עבודת בית 3

מגישים:

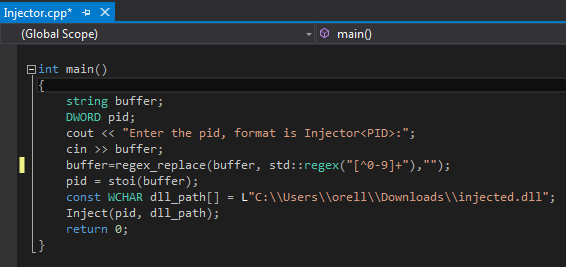
אוראל לביא , 207632118

סניה קלמה , 321969941

תרגיל 1

Injector.exe

נתחיל עם הצגת פונק' הmain בקובץ injector.cpp אשר מבצעת את הקלט והעברת הפרמטרים הנדרשים (pid ומיקום הdll בfile system)



הפונק' Inject מקבלת pid ואת הpath של הdll,



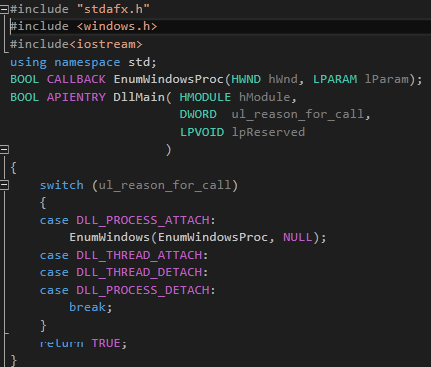
כאשר מבוצע GetProcAddress מkernel32 כדי לקבל את הכתובת של LoadLibraryW מהexport table שלו. זה מבוסס על העובדה שkernel32 ממופה לכל process , וכן ממופה אף לאותם addresses. נשתמש ב LoadLibraryW כפונק' הThreadProc מכיוון שיש לה חתימה דומה. אותה כתובת שנשמרת בLoadLibAddr תיהיה אותה הכתובת גם בתהליך השני , ועל סמך זה מבוסס הinjection. את ההזרקה לתהליך השני בעל המזהה pId נבצע על סמך השילוש VirtualAllocEx, WriteProcessMemory (כדי לבצע אלוקציה בזכרון אצל התהליך בשביל להחדיר את הפרמטר שמציין את שם הdll , הפרמטר בשביל LoadLibraryW שירוץ אצלו) ובנוסף שימוש ב- CreateRemoteThread.

על סמך ערך החזרה של הפונק' הmain של הthread שנוצר בתהליך השני נדע אם הצליחה הפעולה של החדרת הdll (ערך החזרה שונה מ-0).

לאחר מכן נשחרר את המשאבים עם VirtualFreeEx וכו'.

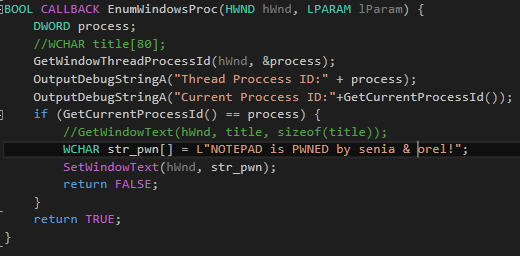
Injected.dll

נתחיל עם dllmain



נשים לב שכאשר הdll נעשה attached לאותו process שביצע לו load , הפעלנו את EnumWindows שמבצעת אנומרציה על כל החלונות , כאשר לה נספק את פונק' הcallback שלנו שבה נבצע את התהליך הרצוי בעצם.

נסתכל על פונק' הcallback EnumWindowsProc



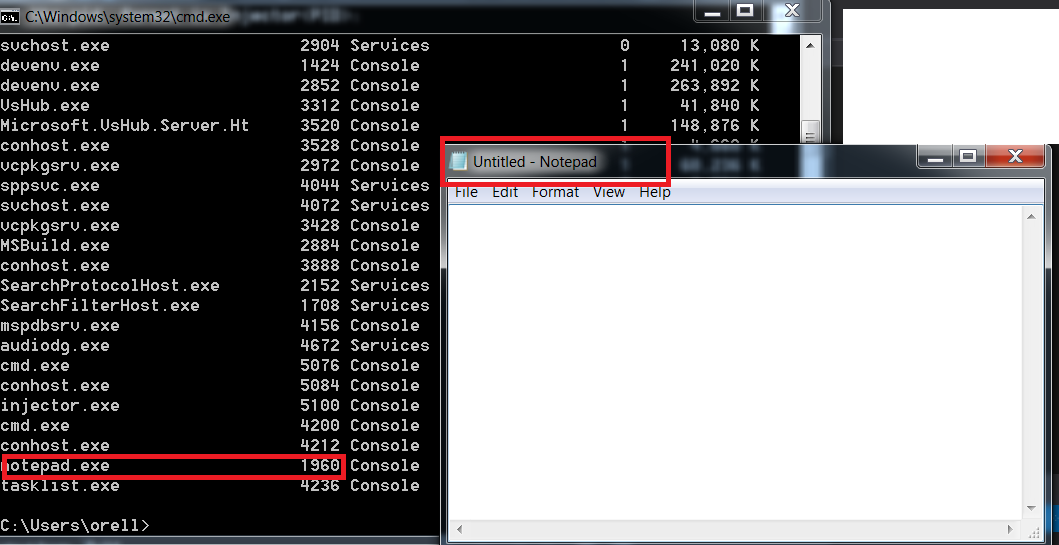
נשתמש ב GetWindowThreadProcessId כדי לקבל את הpid של הprocess שיצר את החלון ( הפרמטר process מחזיק את הprocess id).

נשתמש ב GetCurrentProcessId כדי לקבל את pid של הprocess הנוכחי, ואם הם שווים, אנו בחלון הרצוי (יצרנו חלון אחד של הnotepad) ונבצע את הדריסה של הtext בtitlebar ע"י שימוש בפונקציה SetWindowText.

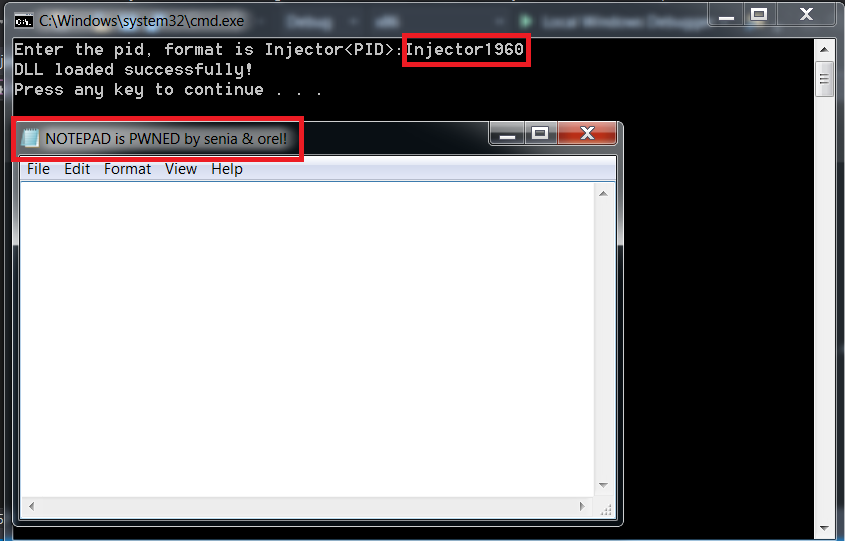
נראה דוגמא להרצת התוכנית וביצוע השינוי ל titlebar:

לפני

נבצע שימוש בפקודה tasklist כדי לראות את הprocesses הפתוחים



אחרי

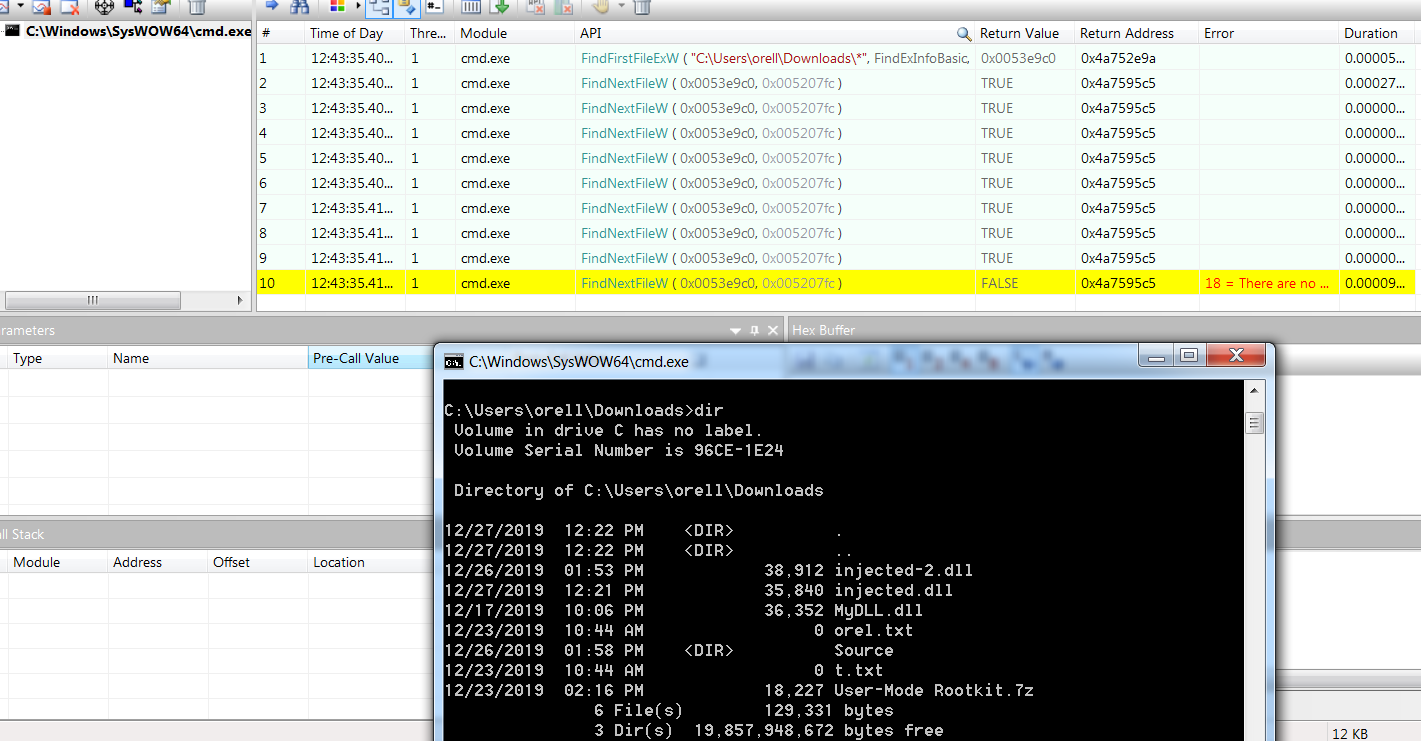


תרגיל 2

חלק 1

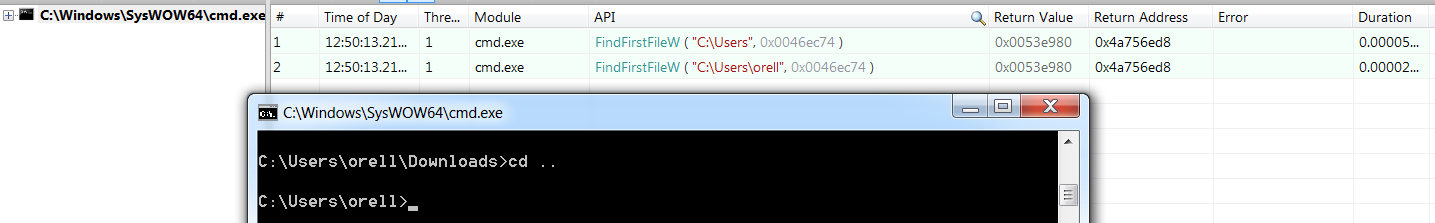
בתרגיל זה החלטנו לבצע מעין variant של user level rootkit לcmd , כך שנסתיר קובץ שכן יופיע בfilesystem , אבל בפקודה dir הcmdline לא יראה אותו.

נשתמש בapi monitor על מנת לראות באילו api calls משתמש הcmd בפקודה dir על מנת להציג תוכן



נראה כי על מנת לחפש קבצים ע"י פקודה dir הוא משתמש ב win api class הבאים: FindFirstFileExW ו FindNextFileW. להן נבצע IAT Hook בתהליך שאליו יוזרק הdll.

נשים לב כי כאשר נבצע שינוי תיקיות ע"י פקודה cd

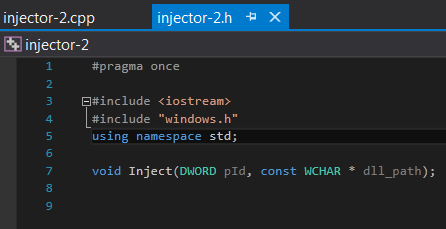


יש שימוש בFindFirstFileW , ועל כן החלטתי לבצע IAT Hook גם לה.

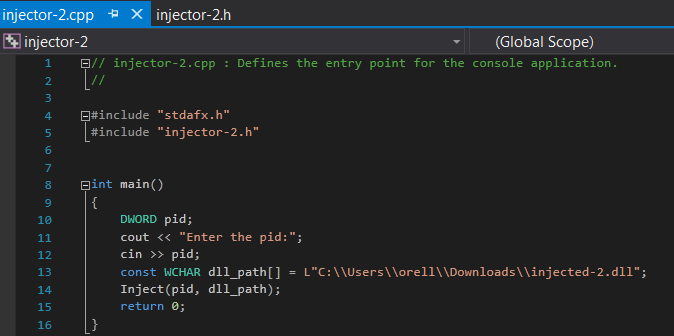
Injector-2.exe

בדומה לתרגיל 1 , נבצע את התהליך של הזרקת הdll באופן דומה

Injector-2.h

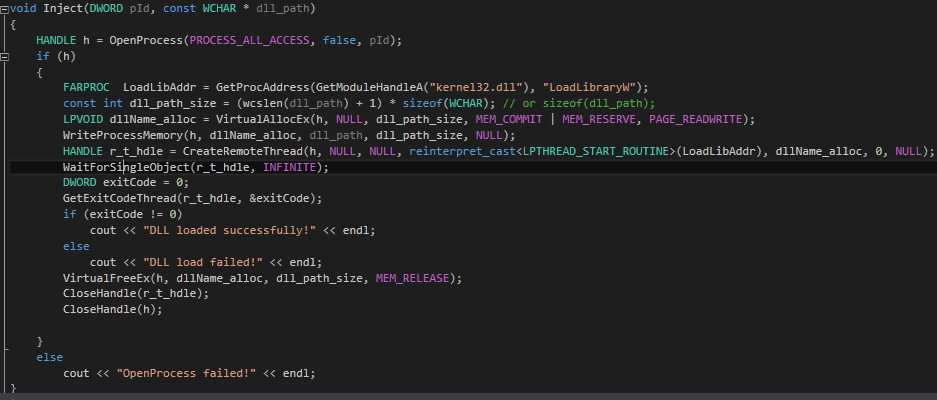


Injector-2.cpp



פונק' הmain מבצעת את הקלט והעברת הפרמטרים הנדרשים (pid ומיקום הdll בfile system)

פונק' Inject מבצעת גם כן באופה דומה למה שתואר לעיל בתרגיל 1



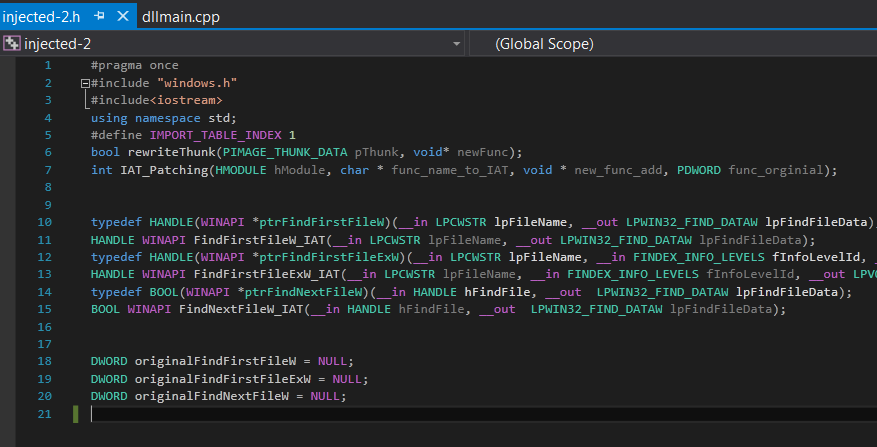
כאשר מבוצע GetProcAddress מkernel32 כדי לקבל את הכתובת של LoadLibraryW מהexport table שלו. זה מבוסס על העובדה שkernel32 ממופה לכל process , וכן ממופה אף לאותם addresses. נשתמש ב LoadLibraryW כפונק' הThreadProc מכיוון שיש לה חתימה דומה. אותה כתובת שנשמרת בLoadLibAddr תיהיה אותה הכתובת גם בתהליך השני , ועל סמך זה מבוסס הinjection. את ההזרקה לתהליך השני בעל המזהה pId נבצע על סמך השילוש VirtualAllocEx, WriteProcessMemory (כדי לבצע אלוקציה בזכרון אצל התהליך בשביל להחדיר את הפרמטר שמציין את שם הdll , הפרמטר בשביל LoadLibraryW שירוץ אצלו) ובנוסף שימוש ב- CreateRemoteThread.

על סמך ערך החזרה של הפונק' הmain של הthread שנוצר בתהליך השני נדע אם הצליחה הפעולה של החדרת הdll (ערך החזרה שונה מ-0).

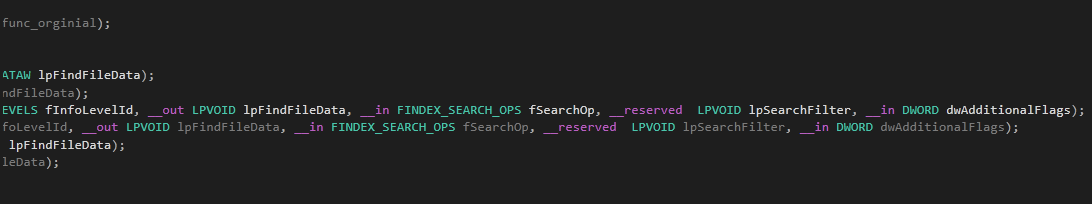
לאחר מכן נשחרר את המשאבים עם VirtualFreeEx וכו'.

Injected.dll

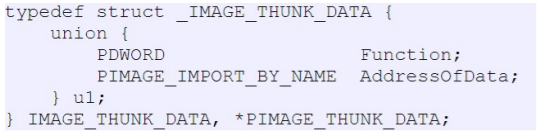
Injected-2.h file



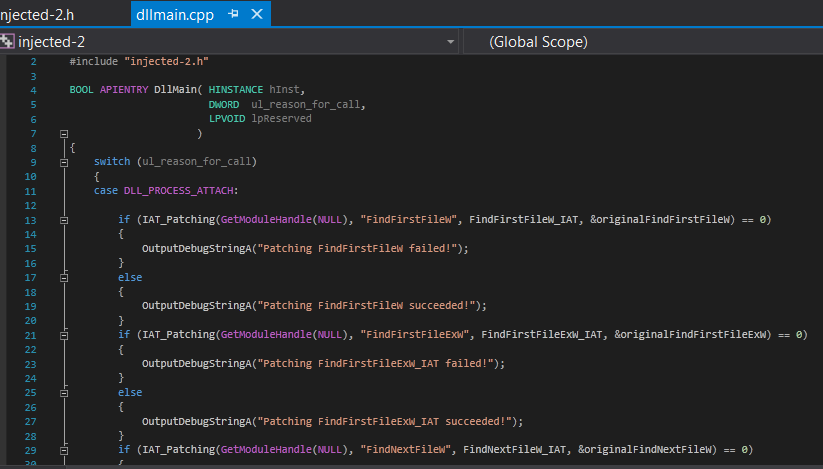
ההמשך החתוך מצד ימין



לכל פונק' שנרצה לבצע IAT Hook נשמור את הכתובת המקורית בIAT ע"י ptr מהסוג וחתימה זהים. את כתובת הפונק' נשמור במשתנים origingalFind… מסוג DWORD (כך גם מוגדרת כתובת הפונק' בתוך הImport table בstruct Image\_thunk\_data (PDWORD כפוינטר לערך הנ"ל בשם Function , כאשר ניקח את תוכנו).

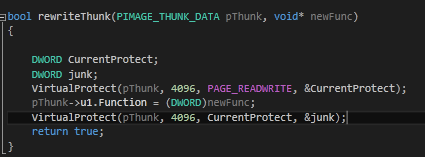


dllmain.cpp file



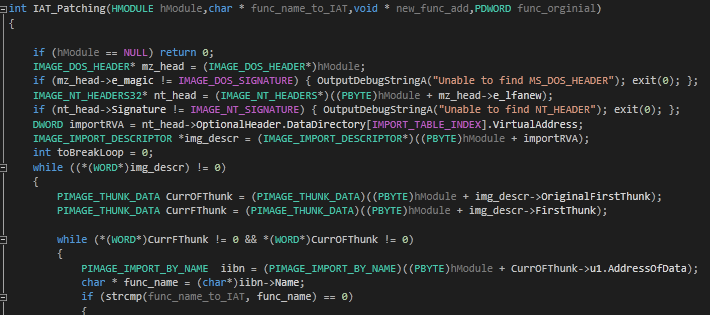
נבצע קריאה לIAT\_Patching עבור כל פונק' , כאשר נעביר את הHModule של הprocess שאליו מזריקים ( הprocess והdll משתפים בניהם את הmodule) , את שם הפונק' שנרצה לבצע לה patching, את הפונק' החדשה ואת הכתובת של המשתנה לשמירת הכתובת המקורית.

הפונק' rewriteThunk



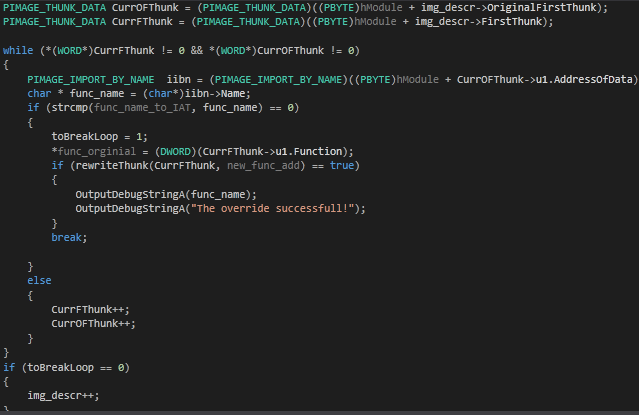
מבצעת את החלפת הכתובת בטבלת הimport , ע"י שימוש בVirtualProtect כדי לשנות את הרשאות הדף ,בשביל לכלול בהרשאות כתיבה, ולהחזיר לאחר השינוי את ההרשאות למצב המקורי.

הפונק' IAT\_Patching



נבצע traverse על הhmodule שמייצג את הPE של אותו הprocess המוזרק, כאשר על מנת להגיע לimport table (IMAGE\_IMPORT\_DESCRIPTOR \* ) נשתמש בכתובת בסיס ונוסיף אליה את הRelative Virtual Address(RVA) שבoption header שבNT Headers.

לאחר מכן , נעבור בלולאה על כלל מבני ה - IMAGE\_IMPORT\_DESCRIPTOR , כאשר כל אחד מהם מתאר dll , ונחפש את הפונקציה הרצויה לpatch.

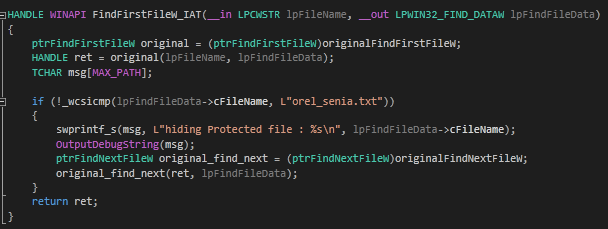


כל פעם נבחר את הThunk על מנת לגשת לפונק' עצמן שבוצע להם import.

נשתמש בOriginalFirstThunk כדי לגשת למבנים שמייצגים את הפונק' (IMAGE\_IMPORT\_BY\_NAME) , כאשר אותו OriginalFirstThunk מייצג את הINT לאחר שבוצע load לתוכנית ע"י הloader.

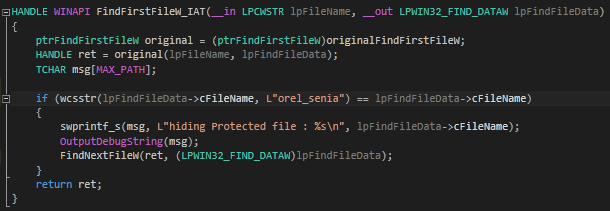
לאחר שהפונק' נמצאה (השם שלה זה השם של הפונק' שהעברנו לפונק' הiat\_patch כקלט), נשמור את הכתובת דרך הIAT (FirstThunk) בתוכנו של הפוינטר func\_original שמייצג את הכתובת למשתנה originalFind… שתואר לעיל. לאחר מכן, נעביר את הכתובת ל rewriteThunk על מנת לבצע את הpatching.

הפונק' FindFirstFileW\_IAT

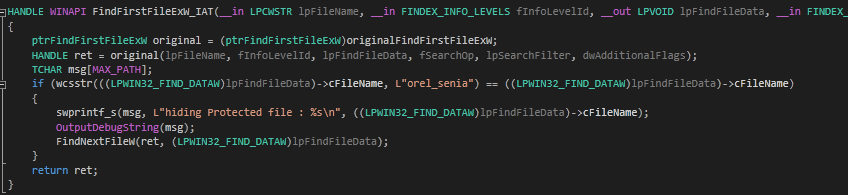


מייצגת את הפונק' IAT\_PATCH ל FindFirstFileW.

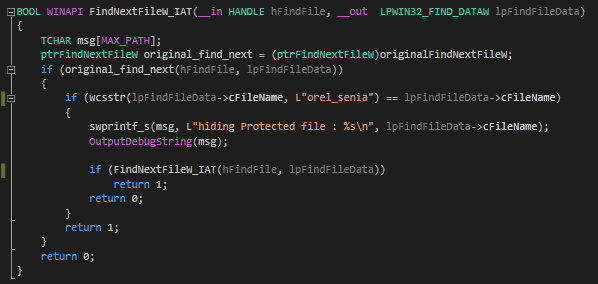
נבצע שימוש בoriginal FindFirstFileW על מנת לבצע את החיפוש עבורנו, וכאשר הפונק' תמצא את הקובץ עם שם בעל התחילית "orel\_senia" נסתיר אותו מהפלט של dir בcmd.



אותו כנ"ל לגבי FindFirstFileExW\_IAT



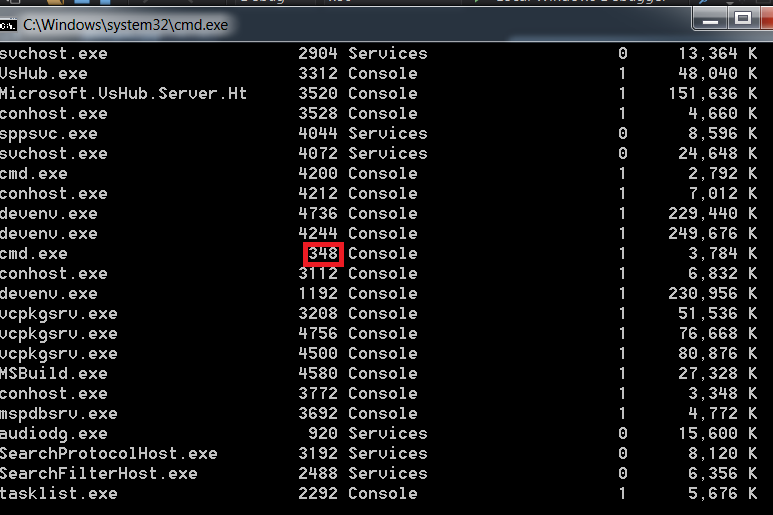
אותו כנ"ל לגבי FindNextFileW\_IAT

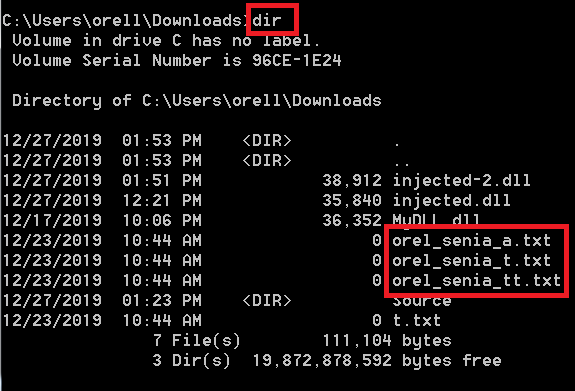


נראה דוגמא להרצת התוכנית וביצוע השינוי לפלט של dir:

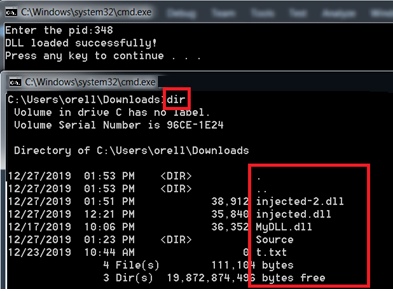
לפני

נבצע שימוש בפקודה tasklist כדי לראות את הprocesses הפתוחים

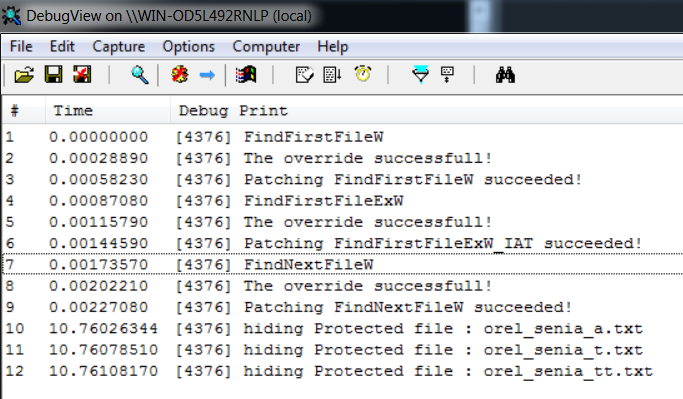




אחרי



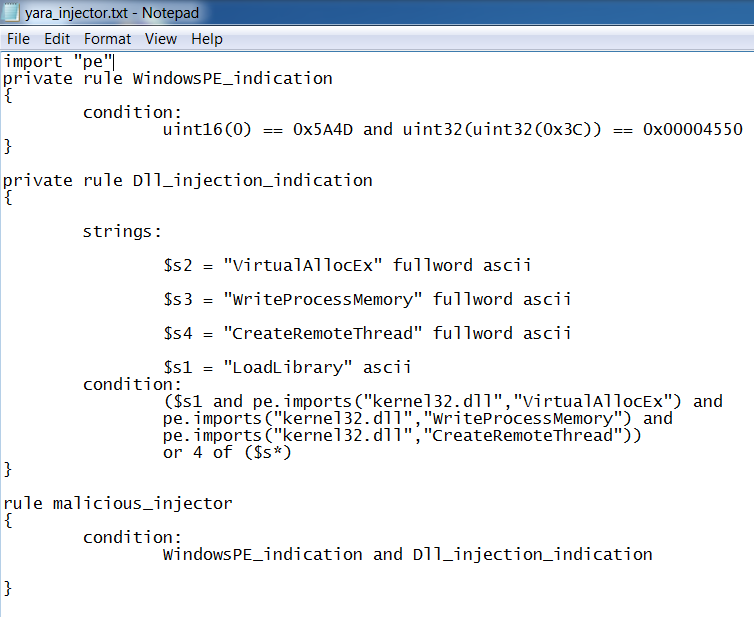
רואים כי אכן הקבצים שהתחילו בorel\_senia נעלמו. ניתן לראות זאת גם בDebugView:



חלק 2

1. Injector-2.exe

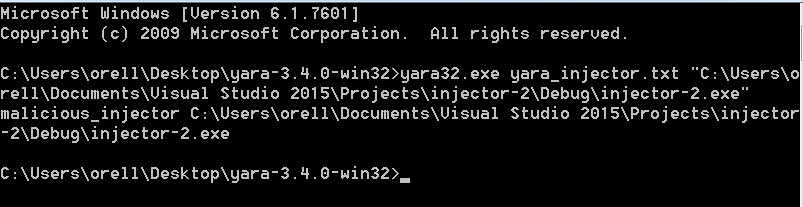
נראה את הyara rules



תפקידו של הrule WindowsPE\_indication לציין האם מדובר בexe

תפקידו של הrule Dll\_injection\_indication לציין האם מדובר בdll injection דרך VirtualAllocEx+WriteProcessMemory+CreateRemoteThread ושימוש בLoadLibrary.

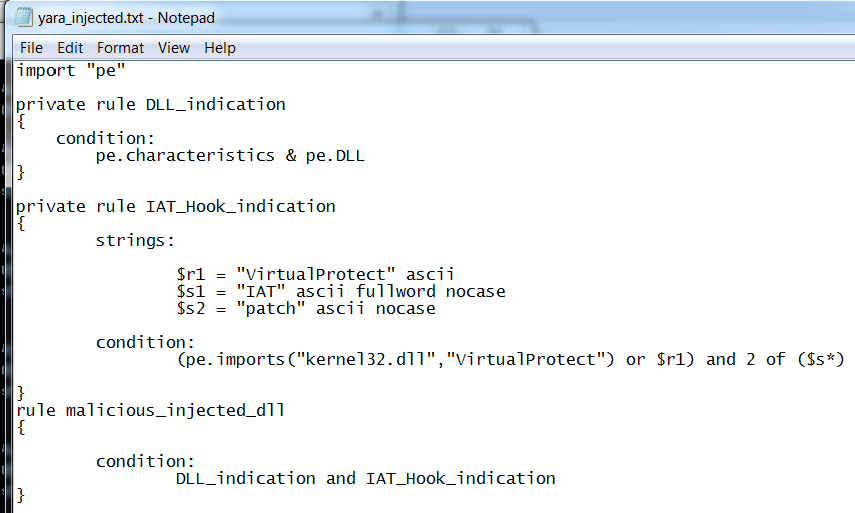
בין אם מדובר בstrings , בין אם מבוצע import של אותם פונק' וגם יש שימוש בקוד בstring של LoadLibrary (גם אם יש סיומת עבור גרסאות שונות ,כמו A או , W או בלי – בשל כך ללא תג הfullword)



נראה כי אכן הexe מתגלה ע"י אותו rule malicious\_injector שמפעיל את שתי הrules הנ"ל.

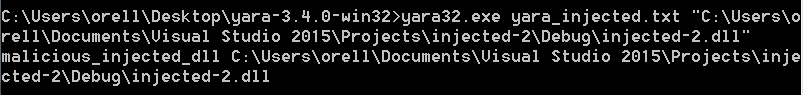
Injected-2.dll

נראה את הyara rules



תפקידו של הrule DLL\_indication לציין האם מדובר בקובץ dll

תפקידו של הrule Dll\_injection\_indication לציין האם מדובר IAT Patch דרך VirtualProject , בין אם string או כimport, ואזכור לstrings המתארים IAT וpatch.



נראה כי אכן הdll מתגלה ע"י אותו rule malicious\_injected\_dll שמפעיל את שתי הrules הנ"ל.

1. ניתן לעקוף את הrules באופן הבא

Injector-2.exe

את הrule הנ"ל ניתן לעקוף בקלות למשל ע"י שיטות injection אחרות כמו direct dll injection או העלאה דרך הregistry (appInit/Cert) וכו'.

כמו כן, ניתן לבצע זאת ע"י packing של הstring LoadLibrary , וביצוע unpack שלו בruntime (אפשר xor עם ערך אחר , ואז אח"כ שוב xor כדי לבטל וכו')

Injected-2.dll

את הrule הנ"ל ניתן לעקוף בקלות ע"י אי אזכור של IAT או של Patching כלל כstring.

וכן, ניתן לעקוף ע"י שימוש בפונק' אחרת במקום VirtualProtect, אולי פונק' אחרת מהwindows api כמו VirtualProtectEx או פונק' custom made לשינוי הרשאות וכו'