

Manual Unpacking

מאת Zerith (אורי)

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 חינומי המיועד לזיהוי חתימות של Packers, הצפנות וכו'. ניתן להורדה בכתובת <http://www.peid.info>

תהליך ביטול האריזה:

1. **שחזור הקוד המקורי:** כדי לארוז קובץ עלינו להצפין חלקים מהתוכנה ולשבש את הקוד. משום שה-Packer הוא גם 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. כתב 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.

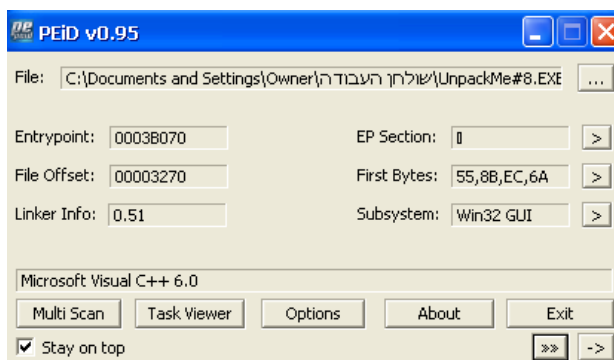
כל זה נעשה על ידי ה-IAT, ה-IAT הוא טבלה של מצביעים לפונקציות המתמלא ע"י ה-Windows Loader בעת טעינת הקובץ והספריות המקושרות אליו לזיכרון, במקום לשנות כל קריאה לפונקציה שתקרא לכתובת המתאימה, פשוט נקרא לכתובת השמורה ב-IAT.

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.

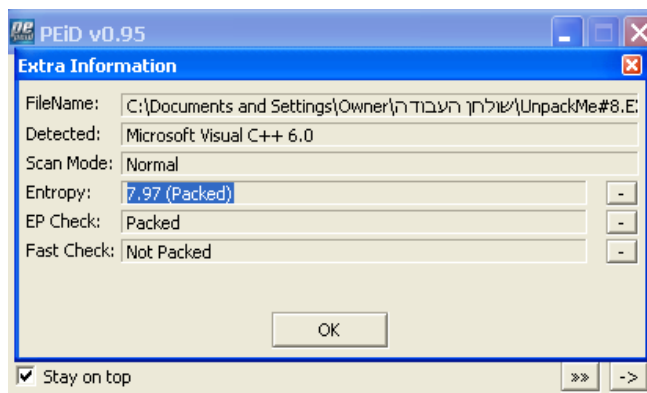
מתחילים

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!

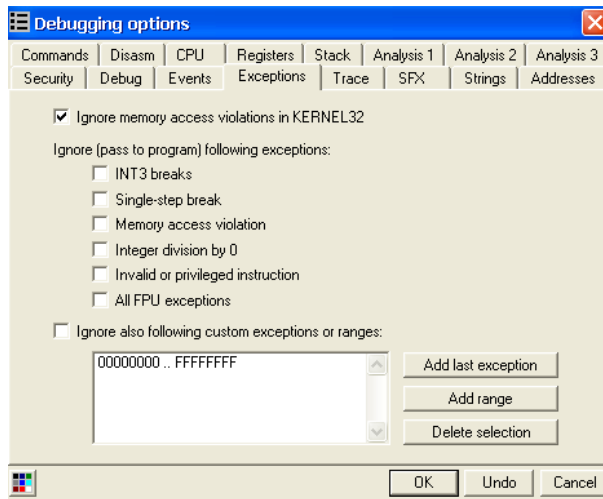


אחרי שראינו כי הקובץ ארוז, ננסה להריצו ב-Olly. בניסיון הראשון להריץ את התוכנה ב-Olly, התוכנה מגיע לחריגה שבה אינה מסוגלת לטפל, וקורסת.

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 -> Exceptions -> Debugging Options -> Options 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 F3:84 REP MOV8 BYTE PTR ES:[EDI],BYTE PTR DS:
00291229 8BF3 MOV ESI,EBX
0029122A 808D 491F0010 LEA EDI,DWORD PTR SS:[EBP+10001F49]
0029122B 012F ADD DWORD PTR DS:[EDI],EBP
0029122C 016F 04 ADD DWORD PTR DS:[EDI+4],EBP
0029122D 016F 08 ADD DWORD PTR DS:[EDI+8],EBP
0029122E 8D8D 2D1F0010 LEA ECX,DWORD PTR SS:[EBP+10001F2D]
0029122F 51 PUSH ECX
00291230 E8 57010000 CALL 00291390
00291231 6A 00 PUSH 0
00291232 56 PUSH ESI
00291233 E8 74050000 CALL 002917C2
00291234 8B85 291F0010 MOV EAX,DWORD PTR SS:[EBP+10001F29]
00291235 85C0 TEST EAX,EAX
00291236 74 07 JE SHORT 0029125F
00291237 FF00 CALL EBX
00291238 85C0 TEST EAX,EAX
00291239 74 01 JE SHORT 0029125F
0029123A 4B DEC EBX
0029123B 8B4E 2C MOV ECX,DWORD PTR DS:[ESI+2C]
0029123C 898D 591F0010 MOV DWORD PTR SS:[EBP+10001F59],ECX
0029123D 6A 40 PUSH 40
0029123E 68 00100000 PUSH 1000
0029123F 51 PUSH ECX
00291240 6A 00 PUSH 0
00291241 FF95 651F0010 CALL DWORD PTR SS:[EBP+10001F65]
00291242 8985 551F0010 MOV DWORD PTR SS:[EBP+10001F55],EAX
00291243 56 PUSH ESI
00291244 E8 F6030000 CALL 0029167A
00291245 8D8D D11D0010 LEA ECX,DWORD PTR SS:[EBP+10001DD1]
00291246 85C0 TEST EAX,EAX
00291247 0F35 94000000 JNB 00291326
00291248 56 PUSH ESI
00291249 E8 40030000 CALL 002915D8
0029124A 56 PUSH ESI
0029124B E8 55020000 CALL 002914F3
0029124C 90 NOP
0029124D 90 NOP
0029124E 90 NOP
0029124F 90 NOP
00291250 90 NOP
00291251 90 NOP
00291252 90 NOP
00291253 90 NOP
00291254 6A 01 PUSH 1
00291255 5C POP ECT

```

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.

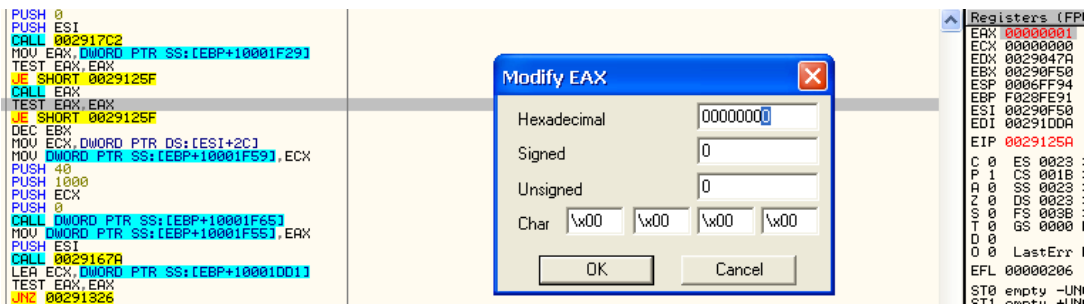
```

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.



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

שיטה נוספת של anti-debugging: שילוב INT3

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

ה-INT3 הינו Software 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 לדוגמה הוא http://www.veracode.com/images/pdf/whitepaper_antidebugging.pdf.

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

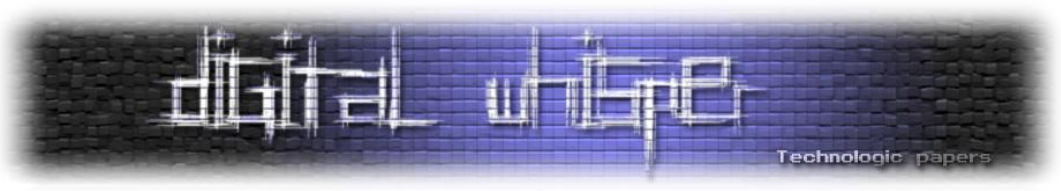
| | | | |
|----------|---------------|-------------------------------------|-----------------------------------|
| 01015A3F | C1E8 08 | SHR EAX,8 | |
| 01015A42 | 87D3 | XCHG EBX,EDX | |
| 01015A44 | EB 01 | JMP SHORT UnpackMe.01015A47 | |
| 01015A46 | 630F | ARPL WORD PTR DS:[EDI],CX | |
| 01015A48 | CB | RETN | Far return |
| 01015A49 | 87D2 | XCHG EDX,EDX | |
| 01015A4B | 0FCB | BSWAP EBX | |
| 01015A4D | 0FC8E0 | XADD AL,AH | |
| 01015A50 | 0FC0C7 | XADD BH,AL | |
| 01015A53 | EB 01 | JMP SHORT UnpackMe.01015A56 | |
| 01015A55 | D8EB | FSUBR ST,ST(3) | |
| 01015A57 | 01B0 87C2C1C1 | ADD DWORD PTR DS:[EAX+C1C1C287],ESI | |
| 01015A60 | 4F | DEC EDI | |
| 01015A5E | 86E5 | XCHG CH,AH | |
| 01015A60 | EB 01 | JMP SHORT UnpackMe.01015A63 | |
| 01015A62 | 3D0B | LEA EAX,EBX | Illegal use of register |
| 01015A64 | F8:09EB | LOCK OR EBX,EBB | LOCK prefix is not allowed |
| 01015A67 | 013CC0 | ADD DWORD PTR DS:[EAX+EAX*8],EDI | |
| 01015A6A | FF8E | CALL EBX | Unknown command |
| 01015A6C | 87D9 | XCHG ECX,EBX | |
| 01015A6E | D3CA | ROR EDX,CL | |
| 01015A70 | D8EC | 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 | EB 01 | JMP SHORT UnpackMe.01015A7D | |
| 01015A7C | B7 87 | MOV BH,87 | |
| 01015A7E | D30F | ROR DWORD PTR DS:[EDI],CL | |
| 01015A80 | C0FA 0F | SHR 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 | ROR EDX,0CA | Shift constant out of range 1..31 |
| 01015A8E | D1C1 | ROL ECX,1 | |
| 01015A90 | 86F1 | XCHG CL,DH | |
| 01015A92 | 86C5 | XCHG DH,AL | |
| 01015A94 | 86E1 | XCHG CL,AH | |
| 01015A96 | EB 01 | JMP SHORT UnpackMe.01015A99 | |
| 01015A98 | 2E 0FC80FCA | RND EAX,C80FC80F | |
| 01015A9D | 0FCB | BSWAP EBX | |
| 01015A9F | 0FC0DE | XADD DH,BL | |
| 01015AA2 | C1E8 08 | SHL EAX,08 | |
| 01015AA5 | 0FC8E9 | 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 הריבסר. 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?



```

01007300 6A 70          PUSH 70
01007301 68 98130001   PUSH UnpackMe.01001898
01007302 E8 BF010000   CALL UnpackMe.01007568
01007303 33DB         XOR EBX,EBX
01007304 53          PUSH EBX
01007305 8B3D CC100001  MOV EDI, DWORD PTR DS:[10010CC]
01007306 FFD7        CALL EDI
01007307 66+8138 4D5A  CMP WORD PTR DS:[EAX],5A4D
01007308 75 1F        JNE SHORT UnpackMe.0100730A
01007309 8B48 3C      MOV ECX, DWORD PTR DS:[EAX+3C]
0100730A 03C8        ADD ECX,ECX
0100730B 8139 50450000  CMP DWORD PTR DS:[ECX],4550
0100730C 74 12        JBE SHORT UnpackMe.0100730A
0100730D 0FB741 18    MOVZX EAX, WORD PTR DS:[ECX+18]
0100730E 3D 0E010000  CMP EAX,10E
0100730F 74 1F        JBE SHORT UnpackMe.010073F2
01007310 3D 0E020000  CMP EAX,20E
01007311 74 0E        JBE SHORT UnpackMe.0100730F
01007312 89D E4      MOV DWORD PTR SS:[EBP-4],EBX
01007313 EB 27        JMP SHORT UnpackMe.01007406
01007314 8B89 84000000  CMP DWORD PTR DS:[ECX+84],0E
01007315 76 F2        JBE SHORT UnpackMe.010073DA
01007316 33C0        XOR EAX, EAX
01007317 89D F8000000  CMP DWORD PTR DS:[ECX+F8],EBX
01007318 EB 0E        JMP SHORT UnpackMe.01007406
01007319 8B79 74 0E   CMP DWORD PTR DS:[ECX+74],0E
0100731A 76 E2        JBE SHORT UnpackMe.010073DA
0100731B 33C0        XOR EAX, EAX
0100731C 89D E8000000  CMP DWORD PTR DS:[ECX+E8],EBX
0100731D 9F95C0     SETNE AL
0100731E 8945 E4     MOV DWORD PTR SS:[EBP-1C],EAX
0100731F 89D FC     MOV DWORD PTR SS:[EBP-4],EBX
01007320 6A 02      PUSH 2
01007321 FF15 38130001  CALL DWORD PTR DS:[1001388]
01007322 59         POP ECX
01007323 8B0D 9CAB0001  MOV DWORD PTR DS:[100AB9C],FFFFFFFF
01007324 8B0D 80AB0001  MOV DWORD PTR DS:[100AB80],FFFFFFFF
01007325 FF15 34130001  CALL DWORD PTR DS:[1001384]
01007326 8B0D B39A0001  MOV ECX, DWORD PTR DS:[1009AB8]
01007327 8908        MOV DWORD PTR DS:[EAX],ECX
01007328 FF15 30130001  CALL DWORD PTR DS:[1001380]
01007329 8B0D B49A0001  MOV ECX, DWORD PTR DS:[1009AB4]
0100732A 8908        MOV DWORD PTR DS:[EAX],ECX
0100732B 81 2C130001  MOV EAX, DWORD PTR DS:[100132C]
0100732C 8B08        MOV EAX, DWORD PTR DS:[EAX]

```

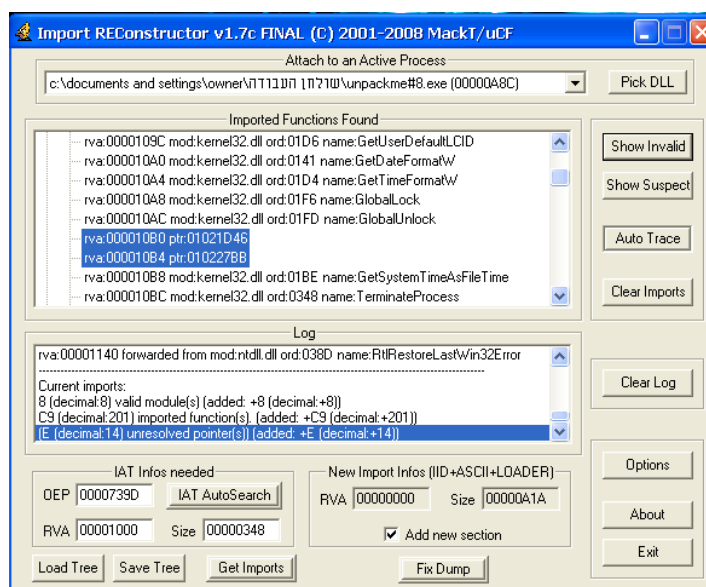
אנו רואים כי זו נקודת הכניסה המקורית של התוכנה! איך אנו יודעים זאת? אם פעם פתחתם קובץ שקומפל ב-Microsoft Visual C++ בדיבאגר, ראיתם בדיוק את נקודת הכניסה הזו. (ניתן להבחין בכך לפי הקריאות ל-msvcrt). כאמור, נקודה זו היא ה-Original Entry Point (OEP).

נפתח את LordPE, ונעשה לתהליך Dump, משום שהקוד האמיתי של התוכנה כבר לא ארוז. (קס) 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-IOEP!) 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 ☹

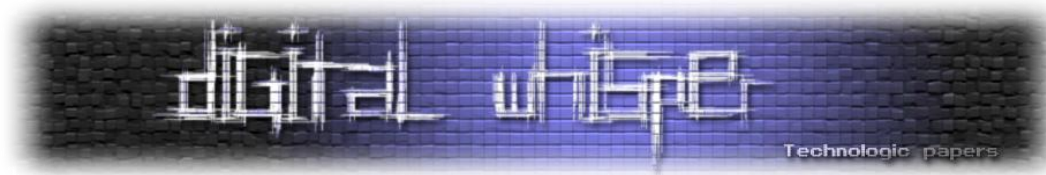


נצטרך לתקן את הערכים המושחתים.

כמו שאנחנו רואים ב-ImpRec, ה-IAT שוכן ב-0x00001000 RVA. 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.

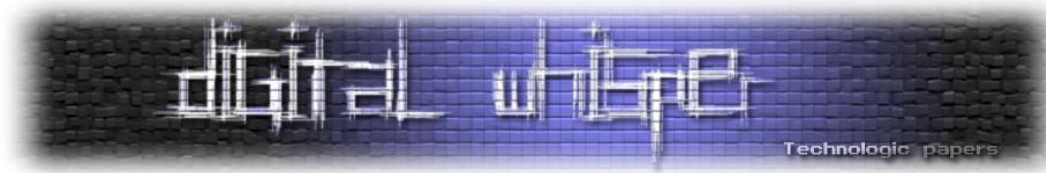
```
0101CEC8 JE SHORT UnpackMe.0101CF0F
```

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
```

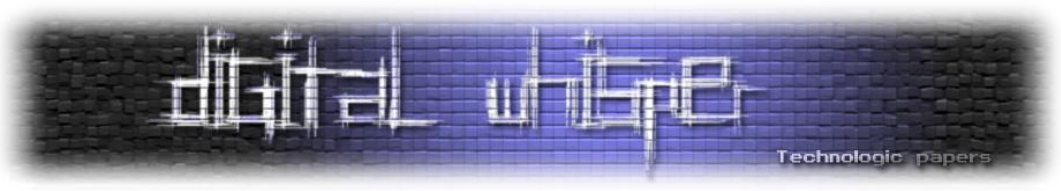
כאן מתבצעת ההחלפה

```
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
0101CEA8 E8 822B0000 CALL UnpackMe.0101FA32
0101CEB0 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
0101CEB3 3B08 CMP DWORD PTR DS:[EAX],8
0101CEB5 51 PUSH ECX
0101CEB6 8B0D 34D00201 MOV ECX,DWORD PTR DS:[102D0341]
0101CEB8 E8 8B5C0000 CALL UnpackMe.01022B4C
0101CEC1 8945 F8 MOV DWORD PTR SS:[EBP-8],EAX
0101CEC4 3370 F8 00 CMP DWORD PTR SS:[EBP-8],0
0101CEC8 EB 45 JMP SHORT UnpackMe.0101CF0F
0101CECA 8D55 F0 LEA EDX,DWORD PTR SS:[EBP-10]
0101CECC 52 PUSH EDX
0101CECE 6A 04 PUSH 4
0101CED0 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
0101CED5 50 PUSH EAX
0101CED6 FF15 2C870201 CALL DWORD PTR DS:[102872C]
0101CED7 55C0 TEST EAX,EAX
0101CED8 75 0A JNZ SHORT UnpackMe.0101CEEA
0101CEEA B9 0B0000EF MOV ECX,EF00000B
0101CEEC E8 482B0000 CALL UnpackMe.0101FA32
0101CEED 8B4D 08 MOV ECX,DWORD PTR SS:[EBP+8]
0101CEE0 8B55 F8 MOV EDX,DWORD PTR SS:[EBP-8]
0101CEE3 3B08 CMP DWORD PTR DS:[EDX],EAX
0101CEE5 8901 MOV DWORD PTR DS:[ECX],EAX
0101CEE7 8D4D F4 LEA ECX,DWORD PTR SS:[EBP-C]
0101EEF2 51 PUSH ECX
0101EEF4 8B55 F0 MOV EDX,DWORD PTR SS:[EBP-10]
0101EEF6 52 PUSH EDX
0101EEF8 6A 04 PUSH 4
0101EEFC 8B45 08 MOV EAX,DWORD PTR SS:[EBP+8]
0101EEFE 50 PUSH EAX
0101EF01 FF15 2C870201 CALL DWORD PTR DS:[102872C]
0101EF03 7745 FC 01000001 MOV DWORD PTR SS:[EBP-4],1
0101EF05 8B45 FC MOV EAX,DWORD PTR SS:[EBP-4]
0101EF07 8B55 FC MOV ESP,EBP
0101EF09 5D POP EBP
0101EF0B C3 RETN

```

לקפיצה מסוג זה קוראים Magic Jump והיא נפוצה בקרב Packers שונים. נרץ 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>