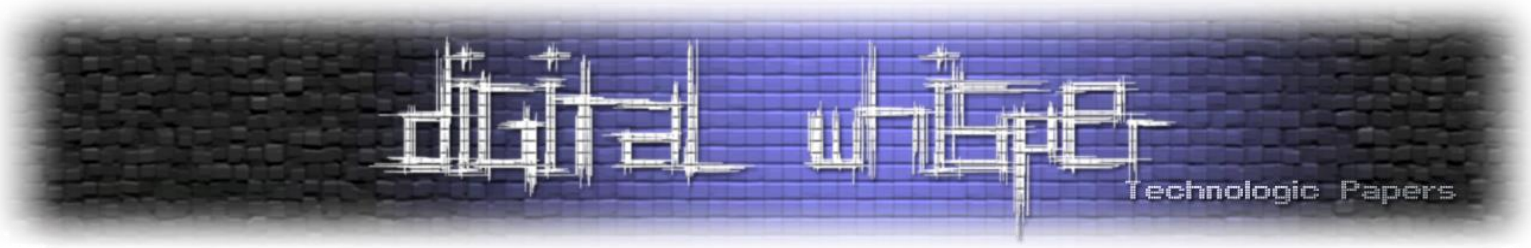


Manual Unpacking

מאת Zerith (אורי / Uri Farkas)

הקדמה

בגיליון שעבר קראתם על Manual Packing או "אריזה ידנית" במאמר המושקע מאת הלל חימוביץ'(HLL) . מאמר זה ידבר על הפעולה ההפוכה בדיוק - Unpacking, החזרת התוכנה למצבה המקורי לפני שארזו אותה.

למה בעצם משתמשים ב-Packers? אריזת התוכנה מטרתה למנוע מעיניים לא רצויות לחקור את התוכנה ולגלות איך היא עובדת - לגלות פרצות אבטחה, לעקוף מנגנוני אבטחה וכדו'. לדוגמא, הרבה ווירוסים ו-Malwares משתמשים ב-Packers כדי למנוע מחוקרי ווירוסים לחקור את הווירוס ולמצוא דרך להשמיד אותו/למצוא את מפעילו. ללא אריזה כל חוקר ווירוס ממוצע יוכל לפתוח את הווירוס ב-debugger ולחקור את קוד האסמבלי שלו ללא בעיה בכלל - זאת תהיה משימה פשוטה למדי. משחקים שונים משתמשים באריזה כדי למנוע מהאקרים לעשות "צ'יטים" או למצוא פרצות אבטחה במשחק ולנצל אותן.

Packers מוכרים ומסחריים בהם משתמשים Malware ומשחקים רבים הם Themida, Winlicense. קיימים לא מעט כלים מסחריים למשימה זאת. לפעמים אפילו משתמשים בכמה פאקרים שונים על אותו הקובץ על מנת להגיע לאבטחה גדולה אפילו עוד יותר.

במאמר זה נדבר על כמה טכניקות נפוצות של Anti-Debugging וכן על שיטות להקשות על ביצוע פעולת Unpacking. תוכנת המטרה שלנו היום היא UnpackMe ברמה בינונית שכתבה Lena151. ניתן להוריד את התוכנה בכתובת http://tuts4you.com/download.php?view.1114

הכלים שנצטרך:

- OllyDbg - כל גרסא מתחת ל-2.0 תתאים.
- ImpRec - תוכנה לתיקון IAT שהושחתו.
- LordPE - כלי לעריכת כותרות PE Headers של קבצי Object של מיקרוסופט.
- PEiD - כלי חינמי המיועד לזיהוי חתימות של Packers, הצפנות וכו'. ניתן להורדה בכתובת http://www.peid.info

קצת רקע

תהליך ביטול האריזה:

1. **שחזור הקוד המקורי**: כדי לארוז קובץ עלינו להצפין חלקים מהתוכנה ולשבש את הקוד. משום שה-Packer הוא גם ה-Unpacker של התוכנה (הוא משחזר את הקוד המקורי של התוכנה כדי להריץ אותה) - כל מה שעלינו לעשות על מנת להפוך את תהליך האריזה הוא לעקוב אחרי התוכנה שלב-שלב עד שכל ההצפנות בוטלו וכל קוד התוכנה המקורית שוחזר, ולשמור את התוצאה. כותב ה-Packer עשוי לשים בשלב זה מלכודות שונות על מנת להקשות על התהליך.
2. **שחזור נקודת הכניסה**: מכיוון שנקודת הכניסה של ה-Packer שונה מנקודת הכניסה המקורית של התוכנה, עלינו להגיע לנקודת הכניסה המקורית של התוכנה (Original Entry Point - OEP). אפשר להשיג זאת ע"י מעקב אחרי הקוד הרץ, משום שה-Packer **חייב** לקפוץ לנקודת הכניסה המקורית של התוכנה ולהריץ אותה ברגע כלשהו!
3. **תיקון Import Address Table**: הדבר האחרון שיש לעשות בתהליך ביטול האריזה הוא לתקן את ה-Import Address Table (בקיצור IAT), במקרה ששובשה על ידי ה-Packer.

נרחיב מעט על IAT - **Import Address Table** (בעברית: **טבלת פונקציות מיובאות**).

ספריות מיובאות הן DLL-ים (Dynamic Link Libraries) שקובץ ה-EXE מקושר אליהן. כתובות של פונקציות בתוך הספריות הללו אינן קבועות, מכיוון שיוצאות כל הזמן גרסאות חדשות של הספריות והכתובות משתנות בין הגרסאות. לפיכך התוכנה אינה יכולה להשתמש בכתובות קבועות מראש, וחייבת דרך כלשהי לגשת לפונקציות מבלי לקמפל מחדש את התוכנה או לבדוק את כתובתן בזמן ריצה.

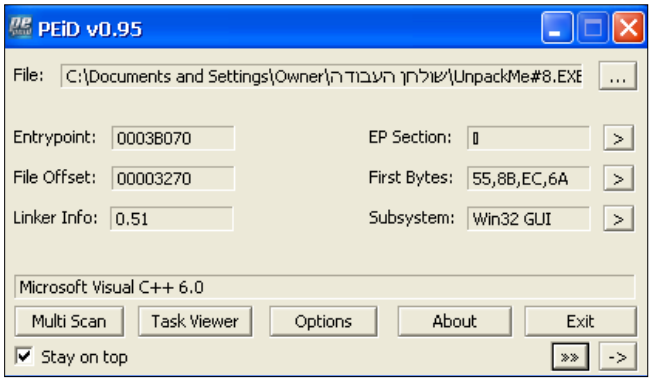
כל זה נעשה על ידי ה-IAT, ה-IAT הוא טבלה של מצביעים לפונקציות המתמלא ע"י ה-Windows Loader בעת טעינת הקובץ והספריות המקושרות אליו לזיכרון, במקום לשנות כל קריאה לפונקציה שתקרא לכתובת המתאימה, פשוט נקרא לכתובת השמורה ב-IAT.

ה-Packer לעיתים משנה את ה-IAT ומפנה את הכתובות לפונקציות משלו אשר קוראות בעקיפין לכתובות האמיתיות של ה-Imports, על מנת להקשות עלינו עוד יותר. עלינו לתקן את ה-IAT במקרה שהיא שונתה על ידי ה-Packer.

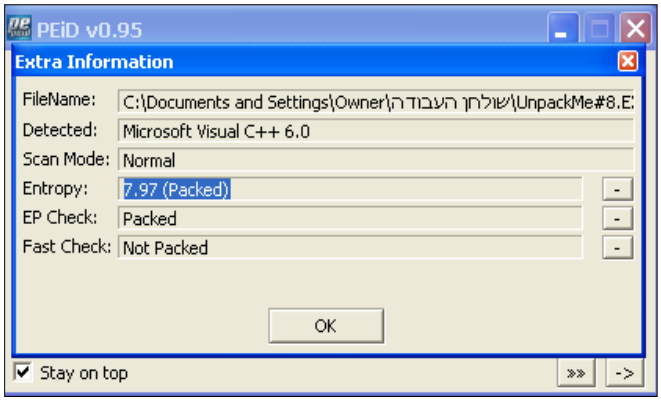
מתחילים

בהרצה ראשונה של המטרה מחוץ ל-Debugger, ניתן להבחין כי נפתח חלון פשוט הנראה כמו פנקס הרשימות. נפתח את התוכנה עם PEiD ונראה האם הוא מזהה משהו.

בבדיקה ראשונית, PEiD לא מוצא Packer וכותב לנו שהתוכנה נכתבה ב-Microsoft Visual C++ 6.0.



אך אם נלחץ על Extra Information ([>>]), נוכל לראות כי הקובץ אכן ארוז!

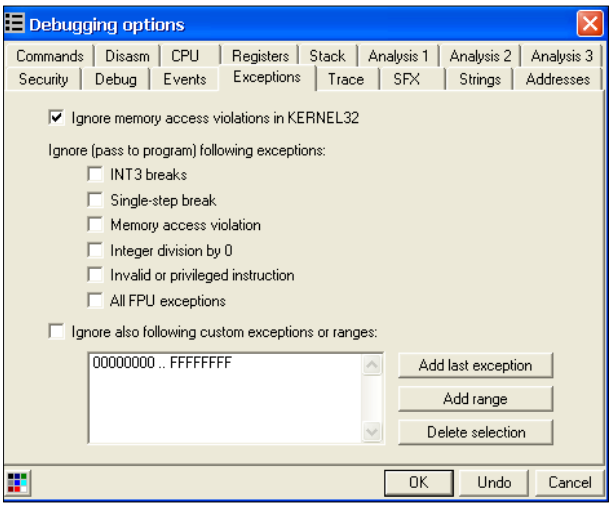


אחרי שראינו כי הקובץ ארוז, ננסה להריצו ב-Olly. בניסיון הראשון להריץ את התוכנה ב-Olly, התוכנה מגיע לחריגה שבה אינה מסוגלת לטפל, וקורסת.

מה זה בעצם אומר? אנו שמים לב כי זרימת התוכנה מתחת ל-Debugger שונה מזרימת התוכנה מחוצה לו. זה סימן מובהק שהקובץ מוגן בהגנת Anti-Debugging. Anti-Debugging הן טכניקות שנועדו לזהות את הדיבאגר, ולשבש את זרימת התוכנה כתוצאה מזיהויו. נסביר לכם על כמה מטכניקות אלו בהמשך.

בניסיון לראות את מקור החריגה, נבטל את העברת כל החריגות לתוכנה ב-Olly, ונראה אם יש חריגות עוד לפני החריגה שלא ניתן לטפל בה.

ניתן לעשות זאת על ידי לחיצה על Options -> Debugging Options -> Exceptions והורדת סימון ה-V מכל האפשרויות חוץ מ-Ignore Memory Access violations in Kernel32. (משום שלא ממש מעניין אותנו אם יש חריגות שלא קשורות לקוד שלנו).



אחרי שביטלנו את העברת כל החריגות לטיפול התוכנה, נריץ מחדש את התוכנה....וניתקל בחריגה!

אנחנו עדיין לא יודעים שחריגה זו היא הגורמת לחריגה שראינו קודם ולכן נמשיך להריץ את התוכנה. אחרי הרצה שנייה נתקל בעוד חריגה. אם נמשיך להריץ את התוכנה נגיע לחריגה השלישית - ונקרוס. בבדיקה ניתן לראות כי אחרי החריגה השנייה מסלול הקוד תמיד מגיע לחריגה השלישית. אנו למדים כי ככל הנראה עלינו למנוע את התרחשות החריגה השנייה כדי לפתור את הקריסה.

אנחנו בחריגה השנייה, אם נלך קצת לאחור בקוד (Backtrace) ונרצה לראות את מקור החריגה - נתקל בקריאה מוזרה, CALL EAX. ואחריה בדיקה של הערך החוזר וקפיצה בהתאם.

מדוע קריאה זו חשודה? משום שאנחנו לא יכולים לדעת מראש איזה פונקציה תיקרא פה. התוכן של EAX נקבע רק בזמן ריצה. אנחנו לא יכולים לדעת מה היה התוכן של EAX ברגע שנקרא לפני ריצת התוכנה. טריק זה הינו דרך של ה-Packer להסוות קריאות שהוא לא רוצה שנראה.

נשים Breakpoint על הקריאה כדי לחקור אותה. נפעיל מחדש את התוכנה, ונריץ עד הקריאה.

```
00291228   F3:A4          REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:
0029122A   8BF3           MOV ESI,EBX
0029122C   8DBD 491F0010  LEA EDI,DWORD PTR SS:[EBP+10001F49]
00291232   012F           ADD DWORD PTR DS:[EDI],EBP
00291234   016F 04        ADD DWORD PTR DS:[EDI+4],EBP
00291237   016F 08        ADD DWORD PTR DS:[EDI+8],EBP
0029123A   8D8D 2D1F0010  LEA ECX,DWORD PTR SS:[EBP+10001F2D]
00291240   51             PUSH ECX
00291241   E8 57010000    CALL 0029139D
00291246   6A 00          PUSH 0
00291248   56             PUSH ESI
00291249   E8 74050000    CALL 002917C2
0029124E   8B85 291F0010  MOV EAX,DWORD PTR SS:[EBP+10001F29]
00291254   85C0           TEST EAX,EAX
00291256 v 74 07          JE SHORT 0029125F
00291258   FFD0           CALL EAX                                 kernel32.IsDebuggerPresent
0029125A   85C0           TEST EAX,EAX
0029125C v 74 01          JE SHORT 0029125F
0029125E   4B             DEC EBX
0029125F   8B4E 2C        MOV ECX,DWORD PTR DS:[ESI+2C]
00291262   898D 591F0010  MOV DWORD PTR SS:[EBP+10001F59],ECX
00291268   6A 40          PUSH 40
0029126A   68 00100000    PUSH 1000
0029126F   51             PUSH ECX
00291270   6A 00          PUSH 0
00291272   FF95 651F0010  CALL DWORD PTR SS:[EBP+10001F65]
00291278   8985 551F0010  MOV DWORD PTR SS:[EBP+10001F55],EAX
0029127E   56             PUSH ESI
0029127F   E8 F6030000    CALL 0029167A
00291284   8D8D D11D0010  LEA ECX,DWORD PTR SS:[EBP+10001DD1]
0029128A   85C0           TEST EAX,EAX
0029128C v 0F85 94000000  JNZ 00291326
00291292   56             PUSH ESI
00291293   E8 40030000    CALL 002915D8
00291298   56             PUSH ESI
00291299   E8 55020000    CALL 002914F3
0029129E   90             NOP
0029129F   90             NOP
002912A0   90             NOP
002912A1   90             NOP
002912A2   90             NOP
002912A3   90             NOP
002912A4   6A 01          PUSH 1
002912A6   56             PUSH ESI
```

אנו רואים כי אכן צדקנו והקריאה באמת הייתה חשודה. בכתובת זו ישנה קריאה לפונקציה IsDebuggerPresent (הבודקת אם התוכנה רצה מתחת לדיבאגר), בדיקה של הערך החוזר - וקפיצה בהתאם, טכניקת Anti-Debugging נפוצה.

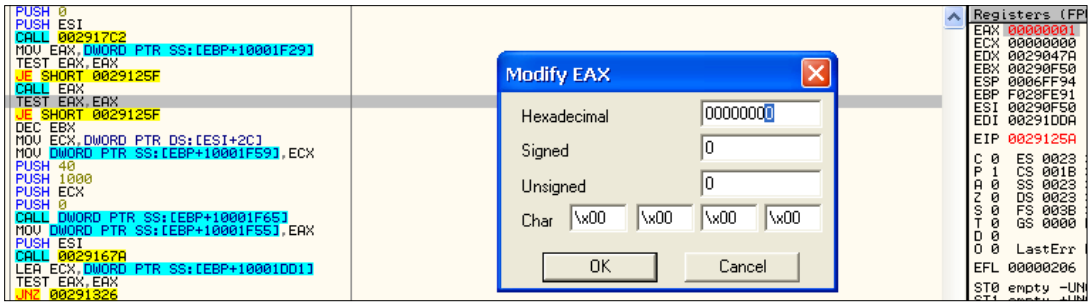
הסבר קצר על IsDebuggerPresent:

הפונקציה IsDebuggerPresent משתמשת במבנה נתונים הנקרא Process Environment Block (מבנה נתונים המכיל פרטים על התהליך הרץ), המכיל בין השאר שדה הנקרא "IsDebugged" שמייצג האם התהליך רץ מתחת לדיבאגר - או לא.

```
7C8130A1    90            NOP
7C8130A2    90            NOP
7C8130A3 $  64:A1 180000  MOV EAX,DWORD PTR FS:[18]
7C8130A9 .  8B40 30       MOV EAX,DWORD PTR DS:[EAX+30]
7C8130AC .  0FB640 02     MOVZX EAX,BYTE PTR DS:[EAX+2]
7C8130B0 .  C3            RETN
7C8130B1    90            NOP
7C8130B2    90            NOP
7C8130B3    90            NOP
7C8130B4    90            NOP
```

- FS - הוא הסגמנט המייצג את כתובות הזיכרון המתחילות מ-0x7FFDF000 (כברירת מחדל).
- FS[18] הוא מצביע ל-TEB (Thread Environment Block, בדומה ל-PEB מכיל פרטים על החוט הרץ).
- TEB+0x30 מצביע ל-PEB.
- [PEB+0x02] הוא השדה IsDebugged ב-PEB.

אנו רואים כי IsDebuggerPresent מחזיר את הערך של שדה IsDebugged. שינוי של ערך זה ב- Process Environment Block (PEB) לא משנה דבר מבחינת פעולת התוכנה ולא ימנע מאיתנו לעשות Debugging לתוכנה. כל מה שעלינו לעשות על מנת לעבור את הקריאה ל-IsDebuggerPresent הוא לשנות את הערך החוזר מהפונקציה מ-1 ל-0.



נריץ את הפונקציה עד הסוף (עד ה-RETN), ונראה שאין חריגה!

שיטה נוספת של anti-debugging: שילוב INT3

לפני שנמשיך בניתוח התוכנה הנתונה נציג טכניקה נפוצה נוספת של Anti-Debugging, שהיא הכנסת ההוראה INT3 באמצע קוד רגיל.

INT3 הינו Software Breakpoint. במקרה של Olly, ושל Debuggers אחרים - כאשר אנחנו שמים Breakpoint בתוכנה (Software BP) ה-Debugger מכניס את ההוראה INT3 כבית הראשון של ההוראה, ובזמן ריצה כאשר המחשב נתקל בהוראה INT3 הריצה נעצרת והשליטה חוזרת למשתמש ול-Debugger.

מה קורה כשהקוד עצמו מכיל את ההוראה INT3? מחוץ לדיבאגר ההוראה תיצור חריגה וזרימת התוכנה תעבור ל-Exception Handler. כאשר התוכנה בשליטת הדיבאגר, הוא חושב שה-INT3 הוא בעצם אחד מה-Software Breakpoints שלו - השליטה לא מועברת ל-Exception Handler והקוד ממשיך מאותו המקום.

למתעניינים בטכניקות Anti-Debugging נוספות, נמליץ על קריאת מאמרים בנושא. מאמר מומלץ לדוגמא הוא http://www.veracode.com/images/pdf/whitepaper_antidebugging.pdf.

נמשיך לחקור את התוכנה. "נצעד" בקוד וננסה להבין אותו. אחרי הפונקציה שנתקלנו בה קודם, אנחנו מגיעים אל הקוד הבא:

```
01015A3F   C1E8 08          SHR EAX,8
01015A42   87D3             XCHG EBX,EDX
01015A44 v EB 01            JMP SHORT UnpackMe.01015A47
01015A46   630F             ARPL WORD PTR DS:[EDI],CX
01015A48   CB               RETF                                  Far return
01015A49   87D2             XCHG EDX,EDX
01015A4B   0FCB             BSWAP EBX
01015A4D   0FC0E0           XADD AL,AH
01015A50   0FC0C7           XADD BH,AL
01015A53 v EB 01            JMP SHORT UnpackMe.01015A56
01015A55   D8EB             FSUBR ST,ST(3)
01015A57   0180 87C2C1C1    ADD DWORD PTR DS:[EAX+C1C1C287],ESI
01015A5D   4F               DEC EDI
01015A5E   86E5             XCHG CH,AH
01015A60 v EB 01            JMP SHORT UnpackMe.01015A63
01015A62   8DC0             LEA EAX,EAX                           Illegal use of register
01015A64   F0:09EB          LOCK OR EBX,EBP                       LOCK prefix is not allowed
01015A67   013CC0           ADD DWORD PTR DS:[EAX+EAX*8],EDI
01015A6A   FFB8             ???                                   Unknown command
01015A6C   87D9             XCHG ECX,EBX
01015A6E   D3CA             ROR EDX,CL
01015A70   D0EC             SHR AH,1
01015A72   C1D0 30          RCL EAX,30                            Shift constant out of range 1..31
01015A75   0FC1C0           XADD EAX,EAX
01015A78   D1C2             ROL EDX,1
01015A7A v EB 01            JMP SHORT UnpackMe.01015A7D
01015A7C   B7 87            MOV BH,87
01015A7E   D30F             ROR DWORD PTR DS:[EDI],CL
01015A80   C0FA 0F          SAR DL,0F
01015A83   C1C3 C0          ROL EBX,0C0                           Shift constant out of range 1..31
01015A86   C8 EFEB01        ENTER 0EBEF,1
01015A8A   5D               POP EBP
01015A8B   C1DA CA          RCR EDX,0CA                           Shift constant out of range 1..31
01015A8E   D1C1             ROL ECX,1
01015A90   86F1             XCHG CL,DH
01015A92   86C6             XCHG DH,AL
01015A94   86E1             XCHG CL,AH
01015A96 v EB 01            JMP SHORT UnpackMe.01015A99
01015A98   25 0FC80FCA      AND EAX,CA0FC80F
01015A9D   0FCB             BSWAP EBX
01015A9F   0FC0DE           XADD DH,BL
01015AA2   C1E0 0B          SHL EAX,0B
01015AA5   0FC0E9           XADD CL,CH
01015AA8   0FC1D9           XADD ECX,EBX
01015AAB   0FC9             BSWAP ECX
```

קוד זה נראה מאוד מוזר, זהו Obfuscation Code (קוד הטעייה) - קוד שנועד לבלבל את הריברסר. לרוב קטע קוד זה לא משפיע כלל על התוכנה ואין צורך בו.

כשאתם נתקלים בקוד מסוג זה, היזהרו מאוד לא לאבד שליטה על התוכנה. (להיכנס בתוך הקריאות ולא מעליהם, וכו'). אין הרבה דרכים לדעת מהו קוד הטעייה ולזהות אותו, אך עם הניסיון תדעו כבר להבחין כשתראו אחד.

דרך לדעת אם קיים קוד הטעייה בקובץ הוא להסתכל ב-PEiD, איפה שראינו שכתוב Entropy: 7.97 (Packed). Entropy בתרגום חופשי הוא "רמת המבולגנות" של הקובץ, כאשר 10 היא הדרגה הגבוהה ביותר.

אחרי הרבה קוד הטעייה, אנו מגיעים לפונקציה שכבר נראית בטוחה (ללא קוד הטעייה). חקירה של הפונקציה עד סופה, תביא אותנו לקריאה חשודה נוספת - CALL EAX. עם זאת, אנחנו רואים כי קריאה זאת אינה קריאה ל-API, וכאן עולה השאלה - למה ירצו להסוות קריאה זו?

```
010073B0   6A 70          PUSH 70
010073B2   68 98180001    PUSH UnpackMe.01001898
010073B7   E8 BF010000    CALL UnpackMe.01007568
010073BC   33DB           XOR EBX,EBX
010073BE   53             PUSH EBX
010073C0   8B3D CC100001  MOV EDI,DWORD PTR DS:[10010CC]      UnpackMe.01022254
010073C6   FFD7           CALL EDI
010073C8   66:8138 4D5A   CMP WORD PTR DS:[EAX],5A4D
010073CD   75 1F          JNZ SHORT UnpackMe.010073DA
010073CF   8B48 3C        MOV ECX,DWORD PTR DS:[EAX+3C]
010073D2   03C8           ADD ECX,EAX
010073D4   8139 50450000  CMP DWORD PTR DS:[ECX],4550
010073DA   75 12          JNZ SHORT UnpackMe.010073DA
010073DC   0FB741 18      MOVZX EAX,WORD PTR DS:[ECX+18]
010073E0   3D 0B010000    CMP EAX,10B
010073E5   74 1F          JE SHORT UnpackMe.010073F2
010073E7   3D 0B020000    CMP EAX,20B
010073EC   74 05          JE SHORT UnpackMe.010073DF
010073EE   895D E4        MOV DWORD PTR SS:[EBP-1C],EBX
010073F1   EB 27          JMP SHORT UnpackMe.01007406
010073F3   83B9 84000000  CMP DWORD PTR DS:[ECX+84],0E
010073FA   76 F2          JBE SHORT UnpackMe.010073DA
010073FC   33C0           XOR EAX,EAX
010073FE   3999 F8000000  CMP DWORD PTR DS:[ECX+F8],EBX
01007404   EB 0E          JMP SHORT UnpackMe.01007400
01007406   8379 74 0E     CMP DWORD PTR DS:[ECX+74],0E
01007408   76 E2          JBE SHORT UnpackMe.010073DA
0100740A   33C0           XOR EAX,EAX
0100740C   3999 E8000000  CMP DWORD PTR DS:[ECX+E8],EBX
01007412   0F95C0         SETNE AL
01007415   8945 E4        MOV DWORD PTR SS:[EBP-1C],EAX
01007418   895D FC        MOV DWORD PTR SS:[EBP-4],EBX
0100741B   6A 02          PUSH 2
0100741D   FF15 38130001  CALL DWORD PTR DS:[1001338]         msvcrt.__set_app_type
01007423   59             POP ECX
01007424   830D 9CAB0001  OR DWORD PTR DS:[100AB9C],FFFFFFFF
0100742B   830D A0AB0001  OR DWORD PTR DS:[100ABA0],FFFFFFFF
01007432   FF15 34130001  CALL DWORD PTR DS:[1001334]         msvcrt.__p__fmode
01007438   8B0D B89A0001  MOV ECX,DWORD PTR DS:[1009AB8]
0100743E   8908           MOV DWORD PTR DS:[EAX],ECX
01007440   FF15 30130001  CALL DWORD PTR DS:[1001330]         msvcrt.__p__commode
01007446   8B0D B49A0001  MOV ECX,DWORD PTR DS:[1009AB4]
0100744C   8908           MOV DWORD PTR DS:[EAX],ECX
0100744E   A1 2C130001    MOV EAX,DWORD PTR DS:[100132C]
01007453   8B00           MOV EAX,DWORD PTR DS:[EAX]
```

אנו רואים כי זו נקודת הכניסה המקורית של התוכנה! איך אנו יודעים זאת? אם פעם פתחתם קובץ שקומפל ב-Microsoft Visual C++ בדיבאגר, ראיתם בדיוק את נקודת הכניסה הזו. (ניתן להבחין בכך לפי הקריאות ל-msvcrt). כאמור, נקודה זו היא ה-Original Entry Point (OEP).

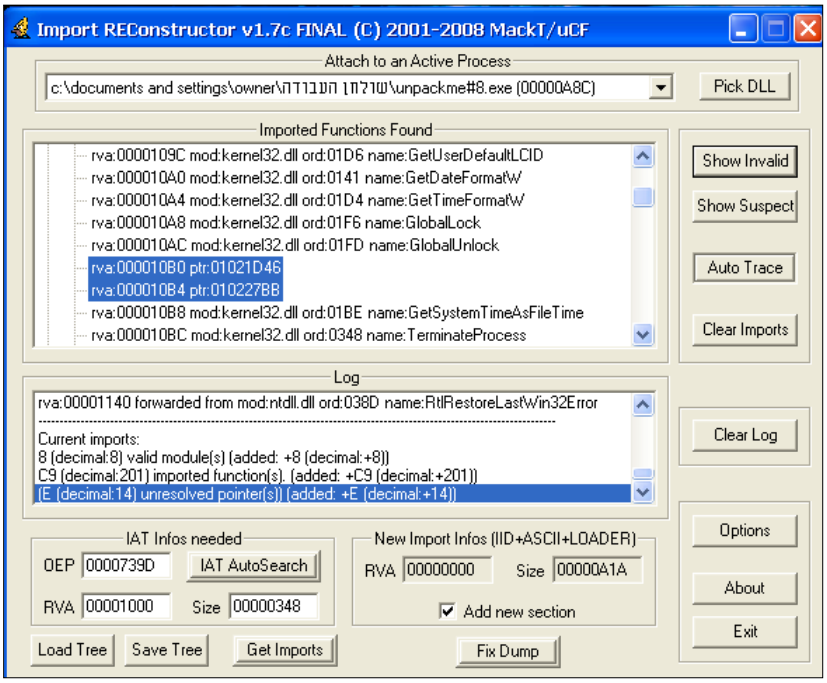
נפתח את LordPE, ונעשה לתהליך Dump, משום שהקוד האמיתי של התוכנה כבר לא ארוז. (Dump הוא העתקה של זיכרון התהליך הרץ ברגע נתון, לתוך קובץ חדש, למשל EXE).

על מנת לעשות Dumping מה שצריך לעשות זה לפתוח את LordPE, לבחור את התהליך הרצוי, לחיצה ימנית -> ולבחור Full Dump.

תיקון ה-IAT (Import Address Table)

נפתח את התוכנה ImpRec, תוכנה אשר מיועדת לתיקון Imports. נכניס את כל הערכים הרצויים (זכרו להחליף את ה-EP של ה-Packer ב-OEP!) ונלחץ על Get Imports. במקרה שלנו, הערך היחיד שיש להכניס ב-ImpRec הוא ה-RVA של ה-OEP שמצאנו. RVA הוא כתובת וירטואלית יחסית לכתובת הבסיס (ImageBase). מושג זה הוסבר במאמר של HLL על Manual Packing בגיליון שעבר.

נוודא שכל ה-Imports תקינים על ידי לחיצה על Show Invalid. במקרה שלנו יש Imports מושחתים ☹



נצטרך לתקן את הערכים המושחתים.

כמו שאנחנו רואים ב-ImpRec, ה-IAT שוכן ב-RVA 0x00001000. נפתח את ה-Debugger בכתובת זו (יש לזכור כי זאת כתובת יחסית ל-ImageBase, ובמקרה הזה ה-ImageBase הוא 0x01000000).

אנו רואים כי 0x010010B0 היא הכתובת של ה-Import הראשון ששונה. מכיוון שבתוכנה המקורית ה-IAT לא היה מושחת כנראה שה-Packer שינה בזמן ריצה את ה-IAT, אז בשביל לראות מתי הכתובת שונתה, נשים Hardware Breakpoint on write בכתובת 0x010010B0 ונריץ מחדש את התוכנה. (לא לשכוח לעקוף מחדש את IsDebuggerPresent).

אחרי ההרצה אנחנו מגיעים לנקודה בה 0x010010B0 שונתה. ניתן לרואת בקוד באזור כי ה-Packer בודק אילו Imports לשנות או לא. ננתח קוד זה.

```
0101CEB0    MOV EAX,DWORD PTR SS:[EBP+8]
0101CEB3    MOV ECX,DWORD PTR DS:[EAX]
```

פה כתובתה של הפונקציה המיובאת נשמרת ב-ECX וניתנת כפרמטר לפונקציה:

```
0101CEB5    PUSH ECX
0101CEB6    MOV ECX,DWORD PTR DS:[102D034]
0101CEBC    CALL UnpackMe.01022B4C
```

קריאה לפונקציה שבודקת אם עליה לשנות ב-IAT את הכתובת של הפונקציה הניתנה כפרמטר

```
0101CEC1    MOV DWORD PTR SS:[EBP-8],EAX
0101CEC4    CMP DWORD PTR SS:[EBP-8],0
```

בדיקה של הערך החוזר מהפונקציה - הפונקציה מחזירה את כתובתה של הפונקציה המחליפה במקרה ויש להחליף את הכתובת ב-IAT, ו-0 במקרה שלא.

```
0101CEC8    JE SHORT UnpackMe.0101CF0F
```

קפיצה בהתאם לתוצאה:

```
0101CECA    LEA EDX,DWORD PTR SS:[EBP-10]
0101CECD    PUSH EDX
0101CECE    PUSH 4
0101CED0    PUSH 4
0101CED2    MOV EAX,DWORD PTR SS:[EBP+8]
0101CED5    PUSH EAX
0101CED6    CALL DWORD PTR DS:[102872C]
```

קריאה לפונקציה kernel32.VirtualProtect על מנת לשנות את הגנת הדף על ארבעת בתי הכתובת ב-IAT. הפונקציה VirtualProtect משנה את הגנת הזיכרון הוירטואלי של התהליך בכתובת הניתנה כפרמטר במספר הבתים (שגם כן ניתנו כפרמטר). כברירת מחדל ההגנה היא PAGE_READEXECUTE (לא ניתן לכתוב לאזור הזיכרון), בקריאה זו ההגנה משתנה ל-PAGE_READWRITE:

```
0101CEDC    TEST EAX,EAX
0101CEDE    JNZ SHORT UnpackMe.0101CEEA
0101CEE0    MOV ECX,EF00000B
0101CEE5    CALL UnpackMe.0101FA32
```

קריאה זאת לא מורצת במקרה והקריאה ל-VirtualProtect הצליחה:

```
0101CEEA    MOV ECX,DWORD PTR SS:[EBP+8]
```

השגת מצביע לכתובת ב-IAT שיש לשנות:

```
0101CEED    MOV EDX,DWORD PTR SS:[EBP-8]
```

מצביע לכתובת המחליפה:

```
0101CEF0    MOV EAX,DWORD PTR DS:[EDX]
0101CEF2    MOV DWORD PTR DS:[ECX],EAX
```

כאן מתבצעת ההחלפה:

```
0101CEF4    LEA ECX,DWORD PTR SS:[EBP-C]
0101CEF7    PUSH ECX
0101CEF8    MOV EDX,DWORD PTR SS:[EBP-10]
0101CEFB    PUSH EDX
0101CEFC    PUSH 4
0101CEFE    MOV EAX,DWORD PTR SS:[EBP+8]
0101CF01    PUSH EAX
0101CF02    CALL DWORD PTR DS:[102872C]
```

kernel32.VirtualProtect - כאן משחזרים את ההגנה הקודמת של הדף על מנת למנוע בעיות בעתיד:

```
0101CF08     MOV DWORD PTR SS:[EBP-4],1
0101CF0F     MOV EAX,DWORD PTR SS:[EBP-4]
0101CF12     MOV ESP,EBP
0101CF14     POP EBP
0101CF15     RETN ; Exit
```

כל מה שעלינו לעשות על מנת למנוע מה-Packer לשנות את ה-Imports הוא לשנות את הקפיצה כדי שתמיד תקפוץ.

```
0101CEA4  75 0A          JNZ SHORT UnpackMe.0101CEB0
0101CEA6  B9 0A0000EF    MOV ECX,EF00000A
0101CEAB  E8 822B0000    CALL UnpackMe.0101FA32
0101CEB0  8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
0101CEB3  8B08           MOV ECX,DWORD PTR DS:[EAX]
0101CEB5  51             PUSH ECX
0101CEB6  8B0D 34D00201  MOV ECX,DWORD PTR DS:[102D034]
0101CEBC  E8 8B5C0000    CALL UnpackMe.01022B4C
0101CEC1  8945 F8        MOV DWORD PTR SS:[EBP-8],EAX
0101CEC4  837D F8 00     CMP DWORD PTR SS:[EBP-8],0
0101CEC8  EB 45          JMP SHORT UnpackMe.0101CF0F          MAGIC JUMP
0101CECA  8D55 F0        LEA EDX,DWORD PTR SS:[EBP-10]
0101CECD  52             PUSH EDX
0101CECE  6A 04          PUSH 4
0101CED0  6A 04          PUSH 4
0101CED2  8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
0101CED5  50             PUSH EAX
0101CED6  FF15 2C870201  CALL DWORD PTR DS:[102872C]          kernel32.VirtualProtect
0101CEDC  85C0           TEST EAX,EAX
0101CEDE  75 0A          JNZ SHORT UnpackMe.0101CEEA
0101CEE0  B9 0B0000EF    MOV ECX,EF00000B
0101CEE5  E8 482B0000    CALL UnpackMe.0101FA32
0101CEEA  8B4D 08        MOV ECX,DWORD PTR SS:[EBP+8]
0101CEED  8B55 F8        MOV EDX,DWORD PTR SS:[EBP-8]
0101CEF0  8B02           MOV EAX,DWORD PTR DS:[EDX]
0101CEF2  8901           MOV DWORD PTR DS:[ECX],EAX           UnpackMe.0100739D
0101CEF4  8D4D F4        LEA ECX,DWORD PTR SS:[EBP-C]
0101CEF7  51             PUSH ECX
0101CEF8  8B55 F0        MOV EDX,DWORD PTR SS:[EBP-10]
0101CEFB  52             PUSH EDX
0101CEFC  6A 04          PUSH 4
0101CEFE  8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
0101CF01  50             PUSH EAX
0101CF02  FF15 2C870201  CALL DWORD PTR DS:[102872C]          kernel32.VirtualProtect
0101CF08  C745 FC 010000 MOV DWORD PTR SS:[EBP-4],1
0101CF0F  8B45 FC        MOV EAX,DWORD PTR SS:[EBP-4]
0101CF12  8BE5           MOV ESP,EBP
0101CF14  5D             POP EBP
0101CF15  C3             RETN
```

לקפיצה מסוג זה קוראים Magic Jump והיא נפוצה בקרב Packers שונים. נריץ את התוכנה כדי שה-Packer יכתוב את כל כתובות ה-Imports המקוריים. נפתח מחדש את ImpRec ונכניס את אותם הערכים.

ו...הכול תקין! ☺ כל מה שנשאר לעשות הוא ללחוץ על Fix Dump ולבחור את ה-Dump שיצרנו קודם לכן, והתוכנה תרוץ ללא אריזה.

- המהדרין יכולים להכנס ל-LordPE ולחתוך את איזור הזיכרון שבו שכן ה-Packer כדי להקטין את גודל הקובץ, משום שאנחנו לא צריכים אותו יותר.

להמשך ההעמקה בתחום...

אם ברצונכם ללמוד עוד על התחום, הרשו לי להפנות אתכם למדריכים של Lena151, שלדעתי הם חובה לכל מתחיל בתחום: http://tuts4you.com/download.php?list.17