848:                             ; CODE XREF: d4d_antiDebugThread_0+C↓j
.text:00CE1848                                         ; d4d_antiDebugThread_0+25↓j
.text:00CE1848                 test    byte ptr [esi+68h], 70h
.text:00CE184C                 jz      short loc_CE1848
.text:00CE184E                 push    3000h           ; size_t
.text:00CE1853                 push    90h             ; int
.text:00CE1858                 push    offset _main    ; void *
.text:00CE185D                 call    _memset
.text:00CE1862                 add     esp, 0Ch
.text:00CE1865                 jmp     short loc_CE1848
.text:00CE1865 d4d_antiDebugThread_0 endp
```

בפונקציה השלישית טוענים קובץ DLL לזיכרון אחרי שמפענחים אותו עם ההצפנה הזו שדומה ל-RC4 בדומה לשלב השני.

בתמונה זו מקבלים את המפתח ל-DLL המוצפן ואת הקובץ DLL שאותו צריך לפענח:

```
.text:00CE1250
.text:00CE1250 loc_CE1250:                          ; CODE XREF: d4d_loadDllToMemory+16↑j
.text:00CE1250                 push    ecx
.text:00CE1251                 push    offset keyForBuf
.text:00CE1256                 lea     ecx, [esp+18h+var_8]
.text:00CE125A                 call    sub_CE16B0
.text:00CE125F
.text:00CE125F
.text:00CE125F                 push    0C00h
.text:00CE1264                 push    offset dllFile
.text:00CE1269                 call    sub_CE15E0
.text:00CE126E
```

אם עוברים על כל הפונקציות בתוכו נמצא את הפונקציה VirtualAlloc כמו שניתן לראות בתמונה הבאה:

```
.text:00CE13E5
.text:00CE13E5                 mov     esi, [ebp+var_28]
.text:00CE13E8                 push    40h             ; flProtect
.text:00CE13EA                 push    3000h           ; flAllocationType
.text:00CE13EF                 push    dword ptr [esi+50h] ; dwSize
.text:00CE13F2                 push    eax             ; lpAddress
.text:00CE13F3                 call    ds:VirtualAlloc
.text:00CE13F9                 mov     PEFile, eax
.text:00CE13FE
.text:00CE13FE
.text:00CE13FE                 test    eax, eax
.text:00CE1400                 jz      loc_CE1572
.text:00CE1406
.text:00CE1406
.text:00CE1406                 push    dword ptr [esi+54h] ; size_t
.text:00CE1409                 push    offset dllFile  ; void *
.text:00CE140E                 push    eax             ; void *
.text:00CE140F                 call    _memmove
.text:00CE1414                 movzx   eax, word ptr [esi+14h]
.text:00CE1418                 add     esp, 0Ch
.text:00CE141B                 movzx   ebx, word ptr [esi+6]
.text:00CE141F                 add     eax, esi
.text:00CE1421                 mov     [ebp+var_4], eax
.text:00CE1424                 test    ebx, ebx
```

כפי שניתן לראות אחרי ההקצאה מעתיקים את הקובץ DLL.

באותה הזדמנות גם בוצע שימוש ב-010 editor כדי לשמור את הקובץ DLL לדיסק לפני שמבצעים את ההעלאה שלו לזיכרון ומשנים relocations וכו'.

יש אפשרות לכתוב סקריפט ב-IdaPython שיפענח בשבילנו את הקובץ DLL עם האלגוריתם של ההצפנה שהושג בשלב השני:

```
import struct

def crypt(buf, key):
    tmp = 0
    pos = 0
    newbuf = ""
    box = []
    for i in range(0x100):
        box.append(i)

    for i in range(0x100):
        pos = (pos + ord(key[i%len(key)]) + box[i]) &0xff
        tmp = box[i]
        box[i] = box[pos]
        box[pos] = tmp

    pos = 0
    k = 0
    for i in range(len(buf)):
        tmp = (pos + 1) & 0xff
        pos = (box[tmp] + k) & 0xff
        k = tmp
        pos1 = (box[pos] + box[tmp]) & 0xff
        buf[i] ^=box[pos1]

    return buf

key = "\xB7\x9F\x0F\xEE\x8B\xB9\x70\x44\xA7\x7C\xB5\xBF\xD3\x4B\xEA\xD4"
addr = ScreenEA() #the cursor of the enrypted dll buffer
buf = ""
for i in xrange(768): # size of dll in dwords
    buf += struct.pack("<I",Dword(addr+i*4))

with open("mydll.dll", "wb") as fp:
    fp.write(crypt(bytearray(buf), key))
    print "dll written to disk"
```

לאחר שניתחנו את כל מה שנמצא ב-init_term אפשר לחזור לפונקציה main, אחרי שעשינו Patch לשתי הפונקציות האלה כמו שרואים בתמונה.

```
00CE17F0      C3                      ret
00CE17F1      64 8B 35 30 00 00 00    mov esi,dword ptr fs:[30]
00CE17F8      80 7E 02 00             cmp byte ptr ds:[esi+2],0
00CE17FC    ^ 74 FA                   je third.CE17F8
00CE17FE      68 00 30 00 00          push 3000
00CE1803      68 90 00 00 00          push 90
00CE1808      68 00 10 CE 00          push third.CE1000
00CE180D      E8 FE 0D 00 00          call third.CE2610
00CE1812      83 C4 0C                add esp,C
00CE1815    ^ EB E1                   jmp third.CE17F8

00CE1840      C3                      ret
00CE1841      64 8B 35 30 00 00 00    mov esi,dword ptr fs:[30]
00CE1848      F6 46 68 70             test byte ptr ds:[esi+68],70
00CE184C    ^ 74 FA                   je third.CE1848
00CE184E      68 00 30 00 00          push 3000
00CE1853      68 90 00 00 00          push 90
00CE1858      68 00 10 CE 00          push third.CE1000
00CE185D      E8 AE 0D 00 00          call third.CE2610
00CE1862      83 C4 0C                add esp,C
00CE1865    ^ EB E1                   jmp third.CE1848
```

לאחר מכן נחזור אל הפונקציית main. כאשר אנחנו מסתכלים על הפונקציית main אפשר לשים לב לדברים הבאים:

טוענים קובץ DLL וקוראים לפונקציה GetComputerNameW:

```
.text:00CE1035                 lea     eax, [esp+28h+LibFileName]
.text:00CE1039                 push    eax             ; lpLibFileName
.text:00CE103A                 call    ds:LoadLibraryA
.text:00CE1040                 mov     esi, eax        ; kernel32
.text:00CE1042                 test    esi, esi
.text:00CE1044                 jz      loc_CE10CA
.text:00CE104A
.text:00CE104A
.text:00CE104A                 push    ecx
.text:00CE104B                 lea     eax, [esp+2Ch+ProcName]
.text:00CE104F                 mov     [esp+2Ch+ProcName], 0
.text:00CE1054                 xorps   xmm0, xmm0
.text:00CE1057                 mov     edx, offset unk_CFB791
.text:00CE105C                 push    eax
.text:00CE105D                 mov     ecx, offset aGetcomputernamew ; "GetComputerNameW"
.text:00CE1062                 movups  [esp+30h+var_10], xmm0
.text:00CE1067                 call    d4d_decodeStrings
.text:00CE106C                 add     esp, 8
.text:00CE106F
.text:00CE106F
.text:00CE106F                 lea     eax, [esp+28h+ProcName]
.text:00CE1073                 push    eax             ; lpProcName
.text:00CE1074                 push    esi             ; hModule
.text:00CE1075                 call    ds:GetProcAddress
.text:00CE107B                 test    eax, eax
.text:00CE107D                 jz      short loc_CE10CA
.text:00CE107F
.text:00CE107F
.text:00CE107F                 push    offset lpnSize
.text:00CE1084                 push    offset lpBuffer
.text:00CE1089                 call    eax
```

כאשר ניסינו להריץ את זה, לא הצלחנו לקבל את השם של המחשב, אחרי שכתבנו על ידי patch את שם המשתמש שלנו הבנו שהוא משתמש בשם הזה בתור מפתח להצפנה הזו שדומה ל-RC4 כמו שבוצע בשלב השני.

כמובן שהמפתח לא היה נכון, אז היה צריך לחשוב עוד קצת מה לעשות. זוכרים את הקובץ שטענו לזיכרון ב-init_term? זהו קובץ DLL מיוחד שמכיל גם פונקציה בשם GetComputerNameW...

Functions window

Function name	Segment
GetComputerNameW	.text
DllEntryPoint	.text

קובץ ה-DLL שטוענים בקוד הוא GetComputerNameW של kernel32.dll אנחנו צריכים לגרום לקובץ DLL שהוקצה בתחילת התוכנית לרוץ. על מנת לעשות את זה משנים את call eax שיצביע לפונקציה שאנחנו רוצים לקרוא באמת.

זו הפונקציה GetComputerNameW הנכונה:

```
.text:00061000
.text:00061000                 ; BOOL __stdcall GetComputerNameW(LPWSTR lpBuffer, LPDWORD nSize)
.text:00061000                                 public GetComputerNameW
.text:00061000                 GetComputerNameW proc near              ; DATA XREF: .rdata:off_62088↓o
.text:00061000
.text:00061000                 computerName    = __m128i ptr -14h
.text:00061000                 lpBuffer        = dword ptr  8
.text:00061000                 nSize           = dword ptr  0Ch
.text:00061000
.text:00061000 55                              push    ebp
.text:00061001 8B EC                           mov     ebp, esp
.text:00061003 64 A1 30 00 00 00               mov     eax, large fs:30h
.text:00061009 83 EC 14                        sub     esp, 14h
.text:0006100C 8B 40 10                        mov     eax, [eax+10h]
.text:0006100F 8B 50 44                        mov     edx, [eax+44h]  ; filePath
.text:00061012 66 83 3A 20                     cmp     word ptr [edx], 20h
.text:00061016 74 18                           jz      short loc_61030
.text:00061018 0F 1F 84 00 00 00 00 00         nop     dword ptr [eax+eax+00000000h]
```

בפונקציה הזו ניגשים ל-PEB בכדי לקבל את ה-path של הקובץ, השורות הראשונות בקטע קוד מחפשות אם יש פרמטר בשורת הפקודה. המטרה היא למצוא את המחרוזת הנכונה בשורת הפקודה כדי לקבל את התוצאה שאנו רוצים.

כפי שרואים בקטע קוד הזה, מתבצעת השוואה בין התוצאה שיצאה לנו ל-מה שרוצים שיצא בקוד:

```
EIP  001E10D2   8B 01          mov eax,dword ptr ds:[ecx]
     001E10D4   3B 02          cmp eax,dword ptr ds:[edx]
     001E10D6   75 62          jne 1E113A
     001E10D8   83 C1 04       add ecx,4
     001E10DB   83 C2 04       add edx,4
     001E10DE   83 EE 04       sub esi,4
     001E10E1   73 EF          jae 1E10D2

eax=002B1100
dword ptr[edx]=[0015FD3C]=341200FE
001E10D4

Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct

Address   Hex                                           ASCII
001E2000  48 CC 37 29 AF 87 FD 45 86 8D F9 EA E6 0D 1C 4D  H7) .ýE..ùêæ..M
```

הדרך הקשה היא לנסות לגלות מה היה בשורת הפקודה, הדרך הקלה היא לעשות patch ולגרום להשוואה להצליח. אנו נבחר בדרך הקלה...

לאחר השינוי של הבאפר שמשווים נמשיך בקטע קוד:

```
Address   Hex                                           ASCII
0015FD3C  48 CC 37 29 AF 87 FD 45 86 8D F9 EA E6 0D 1C 4D  H7) .ýE..ùêæ..M
```

כפי שרואים בקטע קוד הבא היה צריך לעשות Patch ל-ja ו-jb כדי להגיע למקום הנכון בו מחושב "השם משתמש" שאנחנו צריכים בכדי לקבל את הסיסמא:

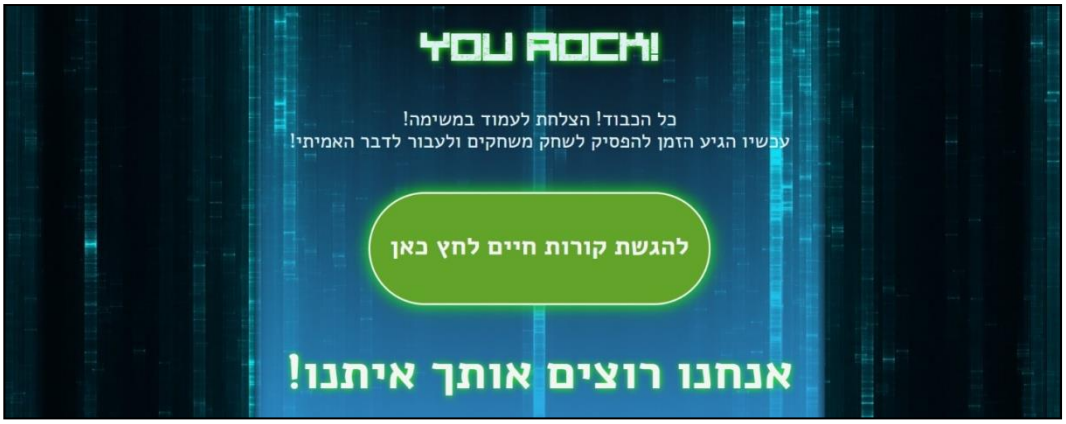
```
     001E10E3   8B 4D 08          mov ecx,dword ptr ss:[ebp+8]
     001E10E6   8D 55 FB          lea edx,dword ptr ss:[ebp-5]
     001E10E9   8D 41 0F          lea eax,dword ptr ds:[ecx+F]
     001E10EC   3B CA             cmp ecx,edx
     001E10EE   77 2B             ja 1E111B
     001E10F0   8D 55 EC          lea edx,dword ptr ss:[ebp-14]
     001E10F3   3B C2             cmp eax,edx
     001E10F5   72 24             jb 1E111B
     001E10F7   8B F2             mov esi,edx
     001E10F9   BA 10 00 00 00    mov edx,10
     001E10FE   2B F1             sub esi,ecx
EIP  001E1100   8A 04 0E          mov al,byte ptr ds:[esi+ecx]
     001E1103   8D 49 01          lea ecx,dword ptr ds:[ecx+1]
     001E1106   FE C0             inc al
     001E1108   88 41 FF          mov byte ptr ds:[ecx-1],al
     001E110B   83 EA 01          sub edx,1
     001E110E   75 F0             jne 1E1100
     001E1110   5F                pop edi
     001E1111   8D 42 01          lea eax,dword ptr ds:[edx+1]
     001E1114   5E                pop esi
     001E1115   8B E5             mov esp,ebp
     001E1117   5D                pop ebp
     001E1118   C2 08 00          ret 8
```

הלולאה מוסיפה 1 לכל בית מהבאפר שהשווינו קודם וזה יהיה המפתח להצפנה שדומה ל-RC4 שיתן את הסיסמא הנכונה שאנו צריכים:

```
May you enter Deep And Dreamless Slumber.
May you enter Deep And Dreamless Slumber.

Great Job!
Your password is:

A_Calm_Lion_President_Sends_His_Regards
```



קישורים לקריאה נוספת

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

על המחברים

- **D4D:** עוסק בתחום ה-Reverse Engineering - בחברת IronSource במחלקת ה-Security ואוהב לחקור משחקי מחשב והגנות, לכל שאלה שיש או ייעוץ ניתן לפנות אלי דרך:
  - שרת ה-IRC של Nix בערוץ: #reversing
  - או באתר: www.cheats4gamer.com
  - או בכתובת האימייל: llcashall@gmail.com.
- **תומר זית (RealGame):** חוקר אבטחת מידע בחברת F5 Networks וכותב Open Source.
  - אתר אינטרנט: http://www.RealGame.co.il
  - אימייל: realgam3@gmail.com
  - GitHub: https://github.com/realgam3