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 וירוסים ו-Malwares משתמשים ב-Packers כדי למנוע מחוקרי וירוסים לחקור את הווירוס ולמצוא דרך להשמיד אותו/למצוא את מפעילו. ללא אריזה כל חוקר וירוס 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.

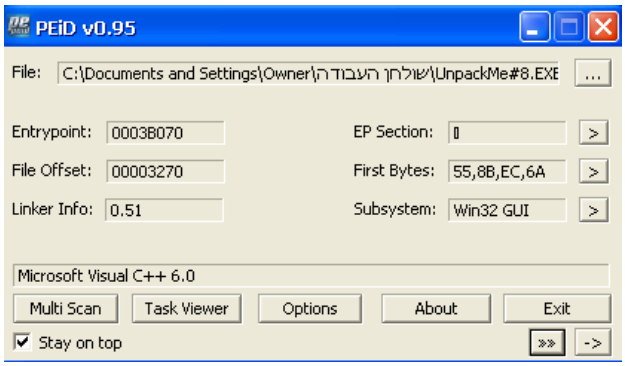
כל זה נעשה על ידי ה-IAT, ה-IAT הוא טבלה של מצביעים לפונקציות המתמלא ע"י ה-Windows Loader בעת טעינת הקובץ והספריות המקושרות אליו לזיכרון, במקום לשנות כל קריאה לפונקציה שתקרא לכתובת המתאימה, פשוט נקרא לכתובת השמורה ב-IAT.

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.

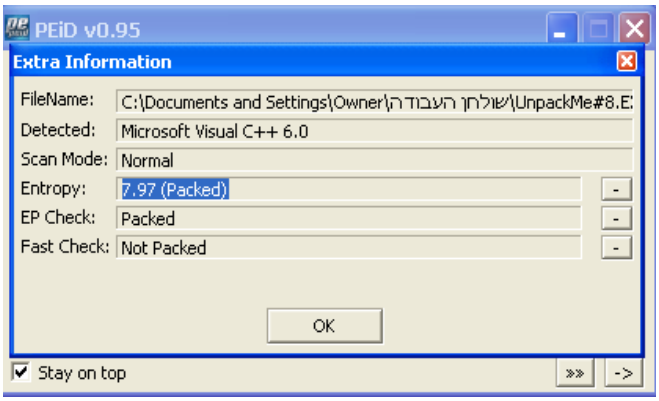
מתחילים

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!

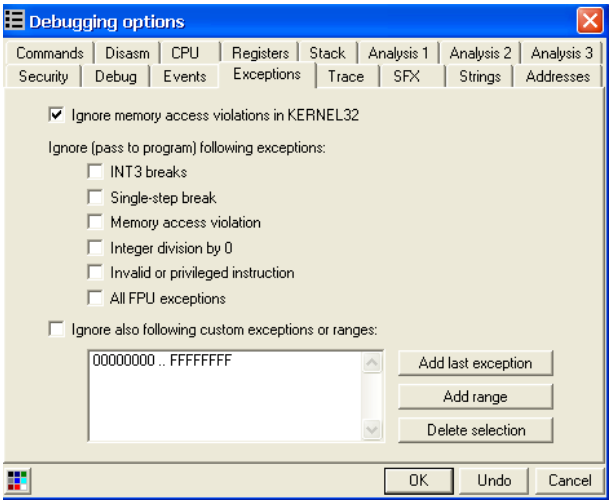


אחרי שראינו כי הקובץ ארוז, ננסה להריצו ב-Olly. בניסיון הראשון להריץ את התוכנה ב-Olly, התוכנה מגיע לחריגה שבה אינה מסוגלת לטפל, וקורסת.

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.

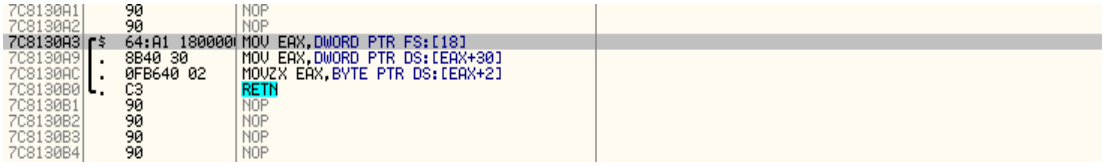
מדוע קריאה זו חשודה? משום שאנחנו לא יכולים לדעת מראש איזה פונקציה תיקרא פה. התוכן של EAX נקבע רק בזמן ריצה. אנחנו לא יכולים לדעת מה היה התוכן של EAX ברגע שנקרא לפני ריצת התוכנה. טריק זה הינו דרך של ה-Packer להסוות קריאות שהוא לא רוצה שנראה.

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

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.

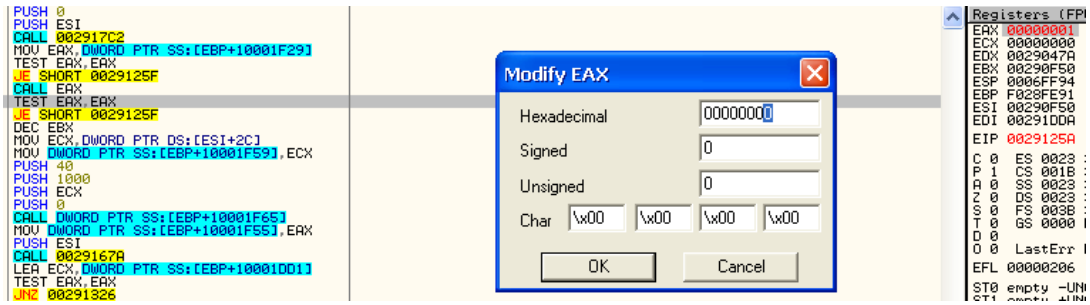
הסבר קצר על IsDebuggerPresent:

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.



- 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

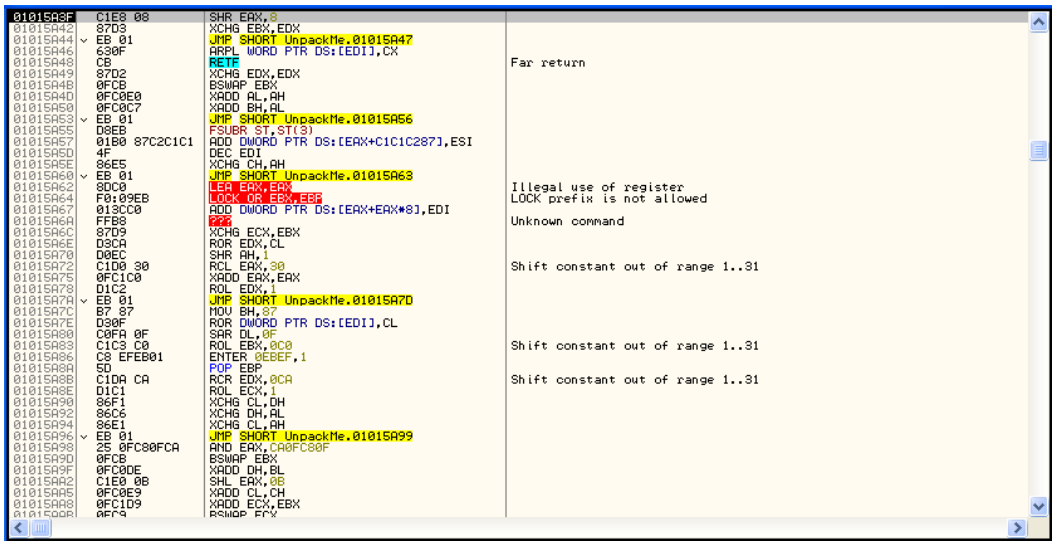
לפני שנמשיך בניתוח התוכנה הנתונה נציג טכניקה נפוצה נוספת של Anti-Debugging, שהיא הכנסת ההוראה INT3 באמצע קוד רגיל.

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:

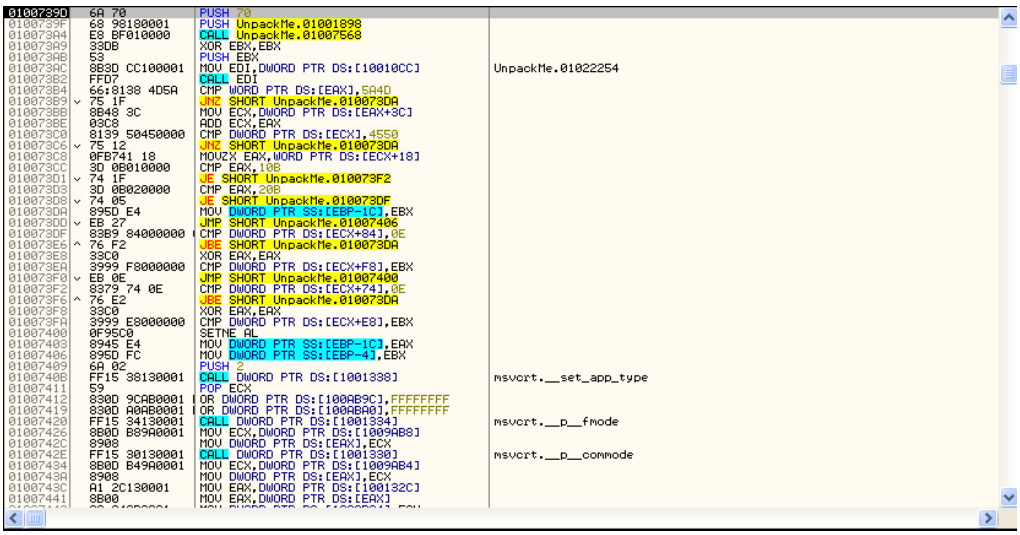


קוד זה נראה מאוד מוזר, זהו Obfuscation Code (קוד הטעייה) – קוד שנועד לבלבל את הריברסר. לרוב קטע קוד זה לא משפיע כלל על התוכנה ואין צורך בו.

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?



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.

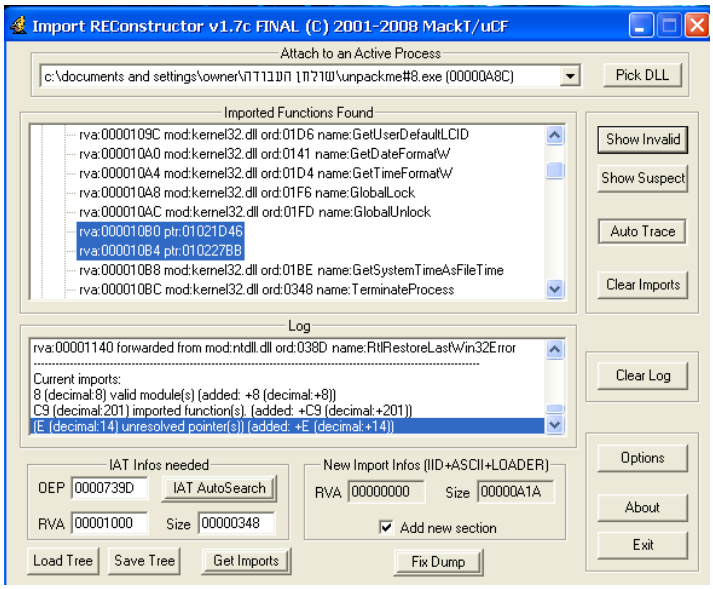
(Dump הוא העתקה של זיכרון התהליך הרץ ברגע נתון, לתוך קובץ חדש, למשל EXE).

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-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
```

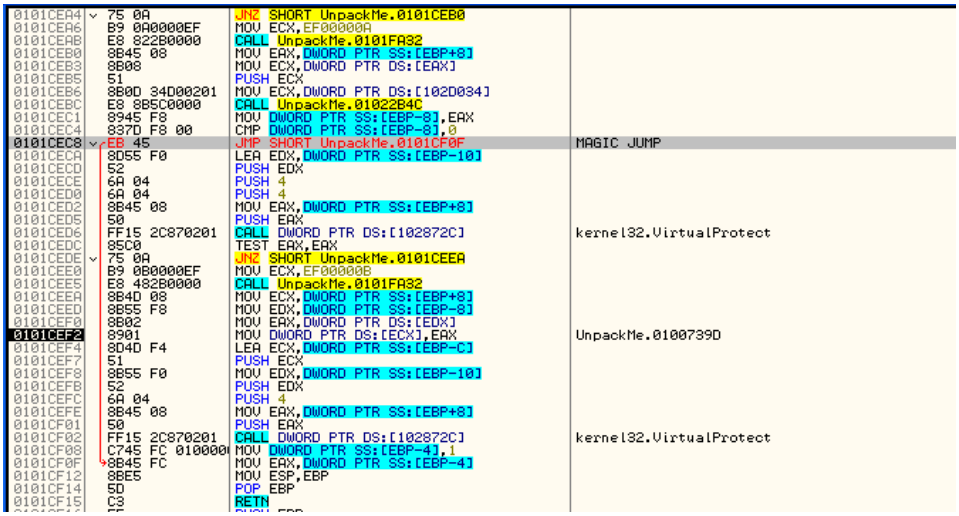
כאן מתבצעת ההחלפה

```
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.



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