

четвъртият - Windows Kernel Manual

שי גילת

הקדמה

בஹמשך של סדרת המאמרים שלי על נושא Windows internals החלטתי להתרכז במאמר זהה בכלים, עצות ועקרונות שכורכים יותר בפיתוח קרנלי כללי. מי שחקור מוצרים קרנליים למערכות Windows יכול להבין ש-99 אחוז מהחולשות שפוגעות במליאוני מערכות בכל העולם הם חולשות לוגיות "על סטרואידים".

זה לא אומר שאין חולשות זיכרון כמו overflow-ים למיניהם, אבל גם הן מבוססות לרוב על הנחות שלא בהכרח נכונות למערכת ולתוכנות שלה, טעויות בהעברת פרמטרים לפעולות בסדר פעולות וכו'.

ניתן לראות דוגמא לכך במקרה של חברת CrowdStrike שגרם לкриיסת כמות עצומה של מערכות בבתי חולים, שדות תעופה, בנקים והרשימה רק ממשיכה. הזריםים של פתרון האבטחה של CrowdStrike בטוח ציפו שהחברה כזו גדולה עם אלפי עובדים וחווים של מיליארדי דולרים, אך למרות זאת המפתח שביצע שינוי לפתרון ההגנה ברגע האחרון ללא בדיקה טוביה ומעמידה שגרם לאחת מהחולשות הבסיסיות ביותר - .NULL dereference.

חולשה זו נגרמת ברגע שימושם בערך\ משתנה השווה ל-0 כתובות למקום בזיכרון שמננו מנסים, denial of service או שאליו מנסים לכתוב נתונים, חולשה זו יכולה לגרום למגאון בעיות כמו Elevation Of Privileges.

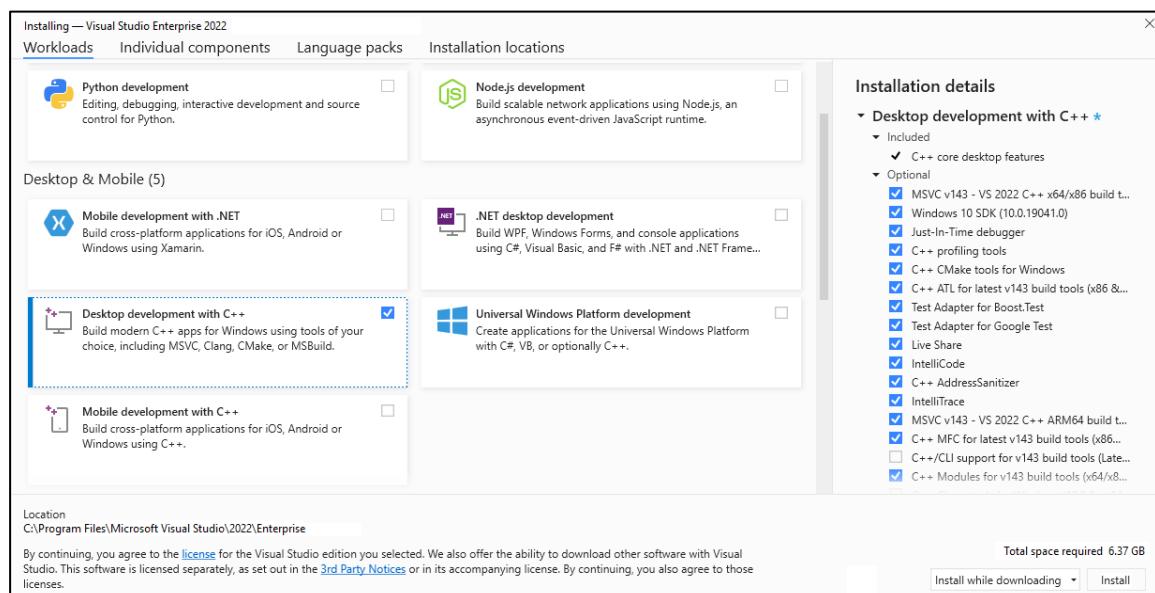
במאמר זה אכנס לנקודות מרכזיות בפעולות ה-Driver, קונספטים חשובים שיכולים לעזור לנו להבין יותר טוב את הסביבה שאנו נמצאים בה, כלים וצורת השימוש בהם כדי לפתח ולדיבג את ה-Driver שלנו ושימוש נכון בפעולות API נפוצות כדי למנוע התנהגות לא צפויה של התוכנה.

תנאים בסיסיים לפיתוח קרNELי במערכות Windows

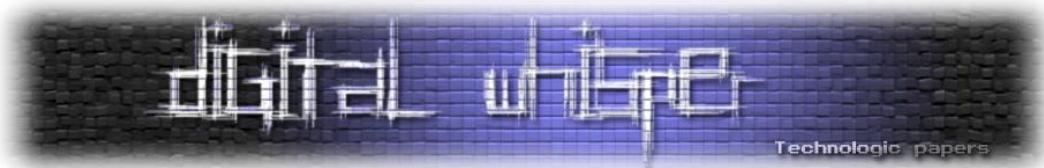
לפני שנתחילה לכתוב קוד קרNELי, נctrיך להכין את סביבת העבודה שלנו כמו שצרי, להתקין את הספריות ההכרחיות והגדרת כלים שיוכלו לעזור לנו בצורה משמעותית לאורך כל תהליך הפיתוח והדיבוג.

:Visual Studio

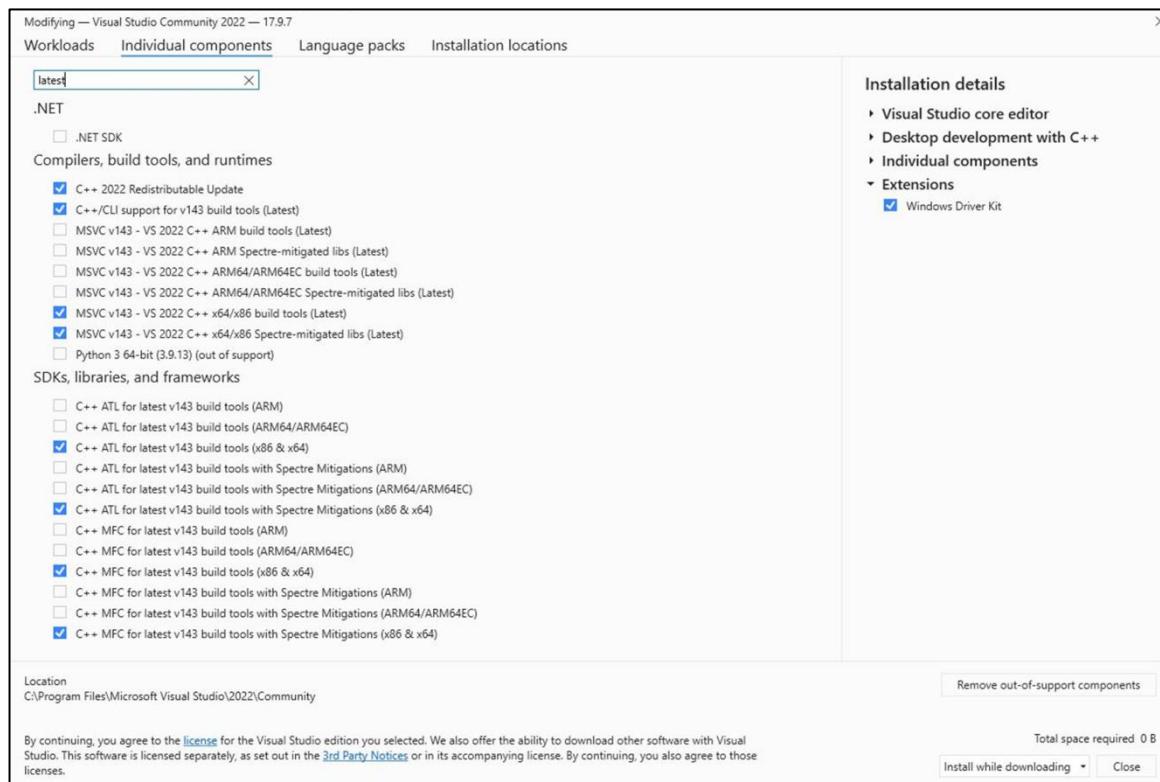
הפלטפורמה שבה נהוג להשתמש לפיתוח -Driver ים היא Visual Studio ושהה שבה נפתח את ה-
Visual Studio יכולה לשתנות לפי העדפה, אך בדרך כלל תהיה +C/C#. אך נודא להתקין את התוכנה **Desktop Development with C++**



עוד חשוב שחייב לשים לב אליו הוא התקנת כל ה-**individual components** שקשורים לפיתוח -Driver ים
למערכות Windows שלא מסומנות כבר על ידי ה-**workload** בתמונה.



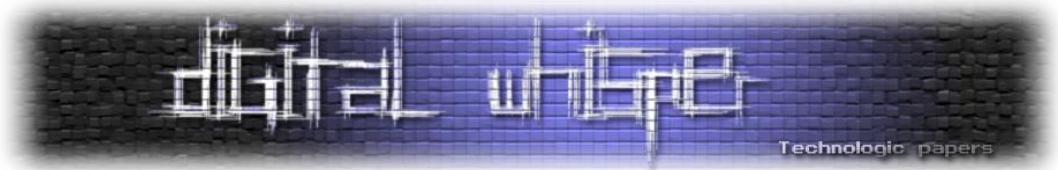
ניתן לראות את כל הרכיבים שצירף לסמן בתמונה הבאה:



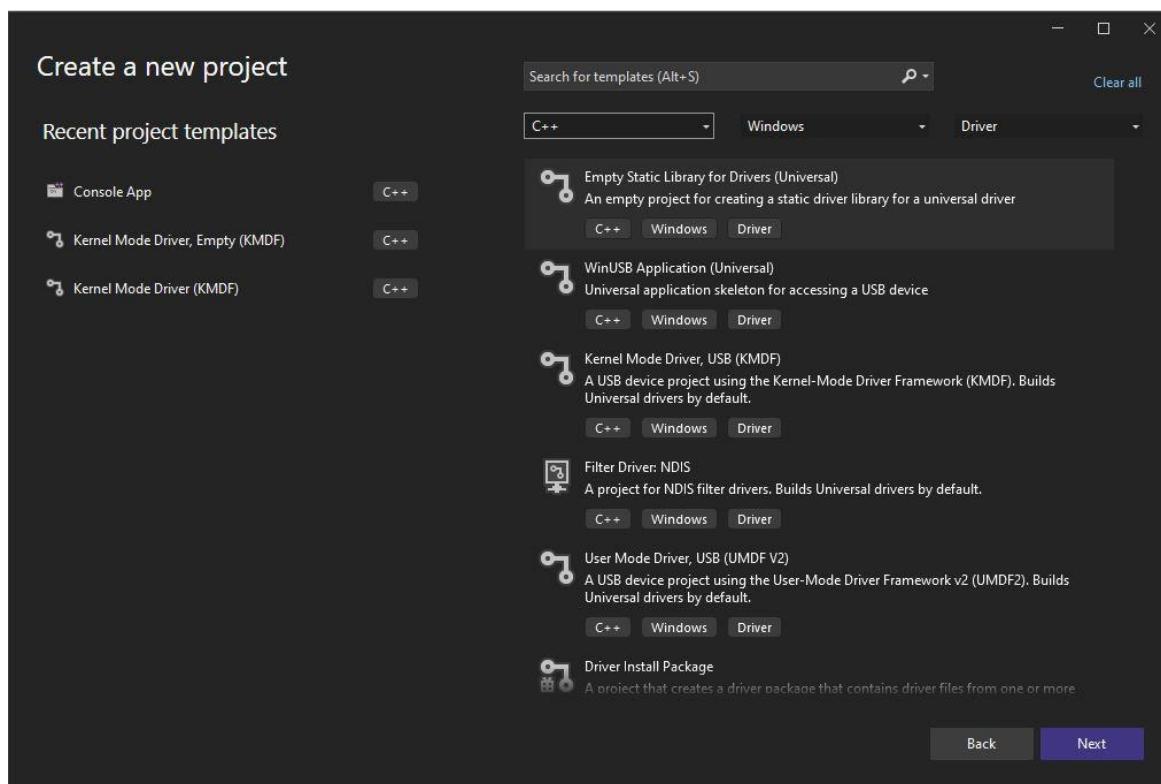
:SDK + WDK

ה-SDK (Software Development Kit) וה-WDK (Driver Development Kit) הם ארכיטקטורת שמייקروسופט מספקת לכל מפתח-ים היכולותtemplates שאיתם ניתן לפתח דרכו Visual Studio מספר סוגים של Driver-ים שונים למטרות שונות כמו USB Driver, file-system Driver ועוד.

ה-SDK כולל גם כלים נוספים כמו debuggers שיכולים להראות לנו הודעות מפורטות על פעילות המערכת ולהביא לנו שליטה מוחלטת על המערכת ותוכנות תייעוד של אירועים שונים המתרחשים בוגע לתחומים ספציפיים כמו ריצת תהליכים, חיבור USB ועוד.



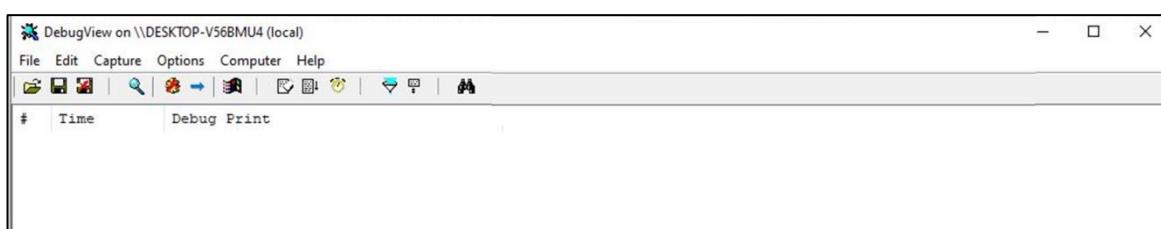
את הקישורים להתקנות הללו הוסף בסוף המאמר, וניתן לראות שההתקנה עבדה לפני הופעת סוג' פרויקטים חדשים ב-Visual Studio:



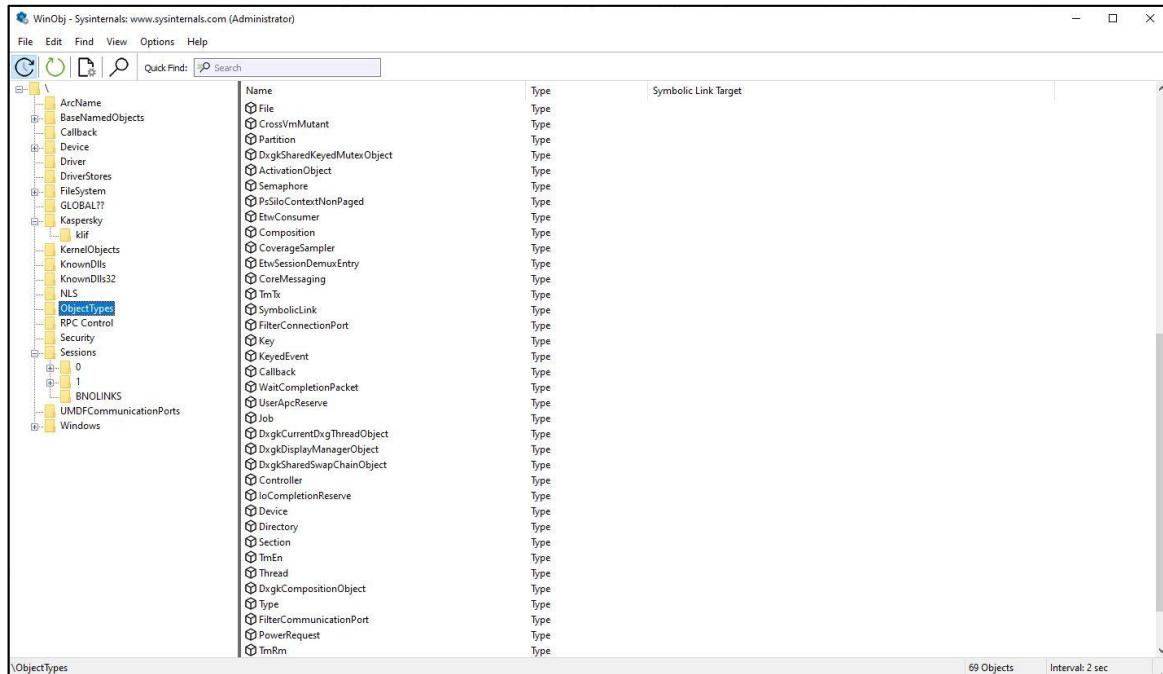
:(WinObj-DbgView SysInternals ובעיקר

למרות שהכלים הללו לא הכרחיים לפיתוח קרנלי במערכות Windows הם יכולים לעזור לראות שה-Driver שלנו פועל כמו שצריך. את ההתקנה ניתן לעשות ישיר מהאתר של מיקרוסופט ועם ההתקנה מגעים הרבה כלים שימושיים, כגון WinObj ו-DbgView.

DbgView הוא כלי שנותן לנו לראות את כל הפעולות Debugging שיוצאות מתוכנות הרצות על המערכת, בין היתר תוכנות מערכת פנימיות כמו ה-Driver שאנחנו רוצים לפתח. התוכנה מאוד פשוטה וモנתנת לנו אפשרות tracing להודעות ספציפיות לשוגים מסוימים של תוכנות (Win32, Kernel Programs וכו'), ועוד של תהליכי booting של המערכת ואפשר חיבור מרוחק למערכת אותה נאтра:



WinObj היא תוכנה שנותנת לנו אפשרות להסתכל על כל האובייקטים הגלובליים במערכת כמו symbolic links,Driver-ים רשומים במערכת ו-devices הקיימים במערכת שניתן לתקשר איתם. תוכנה זו יכולה לזהות מגוון גדול של אובייקטים במערכת ולתאר אותו בצורה הכי מפורטת שאפשר:



:Windbg

אין צורך להשתמש ב-driver debugger מסויים עבור dynamic debugging ל-Driver, אבלwindbg הוא חזק מאוד עם יכולות רבות שיכולים לעזור לנו בתהילך הפיתוח, משקל קל ונוח למשתמש והוא מותקן כבר למערכת חלק מה-SDK של מיקרוסופט. Windbg מאפשר לנו בדיקה דינמית של תוכנות נוספות כגון exe. files רגילים אבל כדי לבדוק את ה-Driver שלנו נצטרך לשנות את הגדרות המערכת שבה נבדק את ה-.kernel debuggingDriver כדי לאפשר.

אני חשב שהה ברור כבר שרכיב תוכנה מסוון וחזק כמו Driver עדיף לבדוק במערכת נפרדת שלא חשובה לנו, או בעדיפות הci גבוהה על מכונה וירטואלית. הרמת הסביבה ל-remote kernel debugging לא יהיה שונה בין מחשב רגיל, מכונת vmware, מכונת v-hyper ובודמה וכן לא נכנס לכל אחת מהאפשרויות בפני עצמה.

להרמת הסביבה יש מספר שלבים:

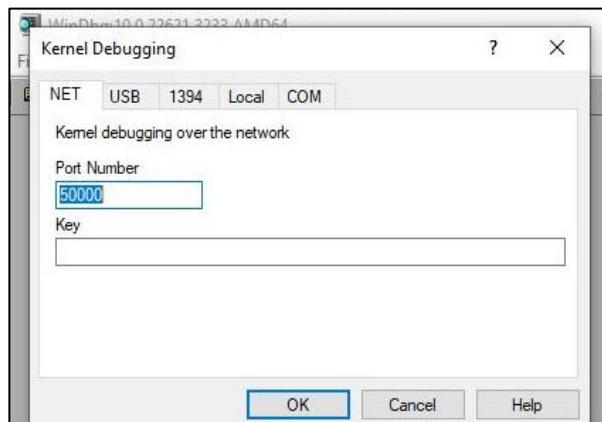
- (1) הדלקת האפשרות ל-kernel debugging: כדי לבצע את הפעולה זו נדרש לפתח חלון cmd עם הרשאות מנהל. בחלון הפקודות נרץ את הפקודה "bcdedit /DEBUG ON" שתאפשר לנו לדבג את המערכת עלייה נבדק את ה-Driver

(2) הגדרת הפרמטרים ל-remote kernel debugging ל-: CAN נctrkr לשנות את הגדרות המערכת עבור הצלחת תהליך debugging-.NET. את השינוי זהה נעשה עם הפקודה:

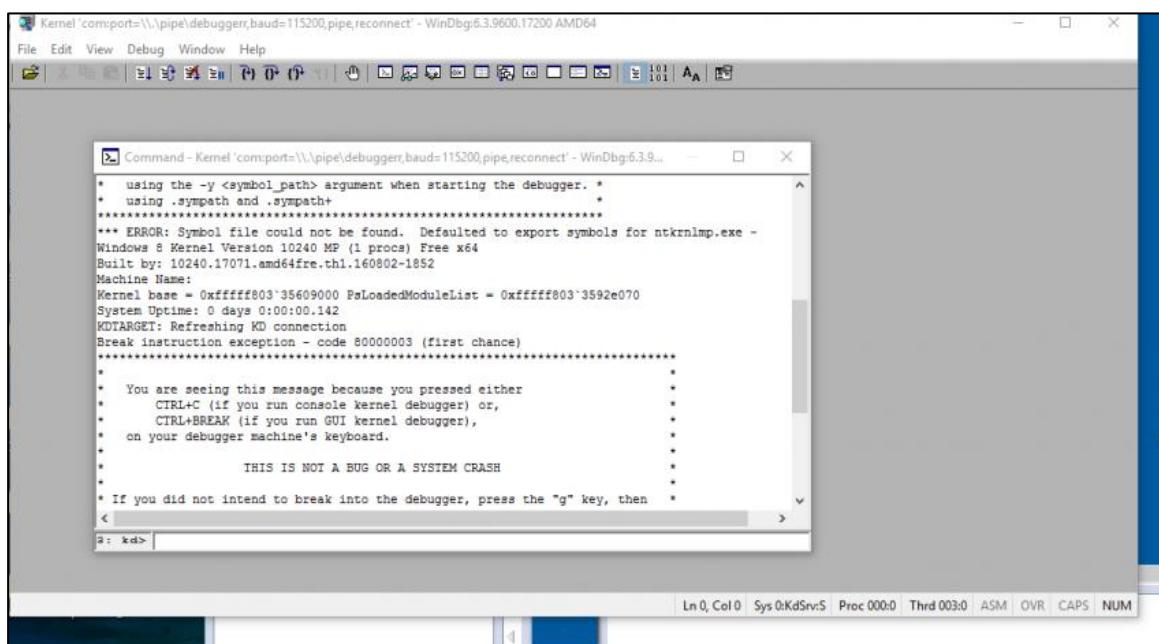
```
bcdedit /dbgsettings NET HOSTIP:{ip_address_of_host_machine} PORT:{debugging_port_on_target_machine} KEY:{debugging_key}
```

במקום הפרמטרים שモוקפים ב-{ } נכnis בהתאם את כתובת ה-IP של המערכת הבודקת שאיתה המערכת הנבדקת ("מערכת המטריה") יכולה לתקשר, מספר ה-port שדרכו נרצה שהמערכת הנבדקת תתקשר עם המערכת הבודקת וערוך מחוזתי לשחו שייה מפתח debugging- שלנו לפי פורמט של "X.X.X" שבו כל X יכול להיות כל צירוף תוויים הקסה-דצימליים

- (3) אתחול המערכת הנבדקת כדי לוודא שההגדרות שהגדכנו מחדש הוגדרו כמו שצרכן
- (4)פתיחה windbg על המערכת הבודקת, בחירת kernel debugging והכנסת ה-port ומספר ומספר debugging במקומות המתאים:

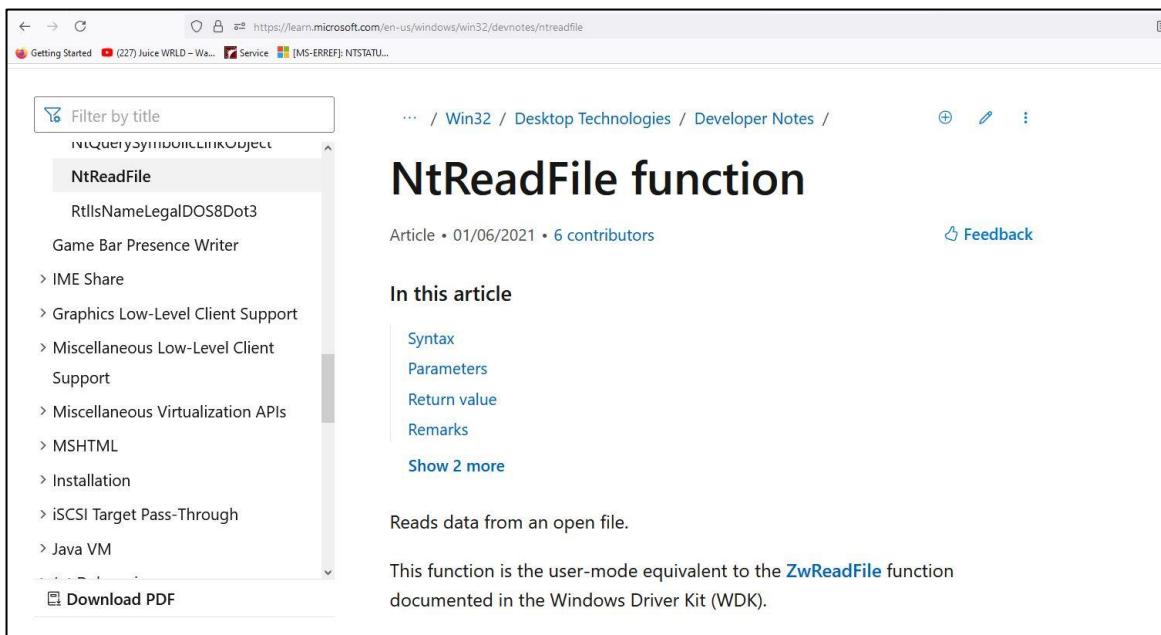


נראה לבסוף שהתהליך עבד כמו שצרכן נראה פלט דומה לפלט הבא:



:MSDN

עבור רוב הפעולות הקרנליות שאנו נא' מבנים איך להשתמש בהם יהו כבר הסברים קיימים באתר MSDN של מיקרוסופט. כדי להציג לאתרים נצטרך רק לכתוב את שם הפעולה, לדוגמה (NtReadFile(), בוגול ונראה באתר שכותרתו אמרה להיות "NtReadFile function (Wdm.h) .."). הכוורת כוללת את שם הפעולה ואת שם header או הספרייה שבו הפעולה מוגדרת\מושחרת, ונראה שהגענו למקום הנכון בהופעת מסר דומה למסר הבא:



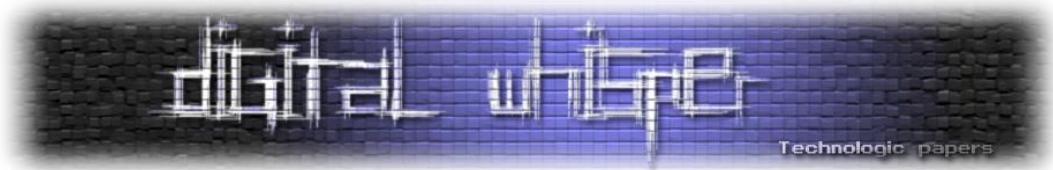
The screenshot shows a Microsoft Learn page for the NtReadFile function. The URL is https://learn.microsoft.com/en-us/windows/win32/devnotes/ntrreadfile. The page title is "NtReadFile function". It includes a sidebar with a search bar and a list of related topics like RtlIsNameLegalDOS8Dot3, Game Bar Presence Writer, and IME Share. The main content area has sections for Syntax, Parameters, Return value, Remarks, and a "Show 2 more" link. It also mentions that this is the user-mode equivalent to the ZwReadFile function documented in the Windows Driver Kit (WDK). There are navigation links for Win32, Desktop Technologies, and Developer Notes.

האתר יעזר לנו בהרבה מקרים שבהם לא נבין לגמרי מה כל פרמטר שעובר לפעולה אומר ומה הערך שלו צפוי להיות עבור מקרים שונים של שימוש, והוא בדרך כלל יקשר בסוף העמוד עם מודדים עם פעולות\كونספטים דומים וקשורים.

מבנה ה-Driver

כפי שתיארתי כבר במאמרם המקורי שלDriver זהה לכל תוכנה אחרת הניתנת להרצה על מכונת Windows. הוא תוכנה הבנויה לפי פורמט PE שמתאר תוכנות רבות של הבינארי הנitin להרצה כגון נקודת הכניסה של התוכנה (הפעולה הראשונה שתורץ אחרי הטענת התוכנה לזיכרון), פעולות המ约谈אות לתוכנה מתוך תוכנות אחרות (Import Address Table) וסוג הקובץ עצמו.

אך עדין יש כמה הבדלים בין תוכנת executable רגילה, לדוגמה notepad.exe, לבין Driver שאוטם נבחן עם .CFF explorer



ספק התוכנה:

The screenshot shows two side-by-side windows of CFF Explorer VIII. The left window displays the file 'KMDriver.sys' with properties like File Name (H:\current\KMDriver\x64\Release\KMDriver.sys), File Type (Portable Executable 64), and File Info (No match found). The right window displays the file 'notepad.exe' with properties like File Name (C:\Windows\notepad.exe), File Type (Portable Executable 64), and File Info (Microsoft Visual C++ 8.0). Both windows show detailed file structures including sections like Dos Header, Nt Headers, File Header, Optional Header, and Data Directories.

כפי שניתן לראות אין הבדל משמעותי בין התכונות הבסיסיות של שני הקבצים, הרו' שניותם קבצי "Portable Executable 64". השוני היחיד שניתן להבחין בו הוא שהקובץ ה-Driver אין חלק של provider, הסיבה לכך היא שה-Driver זהה הוא לא חתום על ידי מיקרוסופט או trusted third-party vendor אחר. זהו מושך ש-Ani בuild שמי שמספק אותו למערכות Windows, החלק מיוחד שאינו בנוי עבור הדוגמא ולכן בغالל שאין provider רשמי שמספק אותו למערכות Windows, שקשרו ל-provider של התוכנה ריק. כמובן ש-Driver כזה לא יכול לרוץ על מערכת רגילה בغالל DSE שונה מ-unsigned Drivers לדוגמה רוץ על מערכות שלא מופעל בהם test-sign וונעדו רק לבדיקת Driver-ים

:Import Directory

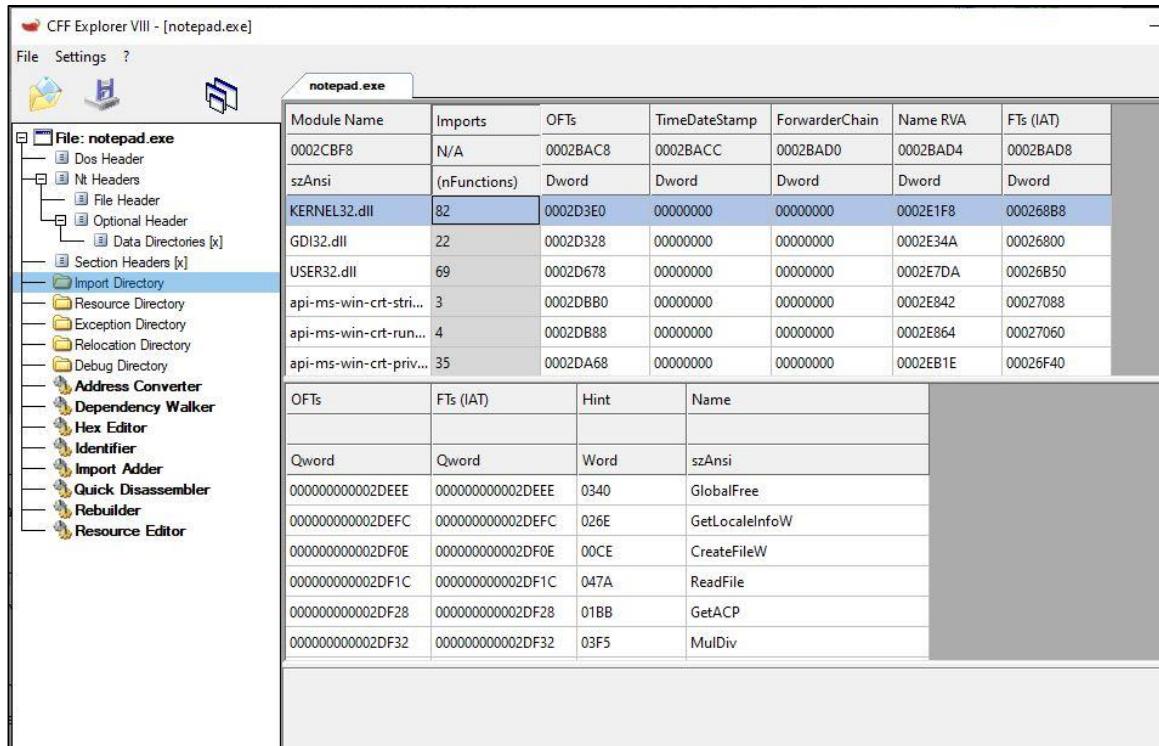
אזרז Import Directory בתוכנה הבנויות על פורמט PE אחראי לטעוד את כל התכונות החיצונית שמהם התוכינה הנוכחית צריכה ליבא פעולות, ומה הם הפעולות שמיובאות לתוכינה הנוכחית מתוך כל תוכינה חיצונית.

באיזור זה נצפה לראות הבדל משמעותי בין התוכנות, הרו' exe notepad.exe (תוכנה רגילה שרצה ב-usermode) כמעט ולא תעשה שימוש באוטן פעולות מיובאות ש-Driver ב-kernemode הרץ בעשויה בהם שימוש חזק מכמה פעולות בסיסיות. ניתן לראות את ההבדל הזה בתמונה הבאה:

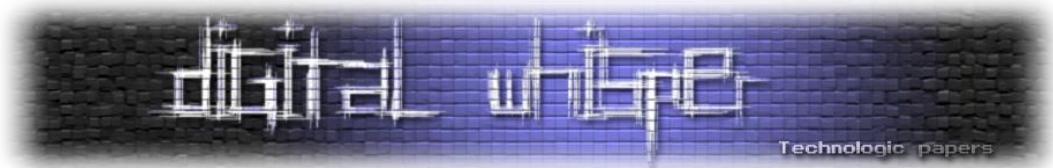
The screenshot shows two side-by-side windows of CFF Explorer VIII comparing the 'Imports' table for KMDriver.sys and notepad.exe. The left table for KMDriver.sys shows one entry: 'stAnsi (nFunctions)' with Module Name 'ntdll.dll', Imports 'Dword', OFTs 'Dword', TimeStamp 'Dword', ForwarderChain 'Dword', Name RVA '0001256A', and FIs (IAT) '0005F000'. The right table for notepad.exe shows multiple entries, including 'stAnsi (nFunctions)', 'KIRNLI32.dll', 'GDI32.dll', 'USER32.dll', and several entries for 'api-ms-win-*' DLLs, each with different module names, imports, and timestamps.

ההבדל הראשון שאפשר לשים לב אלו הוא הספריות המ约谈ות לתוך התוכנה הנוכחית. בזמן שה-Driver מייבא רק פעולות מ-ntoskrnl.exe (הקרנל עצמו), notepad.exe מייבא פעולות מגוון dllים שונים, בין היותר dll32.dll ו-user32.dll המתנים לתוכנה גישה לכל פעולות ה-API Windows. בגלל שה-Driver הוא תוכנה רצחה ליד הקרナル הוא ניגש ישיר לפעולות המוממשות בתוך הקרナル עצמו ומשתמש בהם בדומה ל-API Windows ש-notepad.exe משתמש, ובגלל שאין צורך בסיסי ל-Driver לעשות שימוש בספריות קרנליות אחרות הקרナル יהיה לרוב הספרייה היחידה שמ约谈ות בתוכנת Driver.

הבדל נוסף שנitin לשים לב אלו בפתחה של טבלאות הפונקציות约谈ות הם השמות של הפעולות המ约谈ות מכל ספריה. נתחיל עם exe notepad.exe והספרייה dll:kernel32.dll



ניתן לראות מה שימוש בפעולות API Windows ידועות כמו Win32 CreateFile handle עבור אובייקטים כמו קבצים ו-ReadFile הקוראת כמה מוסימת של מידע מຕוך אובייקט מסוים. בחיפוש ב-MSDN נראה שהפעולות באמת יכולות בתוך API Win32, מה שאומר שהם יכולות מבוססות usemode שנותנות לתוכנות ב-usemode גישה למשאים שונים במערכת ויגרמו לפעולה מתאימה בקרナル לרוץ כדי למש את מטרת הקיראה.



כשנחפש את הגרסה הקרנלית של אותן פעולות נראה ייחוס שונה, שבמקום ל"יחס את הפעולה ל-"Win32 API" מיחס את הפעולה ל-"API

The screenshot shows a Microsoft Learn page for Windows App Development. The URL is <https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-readfile>. The page title is "ReadFile function (fileapi.h)". It includes a sidebar with a search bar and a list of related functions like ReadFileEx, ReadFileScatter, RemoveDirectoryA, etc. The main content area has tabs for Syntax, Parameters, Return value, and Remarks, with a "Show 2 more" link. A note at the bottom states: "Reads data from the specified file or input/output (I/O) device. Reads occur at".

כשמסתכלים על הפעולות המ约谈oted מ-`ntoskrnl.exe` בתוכנת ה-`Driver` ניתן להבחן בפעולות השונות לחילוטין מהפעולות שראינו ב-`notepad.exe` שמותאמות לרוץ ב-`kernelmode` וממומשות על ידי חלקים שונים של ה-`Windows kernel`:

The screenshot shows the CFF Explorer VIII interface for the file `KMDriver.sys`. The left pane shows the file structure with sections like Dos Header, Nt Headers, File Header, Optional Header, Data Directories, Section Headers, Import Directory, Exception Directory, Relocation Directory, Debug Directory, Address Converter, Dependency Walker, Hex Editor, Identifier, Import Adder, Quick Disassembler, Rebuilder, and Resource Editor. The right pane displays two tables. The top table is titled "Imports" and lists entries for `szAnsi` and `ntoskrnl.exe`. The bottom table is titled "ForwarderChain" and lists various kernel functions such as `ExAllocatePoolWithTag`, `ExFreePoolWithTag`, `ZwCreateFile`, `PsInitialSystemProcess`, `PsCreateSystemThread`, and `ZwClose`.

כמו כן, ניתן לראותprefix קבוע בפעולות קרנליות המ约谈oted מהקרנל עצמו - .."Nt.." - .."Zw.." . שני ה- prefixes האלה יכולים להתקיים בו-זמןית וההבדל היחיד ביניהם הוא שגרסת ה-Zw לא מבצעת בדיקות על

הפרמטרים שנשלחו אליה וסומכת עליהם ככallow שהגינו מקריאה של Driver אמין, בשונה מגרסת ה-Nt שמודדת את הולידות של הפרמטרים הנשלחים אליה ולא סומכת על הקורא, ובפועל לאחר הבדיקה גרסת ה-Nt קוראת לגרסה ה-Zw שemmמת את הפעולה עצמה:

The screenshot shows a Microsoft Learn article page. The left sidebar contains a navigation menu with the following items:

- Filter by title
- Programming techniques
- Using Nt and Zw Versions of the Native System Services Routines
- PreviousMode
- Libraries and Headers
- What Does the Zw Prefix Mean?
- Specifying Access Rights
- NtXxx Routines
- Synchronization
- Using Common Log File System
- Kernel Transaction Manager
- Dynamic Hardware Partitioning Techniques

The main content area discusses the differences between Nt and Zw versions of native system services routines. It states that each routine has two slightly different versions with similar names but different prefixes. For example, calls to NtCreateFile and ZwCreateFile perform similar operations and are, in fact, serviced by the same kernel-mode system routine. For system calls from user mode, the Nt and Zw versions behave identically. For calls from a kernel-mode driver, the Nt and Zw versions differ in how they handle parameter values.

A specific section highlights that a kernel-mode driver calls the Zw version of a native system services routine to inform the routine that the parameters come from a trusted, kernel-mode source. In this case, the routine assumes that it can safely use the parameters without first validating them. However, if the parameters might be from either a user-mode source or a kernel-mode source, the driver instead calls the Nt version of the routine, which determines, based on the history of the calling thread, whether the parameters originated in user mode or kernel mode. For more information about how the routine distinguishes user-mode parameters from kernel-mode parameters, see PreviousMode.

:Optional Header

לפנינו נראה הלאה רציתי לעבור גם על optional header שכולל הרבה מידע על כל אחת מהתוכנות כמו נקודת הכניסה של הקוד בקובץ, כמות ה-sections והכתבות הירטואלית שה-linker בחר עבור התוכנה. כפי שתיארתי כבר לא נזהה להבדל משמעותי בין התוכנות בגלל שנייהן תוכנות בינהיהן ניתנות להרצת הבניות על אותו פורמט ונוצרות בצורה זהה:

The screenshot shows the CFF Explorer VIII interface. On the left, a tree view shows the file structure for 'notepad.exe'. The 'Optional Header' node is selected. On the right, a detailed table of the optional header fields is displayed:

Member	Offset	Size	Value	Meaning
AddressOfEntryPoint	00000128	Dword	00023BC0	.text
BaseOfCode	0000012C	Dword	00001000	
ImageBase	00000130	Qword	0000000140000000	
SectionAlignment	00000138	Dword	00001000	
FileAlignment	0000013C	Dword	00000200	
MajorOperatingSystemVersion	00000140	Word	000A	
MinorOperatingSystemVersion	00000142	Word	0000	
MajorImageVersion	00000144	Word	000A	
MinorImageVersion	00000146	Word	0000	
MajorSubsystemVersion	00000148	Word	000A	
MinorSubsystemVersion	0000014A	Word	0000	
Win32VersionValue	0000014C	Dword	00000000	
SizeOfImage	00000150	Dword	00038000	
SizeOfHeaders	00000154	Dword	00000400	
CheckSum	00000158	Dword	00040C20	
Subsystem	0000015C	Word	0002	Windows GUI
DllCharacteristics	0000015E	Word	C160	Click here
SizeOfStackReserve	00000160	Qword	0000000000000000	
SizeOfStackCommit	00000168	Qword	00000000000011000	
SizeOfHeapReserve	00000170	Qword	0000000000100000	

CFF Explorer VIII - [KMDFdriver.sys]

File Settings ?

KMDFDriver.sys

Member	Offset	Size	Value	Meaning
AddressOfEntryPoint	00000108	Dword	00001730	.text
BaseOfCode	0000010C	Dword	00001000	
ImageBase	00000110	Qword	000000140000000	
SectionAlignment	00000118	Dword	00001000	
FileAlignment	0000011C	Dword	00000200	
MajorOperatingSystemVers...	00000120	Word	000A	
MinorOperatingSystemVers...	00000122	Word	0000	
MajorImageVersion	00000124	Word	000A	
MinorImageVersion	00000126	Word	0000	
MajorSubsystemVersion	00000128	Word	000A	
MinorSubsystemVersion	0000012A	Word	0000	
Win32VersionValue	0000012C	Dword	00000000	
SizeOfImage	00000130	Dword	00014000	
SizeOfHeaders	00000134	Dword	00000400	
CheckSum	00000138	Dword	0001ADA9	
Subsystem	0000013C	Word	0001	Native
DllCharacteristics	0000013E	Word	4160	Click here
SizeOfStackReserve	00000140	Qword	000000000000100000	
SizeOfStackCommit	00000148	Qword	0000000000001000	
SizeOfHeapReserve	00000150	Qword	000000000000100000	

כפי שניתן לראות הבדל בין הערכים הגינוי ולא צזה שיכל להצביע על מאפיינים שונים או תהליכי בניה שונה, אך הבדל שאפשר לציין הוא שדה ה-"SubSystem" של כל תוכנה. ערך זה מתעד את ה- dependencies במערכת שהכרחיתם עבור ריצה מוצלחת של התוכנה:

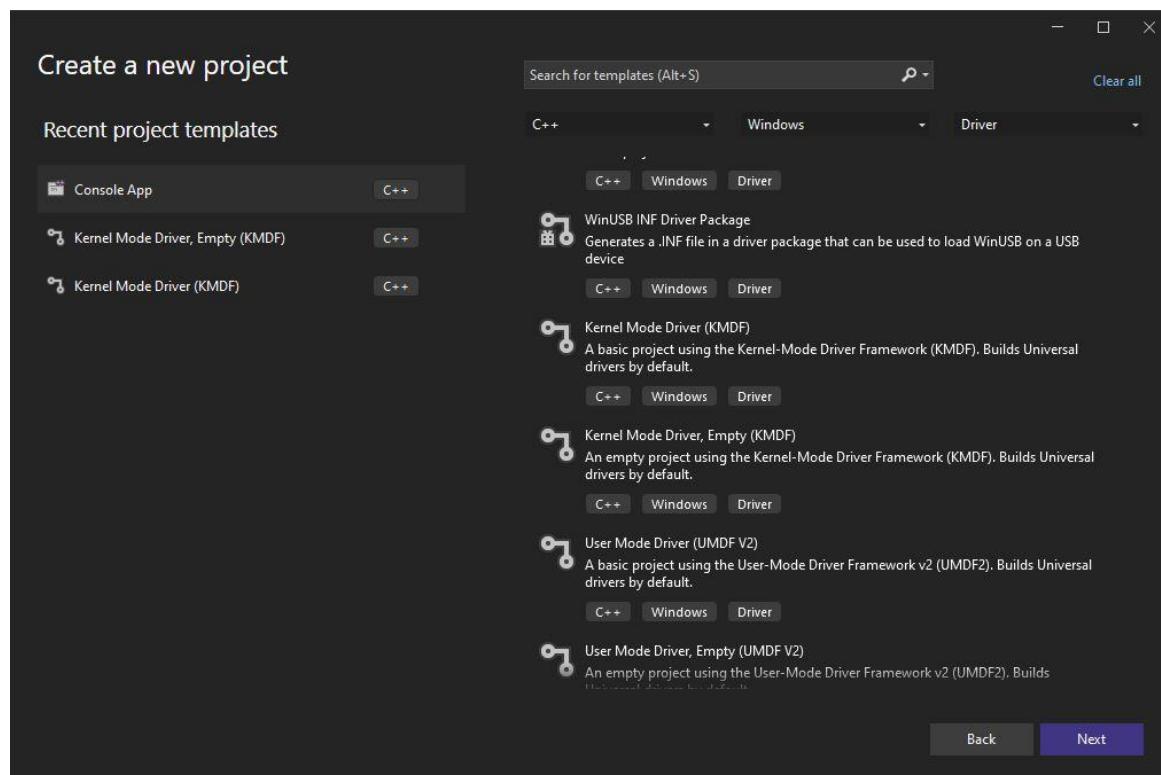
- ערך השדה הוא "GUI", תוכנה המבוססת על ממשק המשתמש הגרפי של Windows לתוכנות רגילוט הרצאות ב-usermode •
- ערך השדה הוא "Native", תוכנה המבוססת רק על התהליכים וה-Drivers הבסיסיים שקיים בכל מערכת Windows •

כתיבת Driver ב-**בסיי**

לאחר הבנת הסביבה והתוכנה שאנו מנסים לכתוב אני אכנס לזרת הכתיבה של Driver בסיסי, אני אתאר את הדוגמא שהכנתי ואת המאפיינים המיוחדים של ה-Driver הגרפי הזה ומה ניתן לעשות אליו. תוך כדי מעבור על עקרונות, מבני נתונים ופעולות שאסבירות המשך או מעבור עליהם בלבד להתעמק בהgel שהסבירתי עליהם כבר במאמרים קודמים ב-digital whisper. קרייה של שאר המאמרים לפני המאמר זהה מומלצת!

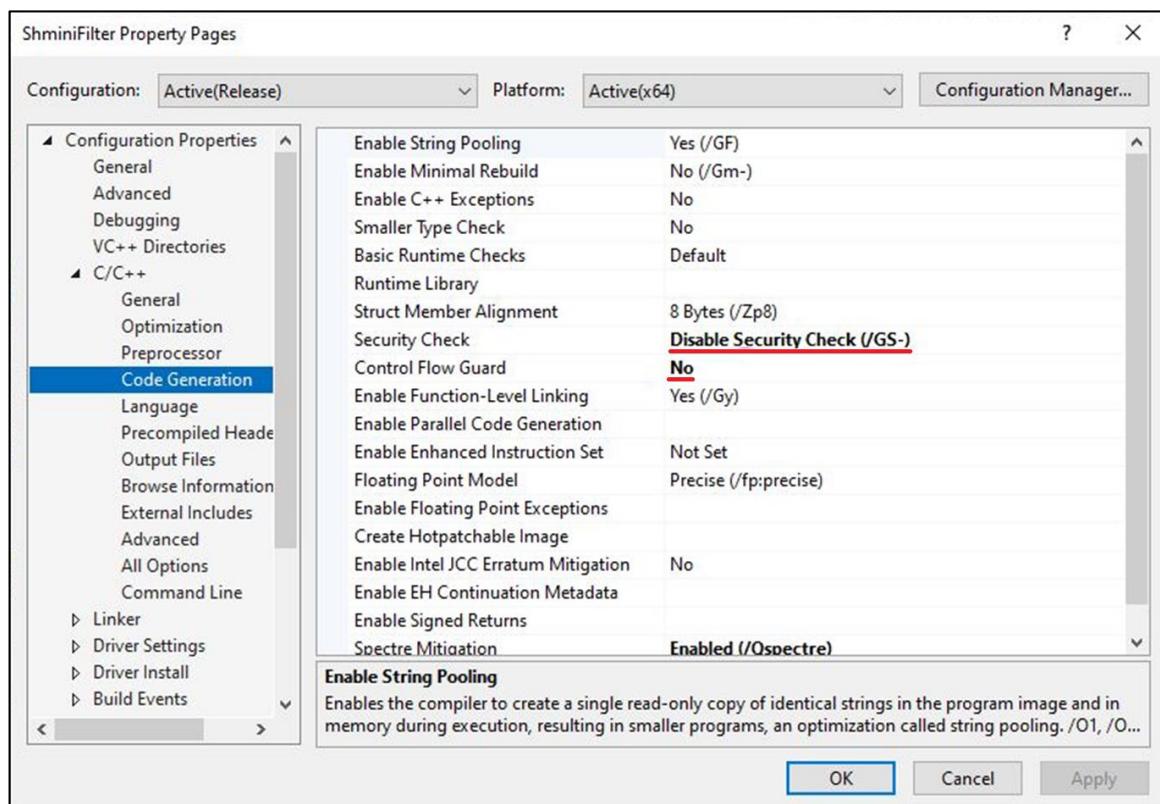
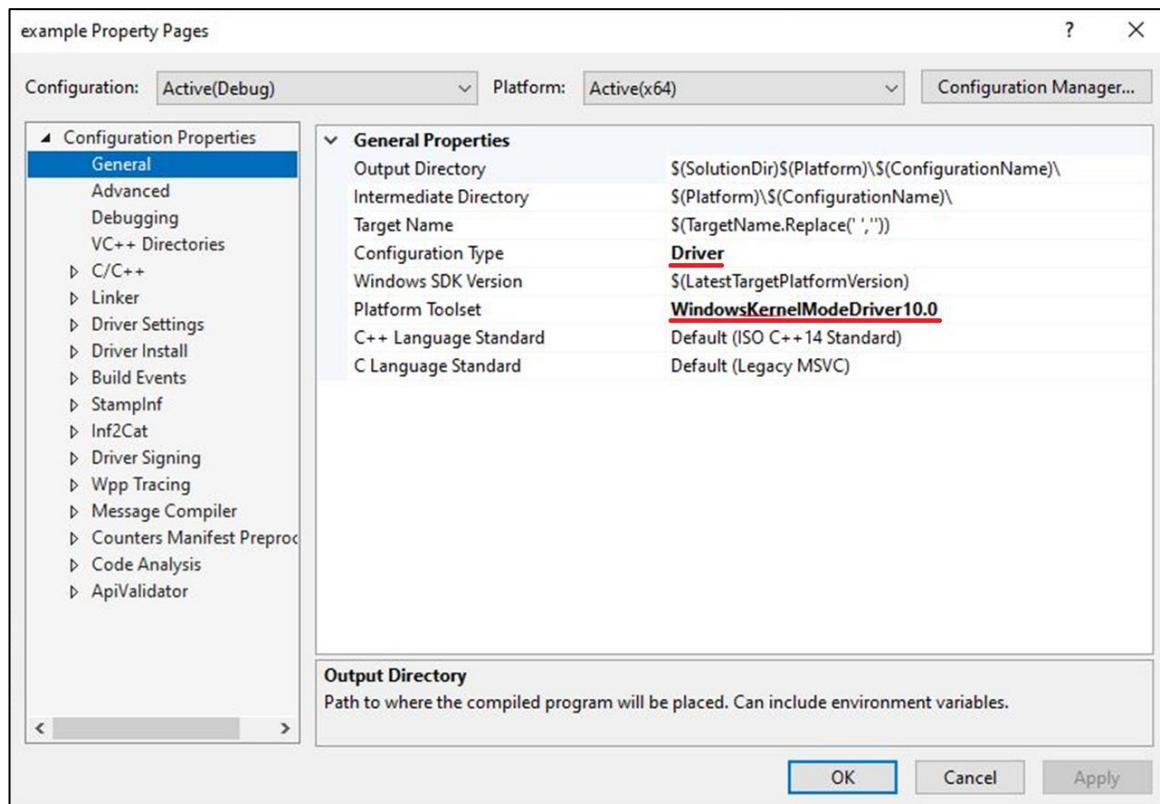
יצירת פרויקט לכתיבת Driver:

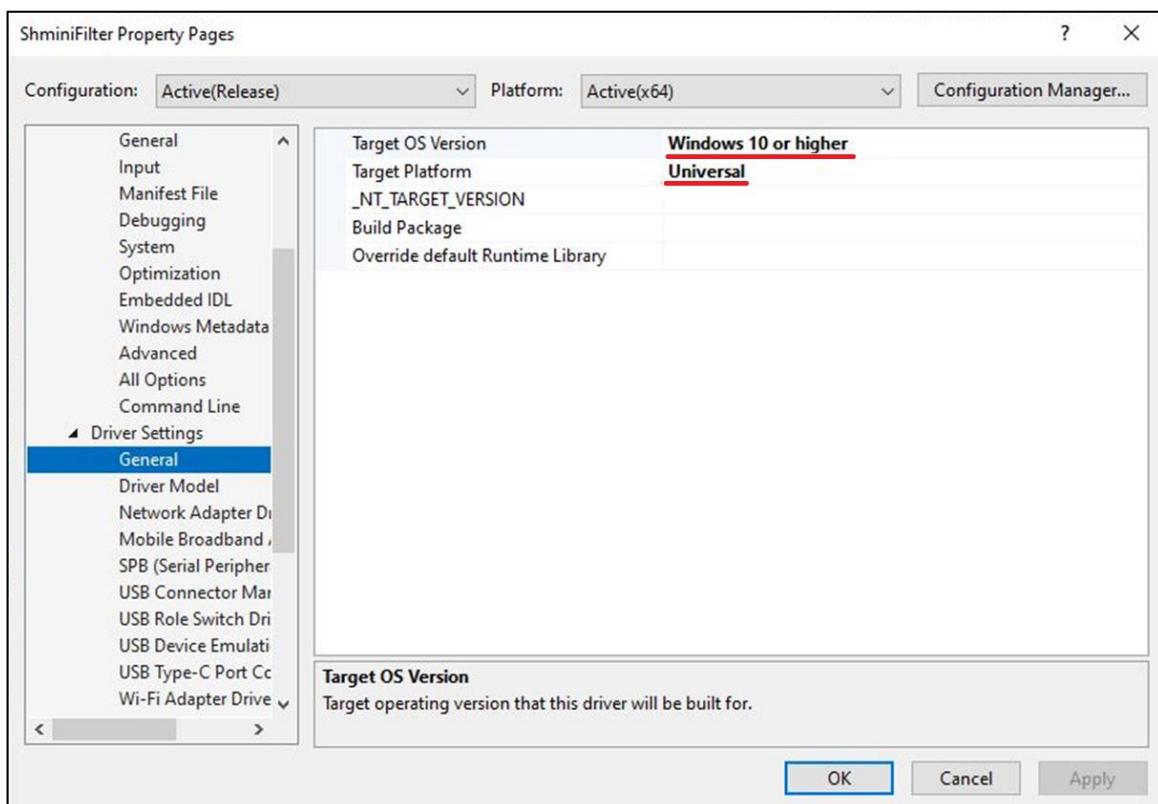
שלב זה מאוד קל כמו יצירת פרויקט לכל תוכנה ב-Visual Studio. כפי שהראיתי בשלב הרמת הסביבה לפיתוח ה-Driver, עם ה-WDK הותקנו גם templates שונים עבור פיתוח סוגיים של Driver-ים ב- "Kernel Mode Driver". במקרה של Driver אין סיבה לבחור סוג פרויקט אחר שהוא לא "Kernel Mode Driver, Empty" או



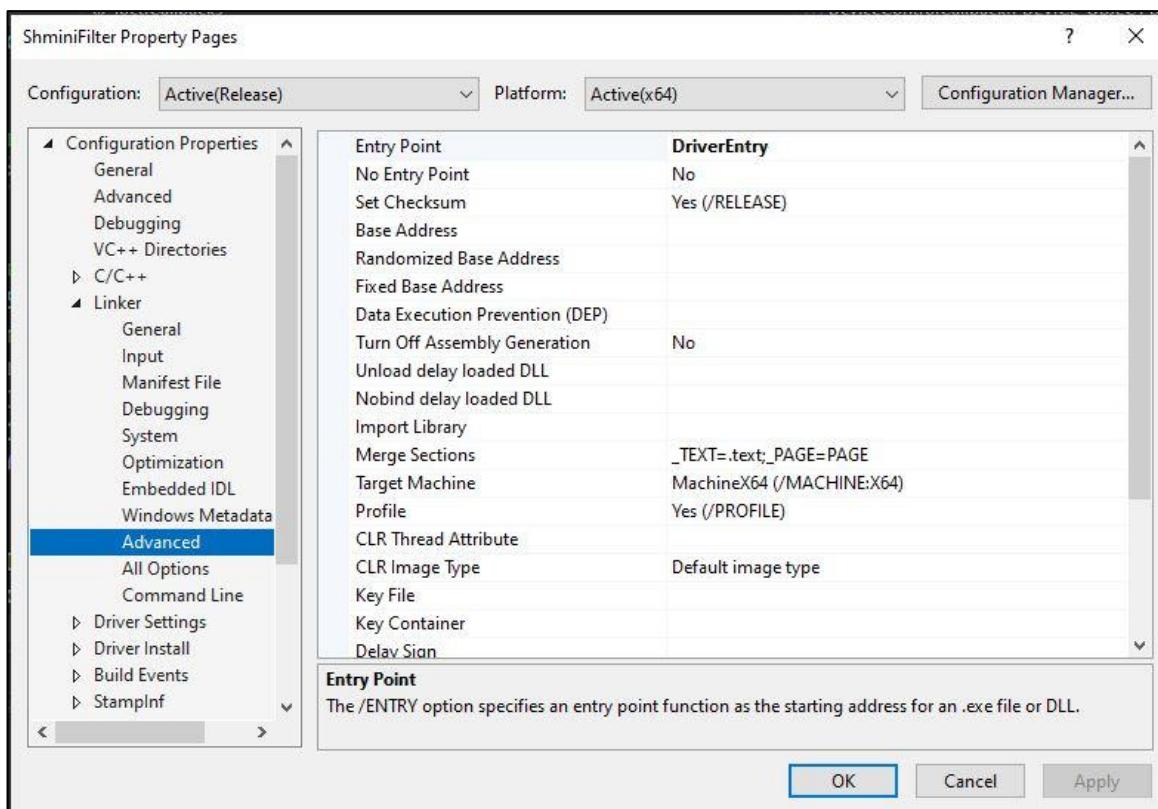
לאחר בחירת סוג הפרויקט ויצירת הפרויקט יש כמה צעדים חשובים לבצע לפני התחלת כתיבת הקוד:

(1) לוודא שהגדרות הפרויקט נמצאות על ההגדרות הבאות:



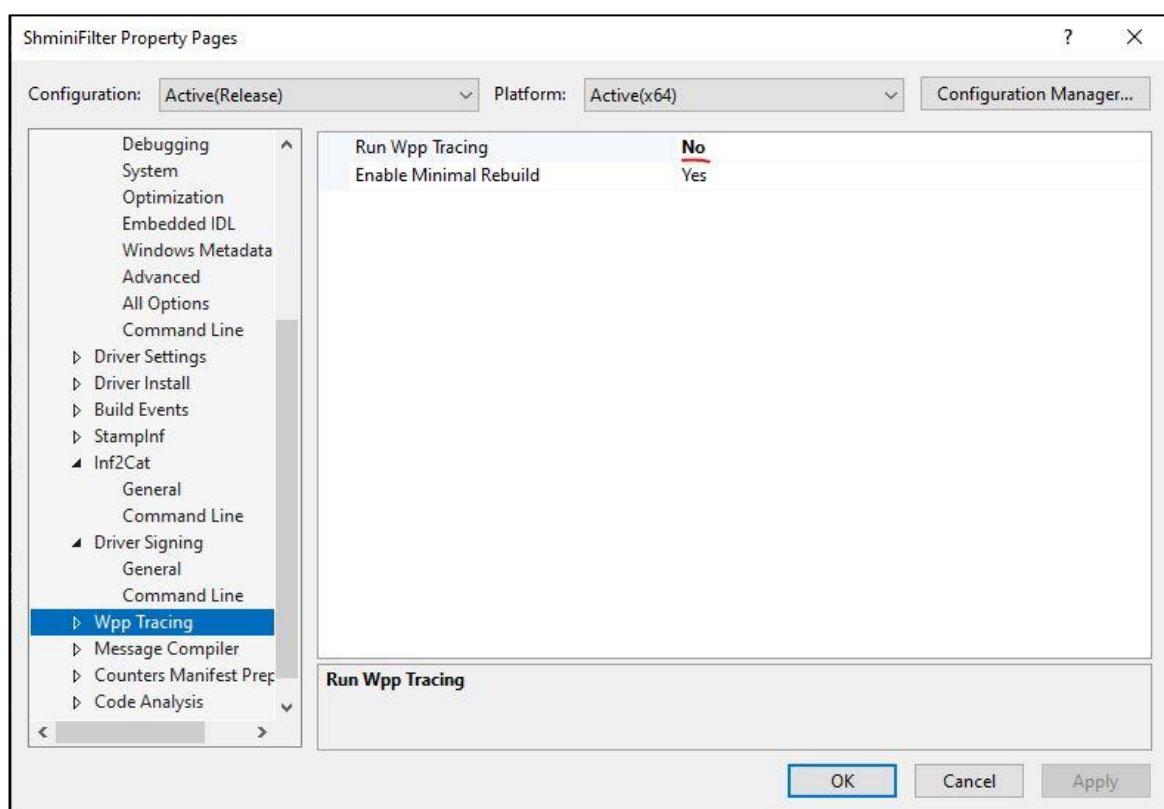
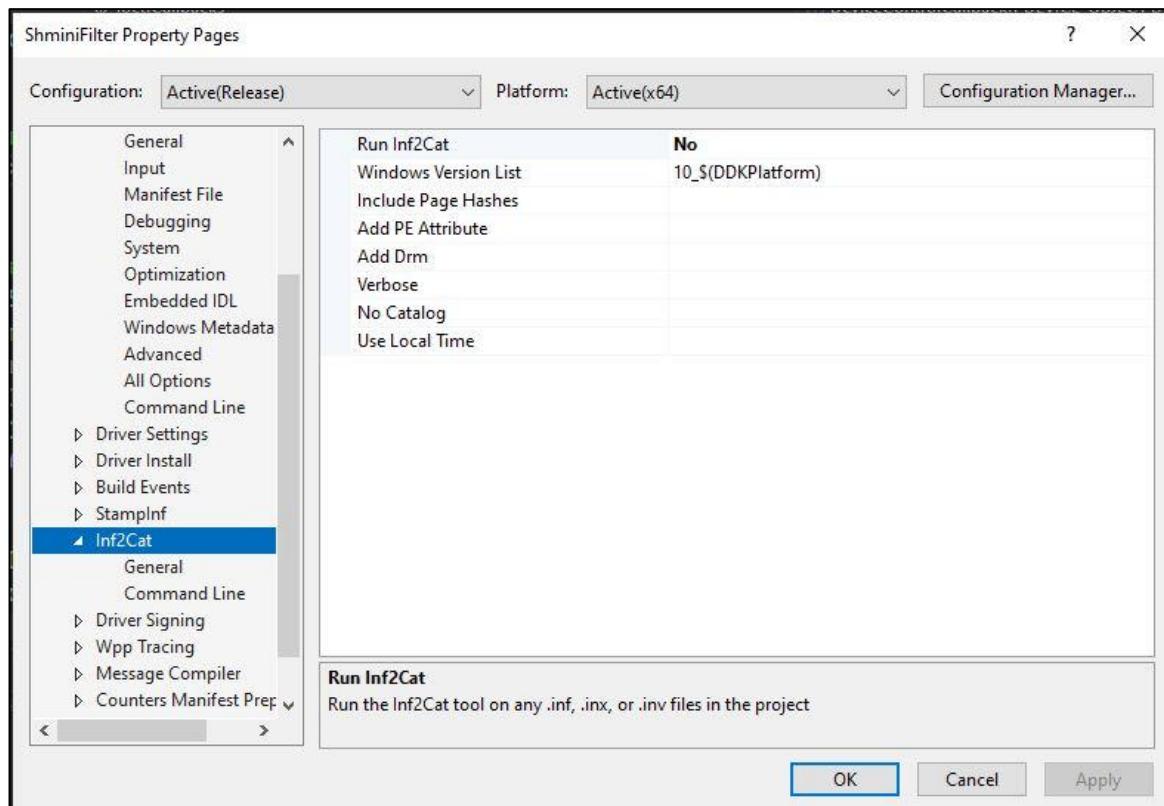


(2) לוודא שפועלות הكنيיה של ה-Driver מוגדרת בהתאם לשם של פועלות הكنيיה שלנו בקוד. לרוב השם משמש כ-common practice- **DriverEntry()** ולכן נctrkr לוודא שהשם כתוב בשדה זהה הוא : "DriverEntry"

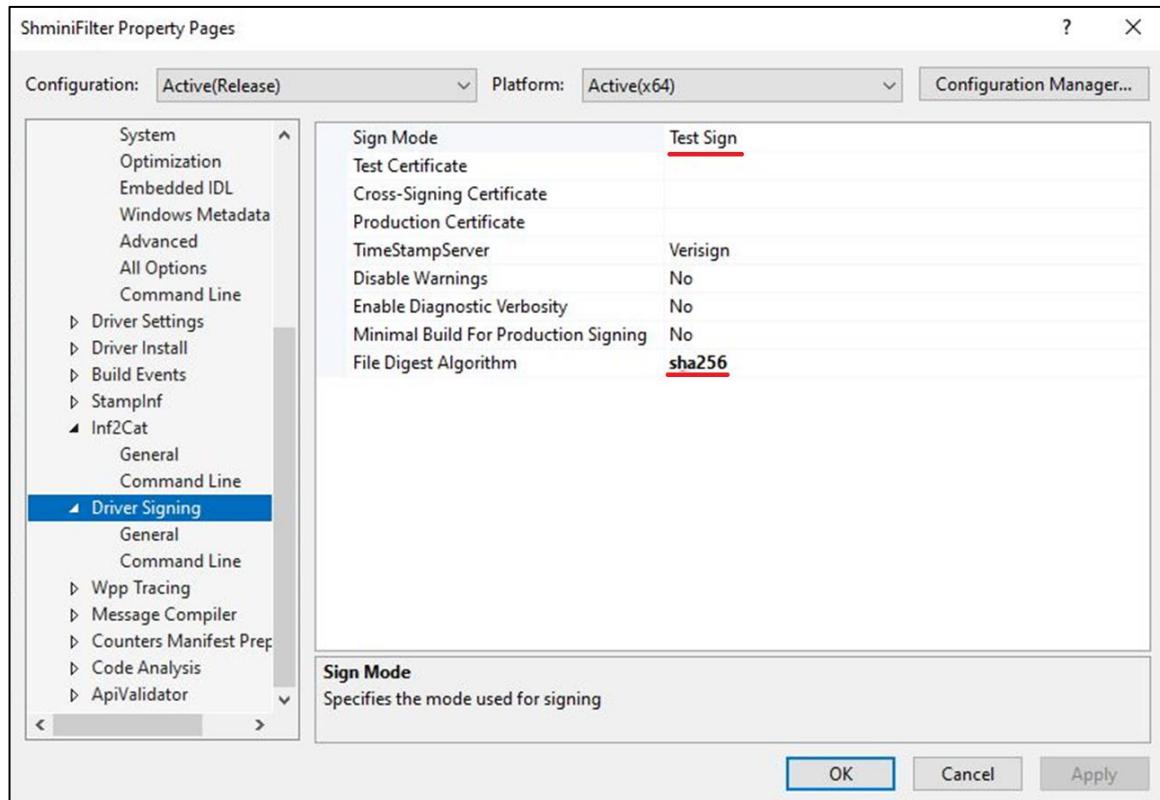


(3) נטרול Inf2Cat ו-WPP שיכולים לגרום לשגיאות בנייה לא צפויות ולא הכרחיים לפעולות ה-

:Driver



4) בטור Driver Signing יש צורך לציין את סוג החתימה של ה-Driver, כולם אם הוא יהיה בשימוש תחת test sign או שיקבל חתימה דיגיטלית רשמית מ-certificate authority. במקרה שלנו נציין sign test signing ונויסף אלגוריתם hashing כלשהו בשורת המ命רך (לא חשוב איזה):



עכשו נוכל באמת להתחיל לכתוב את הקוד עבור ה-Driver שלנו עם כמה מינימלית של שגיאות והזהרות לא צפויות ולא מובנות שייקשו על תהליכי הפיתוח. יש לציין שבאופן ברירת מחדל בגל הכוח שיש לפROYיקט על המערכת, כמעט כל הערה שתהייה לנו בקוד תהפוך לשגיאה שתמנע מסיום בנין הפרויקט.

אם נרצה למנוע משגיאה מסוימת להתרחש בקובץ קוד מסוים (לאחר בדיקת ההערה כדי לוודא שמקור ההערה לא יגרום לבעה בהמשך) נוכל להשתמש ב-(`#pragma warning(disable : warning_number)`):

```

1  #include "hooking.h"
2  #include "HookingGlobals.h"
3  #pragma warning(disable : 4996)
4  #pragma warning(disable : 4302)
5  #pragma warning(disable : 4311)
6  #pragma warning(disable : 4127)
7

```

פעולות ה-DriverEntry וה-DriverUnload

כפי שתיארתי, ()DriverEntry היא פועלות הכניסה של ה-*Driver* שנכתב, מקבילה לפעולות ()main בתוכנות אחרות. הפועלה צריכה להיות מוכנה לקבל 2 פרמטרים:

(1) מצביע ל-DRIVER_OBJECT (או טיפוס PDRIVER_OBJECT). ברגע הטענתה ה-*Driver* לזכרון המערכת
ויצרת לו אובייקט מסווג DRIVER_OBJECT שמכיל את כל התוכנות שצריך לדעת על אותו *Driver*, כמו ה-
path image של קובץ ה-*Driver*, גודל הקובץ של ה-*Driver* ומספר פעולות חשובות שה-*Driver* מממש
עבור טיפול באירועים שונים במערכת

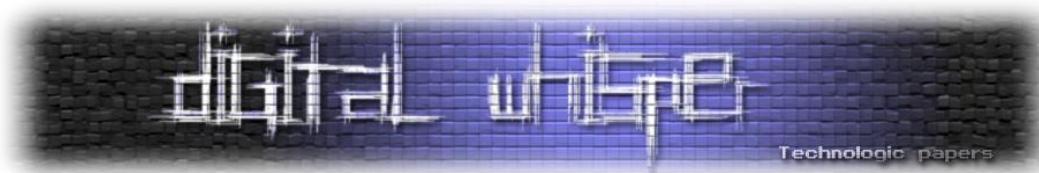
(2) registry key שנותר עבורי ה-Driver (מтипוי UNICODE_STRING). כמו כל service שקיים על המערכת, גם ל-Driver יש מפתח ב-registry המכיל מידע על סוג ה-service (במקרה זה Kernel), צורת עליית ה-Driver לדיזיין (אוטומטיות ב-boot, ידנית לפי בקשה...), המסלול לקובץ ה-Driver בזיכרון ועוד.

סיפוק המסלול ל-Driver יכול לספק ל-Driver גישה לכל הרכיבים שמאוחסנים עבورو ב-registry:

מכאן ה-*Driver* יכול לעשות כרצונו עד סוף פעולה ה-*DriverEntry*, כגון אתחול נתונים, הקצאת זיכרון או קיראה לפעולות אחרות. במקרה שלנו אני אדגים את המשך הנפוץ ביותר של הפעולה שמיועד לתת ממשק לאפליקציות אחרות (גמ אפליקציות קרנליות וגם אפליקציות רגילוט) לתקשר עם ה-*Driver* ולהפעיל פעולות לפי בקша מיוחדת:

(1) Windows devices- ב- Driver ה- über ה- DEVICE_OBJECT יצירת. במאמר הקודם סקרתי הרבה את עז ה- ומה device נועד לעשות, אך בקצרה המטרת של device ה- היא לתת לאפליקציות אחרות גישה לפעולות שמאומנויות בתוך ה- Driver, ככלומר: ה- device נווטן לנו אפשרות לעשות trigger למגוון פעולות כמו Driver מיש, בדרך כלל עבור תקשורת בין האפליקציה שלנו ל- Driver או בקשות מיוחדות כמו קריאה\ כתיבה של קבצים. את זה נעשה עם הפעולה () IoCreateDevice ושם ל- device שאמור להיות רשות רפורטינו:

\Device\DeviceName



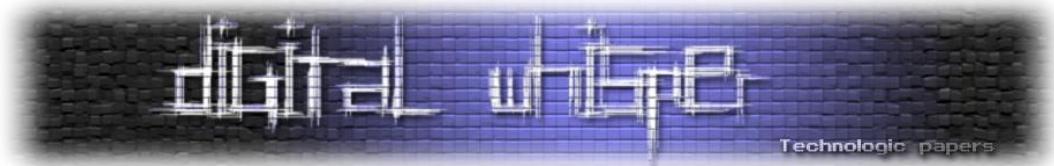
(2) יצירת symbolic link עבור ה-Driver. מהצד של האפליקציה, כדי לתקשר עם device שאינו בקשרו של Driver מסויים נדרש להשיג handle לאותו Driver, בדיקות כמו קובץ. בשימוש ב-CreateFileA() או פעולה מקבילה יש צורך לספק ערך מזהה של ה-Device, במקרה של קובץ זה יהיה המסלול שלו במערכת הקבצים ובמקרה זה Driver זה יהיה ה-Device symbolic link שלו. ניצור symbolic link שנקבלת את שם ה-device שנוצר לפני כן ואת ה-symbolic link שנרצה ליצור, נהוג לתת ערך-link ל symbolic sys לפי השם שהועבר גם ל-device (אך לא חיבר).

בפורמט \\\?\DeviceName או בפורמט \DosDevices\DeviceName

בפועל בתוך ה-DRIVER_OBJECT יש מערך של מצביעים לכל פעולה, כאשר ערכיהם כמו IRP_MJ_CREATE הם אינדקס בתוך המערך, המציין פעולה שתורצ'-Driver עברו שליחת בקשה מסוימת. במקרה זה-IRP major code IRP major code שה-Driver לא מימוש או השאיר בתוך NULL במערך הבקשה תועבר ל-device הבא בתוך שהוא אמרו לעבור בו, לפעמים Driver-ים ימשו פעולות כמו IRP_MJ_CREATE אך בגל שאין צורך להריץ קוד נוספת מיעבד הפעולה שה-Driver מימוש תחזור ישר מבלי לעשות כלום.

(4) רישום פעולה DriverUnload כחלק מפעולות ה-`dispatch`. פעולה `DriverUnload` נמצאת גם כן בתוך ה-`DRIVER_OBJECT` והוא תרוץ במקרה והמערכת מנסה להוציא את ה-`Driver` מהזיכרון ולהפסיק את פעילותו. הפעולה בדרך כלל משחררת אובייקטים, משחררת זיכרון מוקצה ומוחקת קישורים ל-`Driver` באמצעות פונקציית `IoDeleteSymbolicLink()` או `IoDeleteDevice()`. הפעולה זו תקבל פרמטר אחד שהוא ה-`DRIVER_OBJECT` הקשור ל-`Driver`.

```
VOID DriverUnload(_In_ PDRIVER_OBJECT DriverObject) {
    UNREFERENCED_PARAMETER(DriverObject);
    if (MinifilterHandle != NULL) {
        FltUnregisterFilter(MinifilterHandle);
        MinifilterHandle = NULL;
    }
    IoDeleteSymbolicLink(&SymbolicLink);
    if (FilterDeviceObject != NULL) {
        IoDeleteDevice(FilterDeviceObject);
        FilterDeviceObject = NULL;
    }
    DatabaseCallbacks::DeleteDatabase();
    DbgPrintEx(0, 0, "Shminifilter general - DriverUnload() called\n");
}
```



```
#include "FilterCallbacks.h"
#include <ntddscsi.h>

// Global variables:
UNICODE_STRING SymbolicLink = RTL_CONSTANT_STRING(L"\DosDevices\ShminiFilter");
UNICODE_STRING DeviceName = RTL_CONSTANT_STRING(L"\Device\ShminiFilter");
PDEVICE_OBJECT FilterDeviceObject = NULL;
PFLT_FILTER MinifilterHandle = NULL;
```

מבנה Dispatch Function

:PDRIVER_DISPATCH צריכה להיות בפורמט המוגדר בתור Dispatch function

```
_Function_class_(DRIVER_DISPATCH)
_IRQL_requires_max_(DISPATCH_LEVEL)
_IRQL_requires_same_
typedef
NTSTATUS
DRIVER_DISPATCH (
    _In_ struct _DEVICE_OBJECT *DeviceObject,
    _Inout_ struct _IRP *Irp
);

typedef DRIVER_DISPATCH *PDRIVER_DISPATCH;
```

כלומר: פעולה המקבלת את ה-DEVICE_OBJECT שה-Driver יצר ואת ה-IRP הרלוונטי לבקשתו ומחזירה NTSTATUS לפי הגדרות הפעולה. כפי שציינתי dispatch function לא חייב להיות ממומש עבור אף פעולה (במקרה זה הערך במערך יהיה NULL), הפעולה יכולה גם להיות ממומשת אבל לא לעשות כלום ובמקרה זה הפעולה רק תעביר את ה-IRP ל-Driver הבא בתור. כאן נמצאת דוגמא לפעולה כזו עבור IRP_MJ_CLOSE-
:IRP_MJ_CREATE-

```
NTSTATUS IoctlCallbacks::CreateCloseCallback(PDEVICE_OBJECT DeviceObject,
    PIRP Irp){
    UNREFERENCED_PARAMETER(DeviceObject);
    PAGED_CODE();
    Irp->IoStatus.Status = STATUS_SUCCESS;
    Irp->IoStatus.Information = 0;
    IoCompleteRequest(Irp, IO_NO_INCREMENT);
    return STATUS_SUCCESS;
}
```

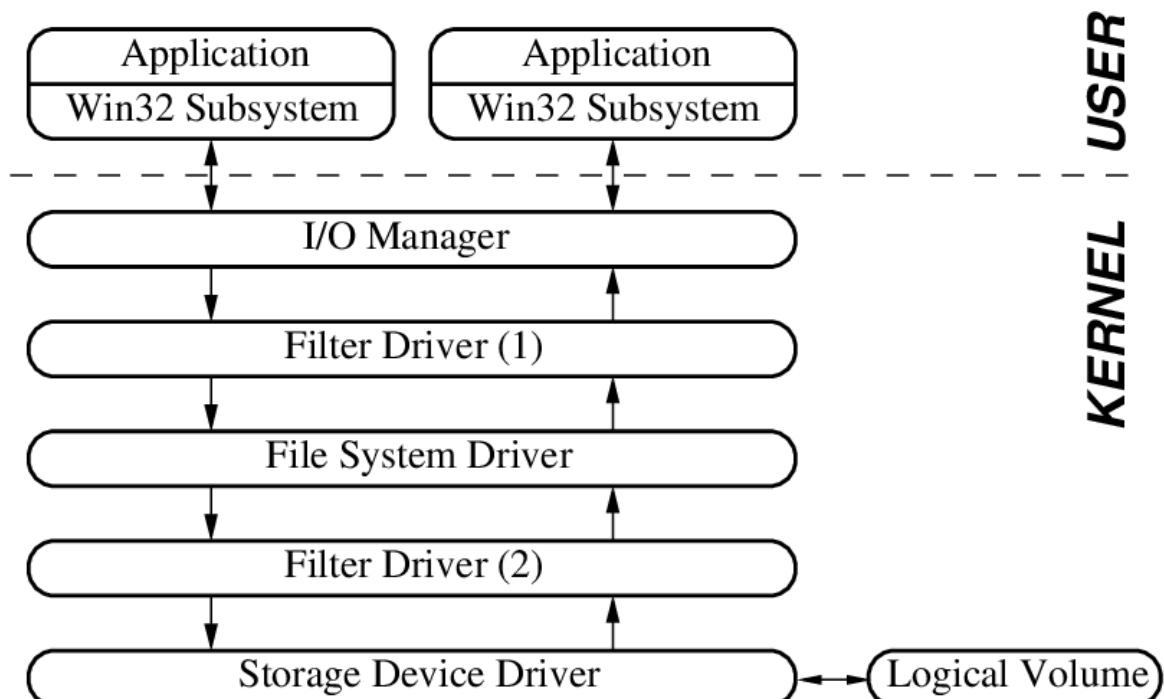
IMPLEMENT הפעולות יהיה תלוי ב-Driver, לדוגמה filesystem Driver נראה ימשך פעולות עבור כתיבה, קוראה, יצירה וסגירה של handles ועוד, אך Driver שנועד לבצע תקשורת מינימלית עם אפליקציות אחרות לא יצטרך ממש את אותן פעולות.

למרות השוני, כמעט כל Driver ימשך פעולות DeviceControl (אינדקס של IOCTL) של DeviceControl שנוועדה עבור תקשורת בין אפליקציות אחרות לבין ה-Driver שלנו. הפעולה הזאת יכולה להיות מופעלת גם על ידי אפליקציות הרצות ב-usermode באמצעות IOCTLs. במקרה שהפעולה הזאת קריטית לרוב ה-Drivers אחרים בזיכרון או CallDriver-IoBuildDeviceControlRequest או IoCallDriver. במקרה שהפעולה הזאת קריטית ל-Driver אחד או מושתת לדוגמה, מי-autorization בפרק הבא של המאמר רשאי על סוג ה-Driver הספציפי שאינו מושתת לדוגמה, מי שידוע איך Minifilter Driver עובד יכול לדלג על החלק הזה

Filesystem device stack

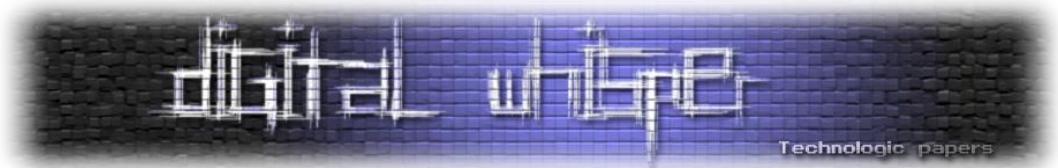
בכל מערכת Windows קיים device tree שמציאן את כל המכשירים שקיים ופועלם על המחשב עבור מטרות שונות, כמו חיבור USB, מקלדת, עכבר ועוד. העץ מתחילה מ-root device-tree שஅחרה לטפל בכל ההודעות שצרכות להשלח למכשיר בעץ וקיימת היררכיה ברורה בתוך ה-device tree, לדוגמה אם אני רוצה לשלוח בקשה ל-device של המצלמה שלי הבקשתה תעבור קודם כל דרך ה-device של ה-USB hub אליו המצלמה מחוברת. מי שמעוניין להתעמק בנושא יכול להסתכל על המאמר הקודם שלי, אך החשיבות בהבנת העץ היא כדי להבין את זרימת הבקשתה שנשלחת בפעולה הקשורה לקבצים מרגע יציאת הבקשתה ועד החזרת התשובה וביצוע הבקשתה.

כך נראה זרימת הבקשתה (ה-IRP) כשמבוצע פועלה הקשורה למערכת הקבצים, כגון קרייה או כתיבה של נתונים:



[מקור: https://www.researchgate.net/figure/Windows-layering-model-Applications-issue-system-calls-which-are-dispatched-by-the-I-O_fig2_220398232

- 1) הבקשתה המקורית יוצאה מתחילה, כגון explorer file, בתור בקשה למידע כלשהו על הקובץ ובסופו של דבר מבוצעת SYSENTER ומעבר מ-usermode ל-kernelmode אל שאר חלקים המערכת הרלוונטיים שיש להם חלק בביצוע הפעולה
- 2) הבקשתה מגיעה ל-I/O manager שஅחרה לטפל בהעברת פעולות קלט/פלט מאפליקציות ל-device drivers במערכת וחזרה. במעבר זהה תבוצע ההמרה בין הפורמטרים הבסיסיים של הבקשתה למבנה IRP שככל חלק מערכות פנימי יכול להבין



- (3) הבקשה מגיעה ל-filesystem filter Drivers (fltmgr.sys) על ה-Filter manager. ה-Filter manager ממערכת הפעלה. ה-Filter manager מושך נוח לכתוב Driver-ים שיתעסכו עם מערכת הקבצים ללא הבלאגן וסבירות הכתיבה שהייתה קיימת קודם לכן, וכך Filter Drivers (או בשם האחר Minifilter Drivers) הם חלקים מסוימים מהרבה אנטיו-וירוסים, כליהם ניתור על פעילות קבצים וכל הצפנה ו恢復 מידע. מתכנתו צד שלישי יכולם להציג Minifilter Driver שלהם שדריכו תüber כבקשה למערכת הקבצים וכך ניתן לנתר כל פעולה הקשורה למיניפולציה של קובץ, לשנות את הפורמטרים או את התוצאות שחזרות יוצאות בעקבות בקשה שנשלחה או אפילו לעצור בקשوت במקום מבלי לחת להם להמשיך להתקדם בעז.
- (4) רק אחרי ה-Filter Driver תגיע ל-Filesystem Driver שארה ל-Filesystem, כגון sys_ntfs.sys. ה-Filesystem Driver יתרגם את הבקשה ואת הפורמטרים שלו לצורה שה-Driver האחראי על התקשרות האמיתית עם הדיסק יוכל להבין, ומשם הבקשה תגיע לדיסק הפיזי ותתבצע על הנתונים שמאוכסנים בדיסק.
- (5) בסוף ביצוע הפעולה המידע הרלוונטי לפעולה יחזור בסדר הפוך חזרה אל הקורא ששלח את הבקשה עם סטוטו הצלחת הבקשה וכל שאר הנתונים שקשרים לפעולה.

בדוגמא זו ה-Filesystem Driver פשוט שומרה חלק מהאפשרויות שה-Filter manager מושך לנוטר לכותבי Driver-ים ותוך כדי מראה דרך כללית לכתיבה Driver ושימוש בפעולות נפוצות עבור מטרות מציאותיות

:Minifilter Driver

בהתל"ר פיתוח של Minifilter Driver יש צורך במימוש של חלקים נוספים שלא קיימים ב-Driver רגיל:

- (1) יצרת מערך של `FLT_OPERATION_REGISTRATION` עבור כל סוג בקשה למערכת הקבצים שנרצה לנתר. מבנה זה עובד דומה ל-`IRP major function table` והפעולות שנמצין בכל מבנה עבור סוג בקשה מסוים יופעלו ככל מזרימת הבקשה שתיארתי מוקדם יותר. מבנה ה-`FLT_OPERATION_REGISTRATION` נראה כך:

```
//  
// Structure used for registering operation callback routines  
  
typedef struct _FLT_OPERATION_REGISTRATION {  
  
    UCHAR MajorFunction; // Major function that is being traced  
    FLT_OPERATION_REGISTRATION_FLAGS Flags; // Can be 0 for general use  
    PFLT_PRE_OPERATION_CALLBACK PreOperation; // Pre-operation dispatch function pointer  
    PFLT_POST_OPERATION_CALLBACK PostOperation; // Post-operation dispatch function pointer  
  
    PVOID Reserved1;  
  
} FLT_OPERATION_REGISTRATION, *PFLT_OPERATION_REGISTRATION;
```

כפי שניתן לראות במבנה יש אפשרות לספק callback function שתקרו בזרימת הבקשה אל הדיסק

לפני ביצוע הפעולה.

ויש גם אפשרות לספק callback function שתקרא בחזרת התשובה לבקשת מהדיסק אל הקורא לאחר ביצוע הפעולה. בסוף המערך נדרש להוסיף רשומה של "}" END_OPERATION_END { } IRP_MJ_OPERATION { }" כדי לסמל שאין עוד אירועים שנריצה לנתר.

(2) יצירת אובייקט FLT_REGISTRATION. המבנה הזה כולל את כל המידע הרלוונטי על ה-Filter, כולל כל ה-callbacks הנמצאים במערך ה-FLT_OPERATION_REGISTRATION, פועלות () DriverUnload () כמו ב-Driver רגיל ועוד נתונים נוספים שלא חיב לציין ערך עבורם:

```
>const FLT_OPERATION_REGISTRATION CallbacksArray[] = [ { ... } ]
>const FLT_REGISTRATION RegistrationInfo = {
    sizeof(FLT_REGISTRATION),           // Size
    FLT_REGISTRATION_VERSION,          // Version
    0,                                // Flags
    NULL,                             // Context registration
    CallbacksArray,                   // Operation callbacks
    GeneralCallbacks::UnloadFilterCallback, // FilterUnload
    GeneralCallbacks::InstanceSetupFilterCallback, // InstanceSetup
    GeneralCallbacks::InstanceQueryTeardownFilterCallback, // InstanceQueryTeardown
    NULL,                            // InstanceTeardownStart
    NULL,                            // InstanceTeardownComplete
    NULL,                            // GenerateFileName
    NULL,                            // GenerateDestinationFileName
    NULL                            // NormalizeNameComponent
};
```

(3) יצירת INF file מיוחד עבור ה-Driver. INF הוא פורמט קובץ מיוחד שמסופק עם Driver ומתאר את אופן ההתקינה שלו על המערכת. בלחיצה ימנית על הקובץ תנתן לנו האפשרות "Install" שת התקין את ה- Minifilter על המערכת. במקרה זה יש צורך לבצע מספר שינויים לקובץ inf עבור יצירת .Driver Akashar בסוף המאמר עמוד ב-MSDN שמતאר בפירוט כל שלב ביצירת קובץ מיוחד זהה.

בעבר עם WDK הותקן template נוסף במיוחד לפיתוח Minifilter Drivers אך בכלל כמה שינויים הוא לא קיים יותר ויש צורך את תהליך ההתקינה בצורה ידנית בעזרת file-.inf. סוג הפרויקט הוא "Kernel mode Driver" כמו רוב ה-Drivers וכדי לאתחל את ה-Filter Minifilter כמו שצורך להוסיף כמה פעולות ל- DriverEntry() וDriverUnload().

- Shmukbel את ה-FLT_OPERATION_REGISTRATION, את ה-DRIVER_OBJECT ומצביע FLTRegisterFilter ל-PFLT_FILTER שמכיל מקום למצביע handle מיוחד עבור ה-Filter. הפעולה הכרחית עבור רישום Minifilter בתוך התור של כל שאר ה-filters. במקרה והאתחול נכשל או ש-() DriverUnload נקרא קיימת פעולה מקבילה לפעולה זו - FiltUnregisterFilter Shmukbel את ה-PFLT_FILTER ומוציאה את ה-Filter מהתור.

- מקבל את ה-PFLT_FILTER ומתחילה להעביר את כל הבקשות למערכת הקבצים דרך ה-*Minifilter Driver* הנוכחי. פעולה זו אין פועלה מקבילה, אך בפועל (*DriverUnload()* נקראת כדי לעצור ו-"*לחסן*" סופית את ה-*Minifilter*

```

DriverObject->MajorFunction[IRP_MJ_CLOSE] = IoctlCallbacks::CreateCloseCallback;
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = IoctlCallbacks::DeviceControlCallback;
DriverObject->DriverUnload = DriverUnload;

// Register the filter driver:
Status = FltRegisterFilter(DriverObject, &RegistrationInfo, &MinifilterHandle);
if (!NT_SUCCESS(Status)) {
    DbgPrintEx(0, 0, "Shminifilter - FltRegisterFilter() failed with status 0x%x\n", Status);
    IoDeleteSymbolicLink(&SymbolicLink);
    IoDeleteDevice(FilterDeviceObject);
    return Status;
}

// Start filtering:
Status = FltStartFiltering(MinifilterHandle);
if (!NT_SUCCESS(Status)) {
    DbgPrintEx(0, 0, "Shminifilter - FltStartFiltering() failed with status 0x%x\n", Status);
    FltUnregisterFilter(MinifilterHandle);
    IoDeleteSymbolicLink(&SymbolicLink);
    IoDeleteDevice(FilterDeviceObject);
    return Status;
}
DbgPrintEx(0, 0, "Shminifilter - registered and started filtering\n");
return Status;
}

```

חשוב לציין שהטור של ה-*Minifilter Drivers* שציינתי הוא תור ממויין, והוא מופיע לפני ערך של כל-הנראה altitude. הערך זהה מצוין בקובץ-hinf של ה-*Driver* וככל שהערך נמוך יותר ה-*Driver* יקרא קודם לכן במעבר הבקשה מהקורא אל הדיסק ומאותר יותר במעבר התשובה מהדיסק

חזרה אל הקורא:

Filter by title

CREATING AN INF FILE

- Load order groups and altitudes
- Allocated altitudes
- Filter altitude request
- > Filter DriverEntry routine
- > Minifilter FilterUnloadCallback routine
- > Preoperation and postoperation callback routines
- > IRP-specific FLT_PARAMETERS
- > Filter driver contexts
- > ECPs in IRP_MJ_CREATE operations
- > Reparse points
- > BypassIO
- Kernel-mode file copy
- > File system placeholders
- Vetoing a bind link

Minifilter load order groups

Windows uses a dedicated set of *load order groups* for file system minifilters and legacy filter drivers that are loaded at system startup. A filter's load order group assignment depends on the filter's type (for example: anti-virus, encryption, etc.).

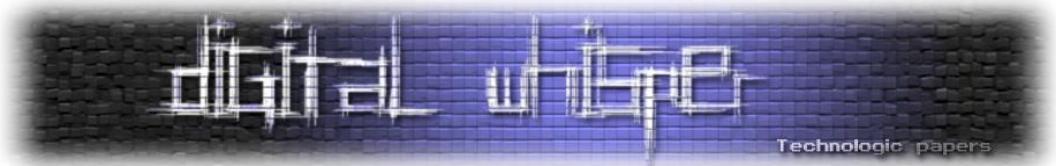
Minifilter altitudes

Each load order group has a defined range of *altitudes*. Every filter driver must have a unique altitude identifier. The filter's altitude defines its position relative to other filter drivers in the I/O stack when that filter is loaded.

The altitude is an infinite-precision string interpreted as a decimal number. A filter driver that has a low numerical altitude is loaded into the I/O stack below a filter driver that has a higher numerical value.

Microsoft must allocate your first altitude value based on filter requirements and load order group. Companies with a Microsoft-assigned "integer" altitude can then *create their own altitudes* within the same load order group.

Altitude values for a filter driver are specified in the **Instance** definitions of the **Strings** Section in the filter driver's INF file. Instance definitions can also be



מבנה Minifilter Dispatch Function

כמו שנition לראות Minifilter יכול לנתר פעולות לפי בקשה שנעשות על מערכת הקבצים בזמן מעבר הבקשה מהקורא אל הדיסק (pre-operation) ובתשובה מהדיסק אל הקורא (Post-operation):

- Pre-operation Dispatch: במקרה זה הפעולה צריכה להיות מוכנה לקבל 3 פרמטרים, מביניהם החשוב ביותר הוא Data.

הפרמטר הזה ניתן להשיג מידע רב על הפעולה כגון שם וסימת הקובץ הרלוונטי, המסלול המלא במערכת הקבצים, גודל הקובץ ועוד עבור קלט\פלט במקרה של כתיבה\קריאה על קבצים.

הפעולה מחייבת סטטוס מטיפוס FLT_PREOP_CALLBACK_STATUS והערכים המשומשים בויתר הם devices FLT_PREOP_SYNCHRONIZE (יצאת מהפעולה הנוכחית ולהמשיך את זרימת הבקשה ל-devices הבאים בתווך) או FLT_PREOP_COMPLETE (יצאת מהפעולה הנוכחית ולחזור ישר לקרוא ללא המשך זרימת הבקשה לממשירים הבאים בתווך).

בדוגמא ניתן לראות dispatch לפעולות JM_CREATE_IRP שמנוע מחיקת קובץ תחת תיקייה אב מוגנת בכך שהוא מזזה את התקינה המוגנת וחזר ישר ברגע שניתן לאתר דגל שנועד למחיקת קבצים עוד לפני שהפעולה עצמה רצתה:

```
FLT_PREOP_CALLBACK_STATUS FLTAPI PreOperationCallbacks::CreateFilterCallback(_Inout_ PFLT_CALLBACK_DATA Data,
    _In_ PCFLT_RELATED_OBJECTS FltObjects,
    _Flt_CompletionContext_Outptr_ PVOID* CompletionContext) {
    UNREFERENCED_PARAMETER(FltObjects);
    UNREFERENCED_PARAMETER(CompletionContext);
    PFLT_FILE_NAME_INFORMATION NameInfo = NULL;
    PFLT_PARAMETERS FilterParameters = NULL;
    PVOID DatabaseEntry = NULL;
    UNICODE_STRING TriggerAccessDeniedParentDir = RTL_CONSTANT_STRING(L"\\\VeryImportantStuffDeleteProtection\\");
    UNICODE_STRING TriggerDeleteBackup = RTL_CONSTANT_STRING(L"\\\VeryImportantStuffBackup\\");
    UNICODE_STRING BackupDirectory = RTL_CONSTANT_STRING(L"\\\DeleteBackupShminiFil1");
    NTSTATUS Status = STATUS_UNSUCCESSFUL;
    UNICODE_STRING FileExtension = { 0 };
    UNICODE_STRING FileName = { 0 };
    FLT_PREOP_CALLBACK_STATUS FilterStatus = FLT_PREOP_SYNCHRONIZE;
    BOOLEAN IsDirectoryDelete = FALSE;

    // Get the file information:
    if (!NT_SUCCESS(FltGetFileNameInformation(Data,
        FLT_FILE_NAME_NORMALIZED
        | FLT_FILE_NAME_QUERY_DEFAULT,
        &NameInfo))) {
        goto FinishLabel;
    }

    // Check if the file is a directory and needs to be deleted
    if (IsDirectoryDelete && NameInfo->Name.Length > 0 && NameInfo->Name.Buffer[0] == '/') {
        // Delete the directory
        Status = DeleteDirectory(TriggerAccessDeniedParentDir, TRUE);
        if (!NT_SUCCESS(Status)) {
            // Log error or handle failure
        }
    } else {
        // Delete the file
        Status = DeleteFile(TriggerDeleteBackup, TRUE);
        if (!NT_SUCCESS(Status)) {
            // Log error or handle failure
        }
    }

    // Set the completion context
    *CompletionContext = FltObjects->CompletionContext;
}
```

```

// Parse the file name from information:
if (!NT_SUCCESS(FltParseFileNameInformation(NameInfo))) {
    goto FinishLabel;
}

// Increment counters for generic create-pre request:
if (!DatabaseCallbacks::IncrementByInformation(Data, NameInfo, CreatePreCount)) {
    DbgPrintEx(0, 0, "Shminifilter preoperation - create-pre operation incrementing failed\n");
}

// Prevent deletion of protected files and backup deleted files:
if (FilterParameters != NULL && FilterParameters->Create.Options & FILE_DELETE_ON_CLOSE) {
    Status = FltIsDirectory(FltObjects->FileObject, FltObjects->Instance, &IsDirectoryDelete);
    if (NT_SUCCESS(Status)) {
        if (IsDirectoryDelete) {
            goto FinishLabel; // Ignore directory deletion
        }
    }
    if (HelperFunctions::IsInParentDirectory(&NameInfo->ParentDir, &TriggerAccessDeniedParentDir)) {
        DatabaseCallbacks::IncrementDetected();
        DbgPrintEx(0, 0, "-- Delete with create is on a file inside a disclosed directory (parent directory = %wZ, disclosed
        &NameInfo->ParentDir, &TriggerAccessDeniedParentDir);
        Data->IoStatus.Status = STATUS_ACCESS_DENIED;
        Data->IoStatus.Information = 0;
        DatabaseEntry = DatabaseCallbacks::CreateDatabaseEntry(Data, NameInfo, "Deleted file is protected, preventing deletion
        \"CREATE PREOPERATION");
        FilterStatus = FLT_PREOP_COMPLETE;
    }
}

else {
    // Create paths for the deleted file and the backup file:
    if (HelperFunctions::IsInParentDirectory(&NameInfo->ParentDir, &TriggerDeleteBackup)) {
        DbgPrintEx(0, 0, "Shminifilter preoperation - Backup parameters: %ws, %ws, %ws, %ws, %ws\n", BackupDirectory.Buf
        NameInfo->ParentDir.Buffer, NameInfo->Name.Buffer, NameInfo->ParentDir.Buffer, NameInfo->Name.Buffer);
        HelperFunctions::CreateBackupOfFile(&BackupDirectory, NameInfo->ParentDir.Buffer, NameInfo->Name.Buffer);
    }
}

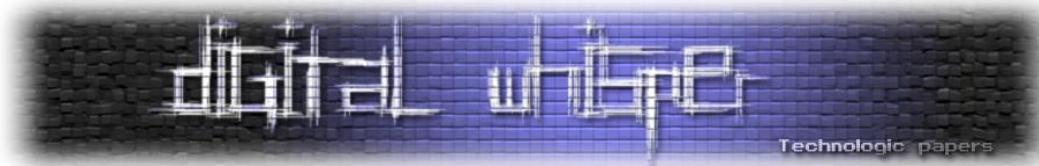
// Add entry to database:
if (DatabaseEntry != NULL) {
    if (!DatabaseCallbacks::AddEntryToDatabase(DatabaseEntry, ((PDETECTED_ENTRY)DatabaseEntry)->EntrySize)) {
        DatabaseCallbacks::DeleteDatabase();
    }
}

FinishLabel:
return FilterStatus;
}

```

- Post-operation Dispatch: הפעולה מצפה לקבל 4 פרמטרים וגם פה Data הוא המרכז והיחיד שבדרכו נרצה להתעוק אליו. הפעולה מחזירה סטטוס מטיפוס FLT_POSTOP_CALLBACK_STATUS וגם לטיפוס זה קיימים מספר ערכי אפשריים, כשהנפוץ הוא FLT_POSTOP_FINISHED_PROCESSING (המעבר על הנתונים והתשובה מהדיסק הסתיים ובסיום הפעולה הנוכחית התשובה מהפעולה תעבור למכשורים הבאים בתור). בדוגמה זו יצרתי dispatch עבור IRP_MJ_READ שמבצע מספר בדיקות ושינויים של המידע שחרזר מהדיסק לאחר הקראיה עצמה:

 - 1) אם מופיע צירוף מיילים מסוים במידע שחרזר מהדיסק (לדוגמא: "The password is:", ה-Driver משנה את כל המידע שקיים אחרי הצירוף ל-...*! כדי להחביא מידע רגיסטר)
 - 2) אם השם של הקובץ מסוים בצוירוף מסוים (לדוגמא: "dirty.txt"), כל המידע שנקרא מהקובץ מוצפן בצורה חד-כוונית (hash algorithm)



(3) אם הקובץ נמצא בתיקית אב מסוימת, כל המידע שנקרא מוחלף בציরוף "CD", לא לחשוף מידע לגורמים ללא הרשות:

```
FLT_POSTOP_CALLBACK_STATUS PostOperationCallbacks::ReadFilterCallback(_Inout_ PFLT_CALLBACK_DATA Data,
    _In_ PCFLT RELATED OBJECTS FltObjects,
    _In_opt_ PVOID CompletionContext,
    _In_ FLT_POST_OPERATION_FLAGS Flags) {
    UNREFERENCED_PARAMETER(FltObjects);
    UNREFERENCED_PARAMETER(CompletionContext);
    UNREFERENCED_PARAMETER(Flags);
    LPSTR TriggerHidingSequence = "The password is:";
    LPSTR AccessDeniedMessage = "ACCESS_DENIED XXX";
    int HidingIndex = -1;
    PVOID* ReadBuffer = NULL;
    PULONG ReadLength = NULL;
    UNICODE_STRING TextExtension = RTL_CONSTANT_STRING(L".txt");
    UNICODE_STRING TriggerObfuscateEnding = RTL_CONSTANT_STRING(L"dirty.txt");
    UNICODE_STRING TriggerAccessDeniedParentDir = RTL_CONSTANT_STRING(L"\VeryImportantStuffPostRead\\");
    UNICODE_STRING FileExtension = { 0 };
    UNICODE_STRING FileName = { 0 };
    PFLT_FILE_NAME_INFORMATION NameInfo = NULL;
    NTSTATUS Status = STATUS_UNSUCCESSFUL;
    BOOL AlreadyPrinted = FALSE;
    LPCSTR DisclosedInformation = "Content holds disclosed information ";
    LPCSTR EncryptedFile = "File matches needed suffix for irreversible encryption ";
    LPCSTR ParentDirectory = "Parent directory is disclosed, erasing read information ";
    char MultipleSpecial[1024] = { 0 };
    PVOID DatabaseEntry = NULL;

    // Get the file information:
    Status = FltGetFileNameInformation(Data,
        FLT_FILE_NAME_NORMALIZED
        | FLT_FILE_NAME_QUERY_DEFAULT
        &NameInfo);
    if (!NT_SUCCESS(Status)) {
        if (Status != STATUS_FLT_INVALID_NAME_REQUEST) {
            DbgPrintEx(0, 0, "Shminifilter postoperation - read operation stopped, FltGetFileNameInformation() returned 0x%x\n",
                Status);
        }
        return FLT_POSTOP_FINISHED_PROCESSING;
    }

    // Parse the file name from information:
    Status = FltParseFileNameInformation(NameInfo);
    if (!NT_SUCCESS(Status)) {
        if (Status != STATUS_FLT_INVALID_NAME_REQUEST) {
            DbgPrintEx(0, 0, "Shminifilter postoperation - read operation stopped, FltParseFileNameInformation() returned 0x%x (%
                Status, &NameInfo->Name);
        }
        return FLT_POSTOP_FINISHED_PROCESSING;
    }

    // Output information about the read data from the file and other attributes:
    FltDecodeParameters(Data, NULL, &ReadBuffer, &ReadLength, NULL);
    FltParseFileName(&Data->Iopb->TargetFileObject->FileName, &FileExtension, NULL, &FileName);

    // Change data from \..txt read to something else:
    if (RtlCompareUnicodeString(&FileExtension, &TextExtension, TRUE) == 0) {
```

```
// Hide content after certain sequences of words:  
HidingIndex = HelperFunctions::DoesContain((LPSTR)Data->Iopb->Parameters.Read.ReadBuffer, TriggerHidingSequence, TRUE);  
if (HidingIndex != -1) {  
    DatabaseCallbacks::IncrementDetected();  
    HelperFunctions::PrintFileInfo(Data, NameInfo, &FileName, &FileExtension, *ReadLength);  
    DbgPrintEx(0, 0, "-- File content contains hiding trigger (%s, index %d), hiding read content ...\\n",  
        TriggerHidingSequence, HidingIndex);  
    AlreadyPrinted = TRUE;  
    FltLockUserBuffer(Data);  
    HelperFunctions::HidefileContent((LPSTR)Data->Iopb->Parameters.Read.ReadBuffer, HidingIndex, TriggerHidingSequence);  
    FltSetCallbackDataDirty(Data);  
    strcat_s(MultipleSpecial, DisclosedInformation);  
}  
  
// Irreversingly obfuscate files with certain name suffixes:  
if (HelperFunctions::EndsWith(FileName.Buffer, TriggerObfuscateEnding.Buffer)) {  
    DatabaseCallbacks::IncrementDetected();  
    if (!AlreadyPrinted) {  
        HelperFunctions::PrintFileInfo(Data, NameInfo, &FileName, &FileExtension, *ReadLength);  
        AlreadyPrinted = TRUE;  
    }  
    DbgPrintEx(0, 0, "-- File name ends with obfuscate trigger (%wZ), obfuscating read content ...\\n", &TriggerObfuscate  
    FltLockUserBuffer(Data);  
    HelperFunctions::ObfuscateFileContent((LPSTR)Data->Iopb->Parameters.Read.ReadBuffer);  
    FltSetCallbackDataDirty(Data);  
    strcat_s(MultipleSpecial, EncryptedFile);  
}  
}  
  
// Make sure that access to file/folder is permitted:  
if (HelperFunctions::IsInParentDirectory(&NameInfo->ParentDir, &TriggerAccessDeniedParentDir)) {  
    DatabaseCallbacks::IncrementDetected();  
    if (!AlreadyPrinted) {  
        HelperFunctions::PrintFileInfo(Data, NameInfo, &FileName, &FileExtension, *ReadLength);  
        AlreadyPrinted = TRUE;  
    }  
    DbgPrintEx(0, 0, "-- Read was on a file inside a disclosed directory (parent directory = %wZ, disclosed directory = %wZ  
        &NameInfo->ParentDir, &TriggerAccessDeniedParentDir);  
    FltLockUserBuffer(Data);  
    for (ULONG FileIndex = 0; FileIndex < *ReadLength; FileIndex++) {  
        RtlCopyMemory((PVOID)((ULONG64)Data->Iopb->Parameters.Read.ReadBuffer + FileIndex),  
            (PVOID)((ULONG64)AccessDeniedMessage + (FileIndex % strlen(AccessDeniedMessage))), 1);  
    }  
    FltSetCallbackDataDirty(Data);  
    strcat_s(MultipleSpecial, ParentDirectory);  
}  
  
// Increment counters for generic read-post request:  
if (!DatabaseCallbacks::IncrementByInformation(Data, NameInfo, ReadPostCount)) {  
    DbgPrintEx(0, 0, "Shminifilter postoperation - read-post operation incrementing failed\\n");  
}  
  
// Create database entry and add it to the current database:  
if (strlen(MultipleSpecial) != 0) {  
    DatabaseEntry = DatabaseCallbacks::CreateDatabaseEntry(Data, NameInfo, MultipleSpecial,  
        "READ POSTOPERATION");  
    if (DatabaseEntry != NULL) {  
        if (!DatabaseCallbacks::AddEntryToDatabase(DatabaseEntry, ((PDETECTED_ENTRY)DatabaseEntry)->EntrySize)) {  
            DatabaseCallbacks::DeleteDatabase();  
        }  
    }  
}  
return FLT_POSTOP_FINISHED_PROCESSING;
```

הערה - הפעולות:

- FltParseFileNameInformation()
- FltGetFileNameInformation()
- FltParseFileName()
- FltDecodeParameters()
- ויעד

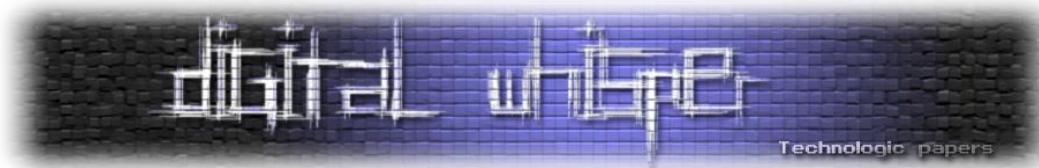
מפרשות את המידע בתוך הפורמטר Data 7-structs נפרדים שייהי יותר קל לעבוד עליהם. בדוגמה אני מתעד כל אירוע חשוב כמו ניסיון מחיקת קובץ בתיקייה שומרה ועוד ב-database קרנלי ומוחרם יותר אראה כיצד אני מעביר אותו לתוכנה הריצה ב-usermode ומראה במשק משתמש ברור את כל האירועים שקרו

כל הקוד של ה-Driver לדוגמה מקשור בסוף המאמר למי שרוצה להעמיק עוד.

מבנה DeviceControl Dispatch Function

אחרי כל ההסבירים הדורשים עברו פעולת ה-ioctl DeviceControl של הדוגמא שיצרתי, עברו עכשו על הפעולה, על המבנה הבסיסי שלו ועל התוספות למבנה זהה שהוספה עבור המטריה שלו. טיפה לפני שמתחילה עם הפרטיקה, אסביר חלק מהתאוריה החדשנית כדי להבין את הפעולה ואת המבנה שלה. על התאוריה זו עברתי גם במאמרים קודמים (במיוחד באחרוני) וכן כדי להתעמק ניתן לקרוא גם אותם:
(1) Passing Method: יש כמה צורות בהן ניתן להעביר את הfrmटרים עבור הבקשה מסוימת ל-Device, והן פשוט מתחבאות במיקום בתוך מבנה ה-IRP שבו כל רכיב קלט\פלט נמצא, לדוגמה באפר הקלט או הגודל של באפר הפלט. האפשרות הדיפולטיבית היא METHOD_BUFFERED ולן בಗיל שכמה עט כל הקוראים לפעולה () או פועלות מקבילות לה משתמשות ב-METHOD_BUFFERED, אין צורך הכרחי לטפל בצורה מיוחדת גם בשאר צורות ההעברה ונitin לשער ש-METHOD_BUFFERED היא צורת העברה שמצומת בבקשת.

(2) IOCTL Code: בנוסף ל-major function code שבמקרה זה הוא MJ_DEVICE_CONTROL, עבר עוד קוד משמעוני כחלק ממבנה ה-IRP שהוא ConTroL code-Input Output. הקוד זה הוא מספר בגודל 4 בתים והוא מצין פעולה מסוימת שה-device יכול לטפל בה. בפועל בתחום הפעולה נצפה לראות () עם ה-IOCTL code שטפל בכל אחת מהאפשרויות שבהם המכשיר תומך. במקרה STATUS_INVALID_DEVICE_REQUEST בו: ה-Driver ייחזיר/device תומך לא קוד שמה לא שולח הוא והוא יבצע שום פעולה. כמובן שבמקרה זה חייב להיות תיאום בין הרכיבים הקיימים ב-device, ובמקרה והפעולה היא פעולה יותר מורכבת הכוללת device stack של יותר מ-1 שרטים ב-usermode, בטבלה יוצג(device-stack) ה.device הולך וגדל בהתאם ל-IRP. אם מופיע אחד - ה-device שולנו יצטרך להעביר את ה-IRP ל-device הבא בתחום בסוף הפעולה גם אם לטפל בה כבר ועם ה-IOCTL code לא נתמך על ידו (במקרה זה ה-Driver לא ייחזיר את הסטטוס המוחדר ב-device שמייחד בstack).



(3) BytesReturned: בסוף כל request מטופל מצופה על ידי ה-Driver להכניס את הערך שמייצג את מספר הבטים שעברו מניפולציה בעקבות הפעולה לשדה ספציפי בטור מבנה ה-IRP. שדה זה יחוור לטור הפעמטר BytesReturned שמסופק לפעולות כמו ()DeviceControl.

עכשו אכנס לקוד של הדוגמא שלי. הדוגמא במקורה זה מטפלת רק בסוג אחד של בקשה המיוצגת על ידי IOCTL code 0x40002000 (או INFOPASS_IOCTL), והוא נשלחת כל כמות זמן מסוימת מהתוכנה המבוססת ב-usermode שנועדה להציג את כל האירועים שעברו ניטור על ידי ה-Driver. ה-Driver אוגר database פנימי בזיכרון את המידע על כל האירועים ובשליחת IOCTL_INFOPASS כל ה-INFOPASS database מועתק על ידי פעולה -(DeviceControl) לזריכון של תוכנת הניטור. ברגע ההעתקה ה-DeviceControl מוחדר כדי לא להشير תיעוד של אירועים שנוטרו כבר לפני כן:

```
NTSTATUS IoctlCallbacks::DeviceControlCallback(PDEVICE_OBJECT DeviceObject,
PIRP Irp) {
/*
Assumes input/output buffers are the same
input: PID value of calling process to inject info into (8 bytes) + dummy address (8 bytes)
output: database UM buffer (8 bytes) + database size (8 bytes)
*/
PIO_STACK_LOCATION ParamStackLocation = NULL;
NTSTATUS Status = STATUS_SUCCESS;
ULONG InputBufferSize = 0;
ULONG OutputBufferSize = 0;
PUCHAR InputBuffer = NULL;
PUCHAR OutputBuffer = NULL;
PVOID Database = NULL;
ULONG64 DatabaseSize = 0;
ULONG64 CallerPID = 0;
PEPROCESS CallerProcess = NULL;
KAPC_STATE CallerContext = { 0 };
PVOID AllocatedUserBuffer = NULL;
UNREFERENCED_PARAMETER(DeviceObject);
PAGED_CODE();
ParamStackLocation = IoGetCurrentIrpStackLocation(Irp);
InputBufferSize = ParamStackLocation->Parameters.DeviceIoControl.InputBufferLength;
OutputBufferSize = ParamStackLocation->Parameters.DeviceIoControl.OutputBufferLength;
```

```
// Determine which I/O control code was specified:
switch (ParamStackLocation->Parameters.DeviceIoControl.IoControlCode) {
case INFOPASS_IOCTL:

    // Verify correct parameters:
    InputBufferSize = ParamStackLocation->Parameters.DeviceIoControl.InputBufferLength;
    OutputBufferSize = ParamStackLocation->Parameters.DeviceIoControl.OutputBufferLength;
    InputBuffer = (PUCHAR)Irp->AssociatedIrp.SystemBuffer;
    OutputBuffer = (PUCHAR)Irp->AssociatedIrp.SystemBuffer;
    if (InputBufferSize != 16 || OutputBufferSize != 16 || InputBuffer == NULL) {
        Status = STATUS_INVALID_PARAMETER;
        break;
    }
    RtlCopyMemory(&CallerPID, InputBuffer, sizeof(ULONG64));
    RtlCopyMemory(&AllocatedUserBuffer, &InputBuffer[sizeof(ULONG64)], sizeof(PVOID));
    if (CallerPID == 0 || CallerPID >= 0xFFFF || AllocatedUserBuffer == NULL) {
        Status = STATUS_INVALID_PARAMETER;
        break;
    }
    DatabaseCallbacks::GetDatabase(&Database, &DatabaseSize);
```

הפעולה המיוחדת מוכנה לקבל באפר קלט בגודל של 16 בתים, מהם 8 הבטים הראשונים הם ערך ה-ID של התהיליך המנתר ו-8 הבטים הבאים הם ערך של כתובות דוגמא מהתוכנה (לא הכרחי אבל בכלל שהבאפר הכלול יהיה בגודל 16 בתים נוסף משחו לקלט). גם הפלט הוא 16 בתים שמכיל את הכתובת בזיכרון של תוכנת הניתור בה ימצא ה-base database שהוותקן לזכרון והגודל של ה-base database בזיכרון:

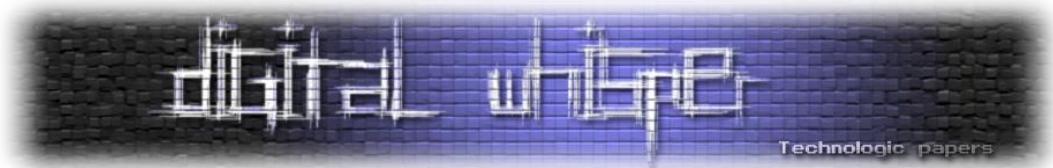
```
// Destroy current database and unlock extracting:  
DatabaseCallbacks::DeleteDatabase();  
DatabaseCallbacks::InitiateDatabase(); // Re-initialize the database with the initial entry  
DatabaseCallbacks::UnlockExtracting();  
Irp->IoStatus.Information = 16;  
DbgPrintEx(0, 0, "Shminifilter IOCTL - Passed the database and re-initiated basic information\n");  
break;  
  
default:  
    Status = STATUS_INVALID_DEVICE_REQUEST;  
    break;  
}  
Irp->IoStatus.Status = Status;  
IoCompleteRequest(Irp, IO_NO_INCREMENT);  
return Status;  
}
```

בדוגמא ביצעת הרצה בדיקות, כמו בדיקה שה-ID לא אפס או ערך לא ואלייד אחר ובדיקות שהקלט והפלט בגודל המתאים ושבאפר קלט-פלט קיימ בכלל חלק מהבקשה. בתהיליך העתקה אני רץ בكونטיקט של התהיליך של תוכנת הניתור בעזרת קונספט הנקרא APC (אסביר אותו בהמשך המאמר), אני מקצת מספיק זיכרון בתור מרחב הכתובות של התהיליך, מעתיק את ה-base database מה-Driver ומএפס את ה-base database הקויי.

ניתן לראות בדוגמה נקודות נוספות בפורמט הכתיבה של פעולה DeviceControl מסודר ונכוון:

- החזרת ה-NTSTATUS של הפעולה גם לתוך השדה המצופה בתוך ה-IRP ולא רק בתור הערך המוחזר מהפעולה
- תלות ממש ברורה ומופרدة בין כל בדיקה לכל אחרת, הרי לבדוק באותו תנאי האם הערך מתוך באפר הקלט ואלייד ורק אז לבדוק אם בכלל הבאפר קיימ יגרום לשגיאה קריטית שתפגע בפעולות כל המערכת במקרה והבאפר קלט לא קיימ או לא גדול מספיק עבור הנתונים
- שימוש ב-GetCurrentIrpStackLocation(): הפעולה זו שולפת את מבנה ה-STACK_LOCATION מຕוך ה-IRP של הבקשה. מבנה זה מכיל את כל הפרמטרים שחשובים לדriver לבדוק, כגון הbuffרים של הקלט והפלט והגדלים של כל באפר, ובמוקם להשתמש במסלול המלא של כל ערך הפעולה זו מעתיקה לנו את המבנה מຕוך ה-IRP אל מבנה נפרד
- שימוש ב-CompleteRequest() כדי לסמל שהבקשהטופלה ב-driver שלנו וההתעסוקות עם הבקשה הסתיימה ברגע זה. אחרי סיום הפעולה התשובה תחזיר ישר לקורא ולא תמשיך לשום device אחר

از לאחר הצגת אבני הבניין של פיתוח קרנלי עבור מטרות שונות וכליות, אני אציג חלק ממשמעוני מהפעולות היכי נפוצות או שימושיות לפני מקרה שיוכלו להיות שימושיות בעת צורך מסוים בכתיבת ה-driver.



פעולות שימושיות למטרות שונות ב-Driver

יש הרבה סוגי של Driver-ים עם הרבה מטרות וצריכים שונים ומגוונים, ובמקרים לרכז את השימוש של כל שירות המערכת וה-API הkernel'י בתוך ntoskrnl יש פיצול של כל השירותים לחלקים שונים לפי קטגוריות ומטרת שימוש. את ההצהרה והגישה לכל הפעולות הללו נשים דרך headers של הkernel' שמאופיעים גם ב-MSDN עבור כל פעולה שנוצרה להשתמש בה, והשימוש יבוצע על ידי החלק בkernel' בין אם בתוך ntoskrnl ובין אם ב-Driver נפרד) שאחראי לטיפול באוטו סוג של בקשה.

השם של פעולה קרנלית בדרך כלל נראה כך: `Prefix + Operation + Input/Output for operation` + ZwQueryInformationFile - Purpose of operation"

- (1) ה-prefix של הפעולה (Zw) אומר שהפעולה ממומשת בתוך ntoskrnl.exe ושהפעולה היא system service שנייה להשתמש בו באמצעות הפקודה `syscall`
- (2) הפעולה במקרה זה היא query - לחפש ולהציג מידע מסוים על אובייקט במערכת. הערך זהה יכול להיות מגוון פעולות כמו Set/Get
- (3) המטרה במקרה זה היא להשיג מידע על קובץ ולן הקולט יהיה איזשהו מאפיין מיוחד של הקובץ (כגון HANDLE)

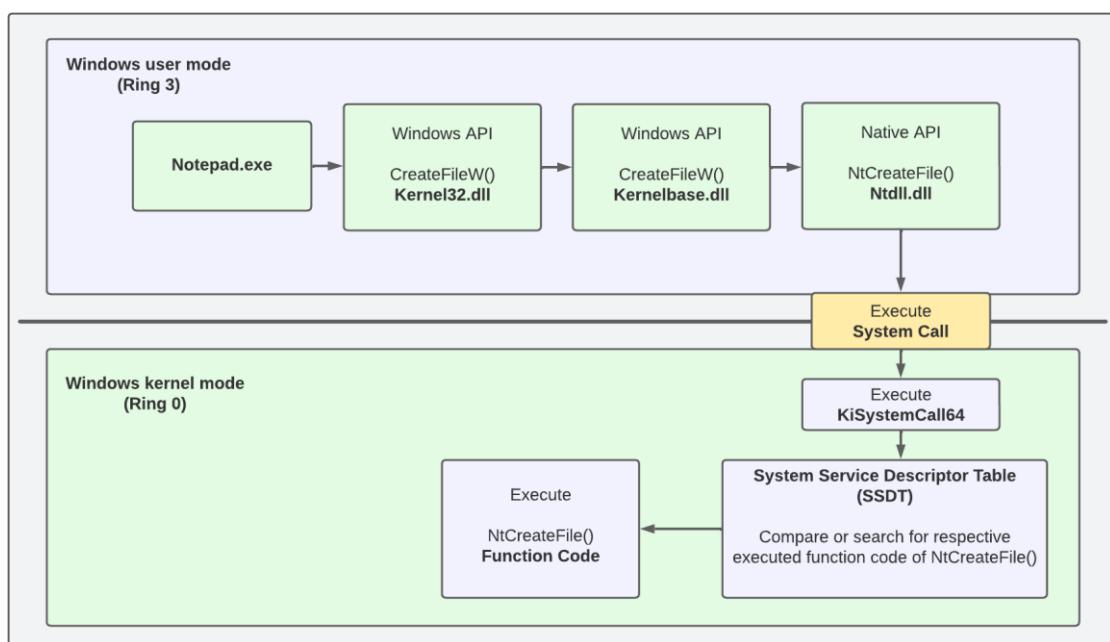
עכשו עבור על הקבוצות המרכזיות שניתן לזהות לפ' prefix שונה של הפעולה, ואסביר איזה חלק במערכת הפעלה ממש אותו ומה הקטגוריה של בקשות שאותה קבוצה מטפלת בה:

:ZW/NT

כפי שתיארתי בתחילת המאמר, פעולות אלו הם system services שממומשות על ידי ntoskrnl.exe ונitin להציג גישה אליהן באמצעות הפקודה `syscalls`. בעת ביצוע פקודות syscall עם המספר המזהה של הפעולה הרצוייה, יבוצע מעבר ריצה מ-usermode ל-kernelmode system call handler יהיה הרכיב שידאג שהפעולה תבוצע כשרה.

הפעולות הkernel'ית מאוד פשוטה: מספר ה-`syscall` מאוחסן ברגיסטר `rax`, וה-`handler` פשוט מעביר את הפקטרים שהגיעו עבור ה-`service` system service אל ה-`service` system service שנמצא באינדקס של ערך `rax` מתחילה ה-`SSDT` system service descriptor table.

כפי שתיארתי הפעולות מקבוצה זו יכולות לשרת הרבה מטרות, כגון השגת מידע/קריה/כתיבת/מניפולציה על הבאים. גישה למשאים הקשורים לתהליכיים רצויים ועדן:



The figure shows the transition from Windows user mode to kernel mode in the context of saving a file within notepad.exe.

:FLT

על הקבוצה זו עברתי כבר בפירוט בחלק על ה-*Minifilter Drivers*, אך קבוצת פעולות זו משרתת את כל ה-*Drivers* שקיימים על המערכת. הפעולות ממומשות על ידי ה-*filter manager* שנמצא בחלק נפרד ממערכת הפעלה המרכזיית בשם "fltmgr.sys". הפעולות מוצחרות ב-".h" וונעדו לעזור לפרש פרמטרים הנשלחים ל-*Driver* מבקשות שונות הקשורות למערכת הקבצים כדי לאפשר פעולות מתאימה רוגוש לאחות וטוניין.

בנוגע לאותם נתוניים:

master ▾ [winsdk-10](#) / [Include](#) / [10.0.14393.0](#) / [km](#) / [fltKernel.h](#) ↗

Import 10.0.14393.0.

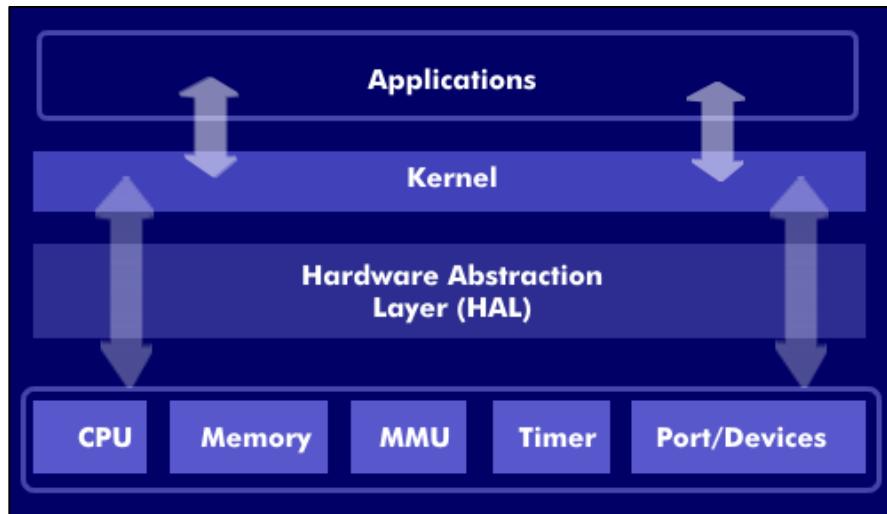
Blame Executable File · 5902 lines (4793 loc) · 153 KB

Raw [Copy](#) [Download](#) [Edit](#) ▾

```
/*++  
Copyright (c) 1989-2002 Microsoft Corporation  
Module Name:  
    fltKernel.h  
Abstract:  
    This contains all of the global definitions for mini-filters.  
Environment:  
    Kernel mode  
--*/
```

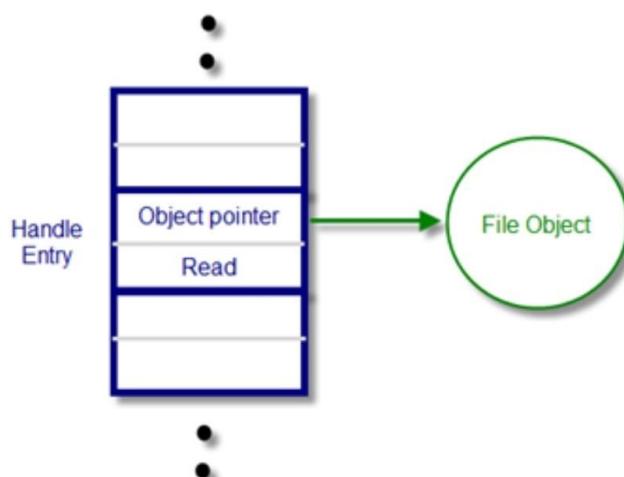
:HAL

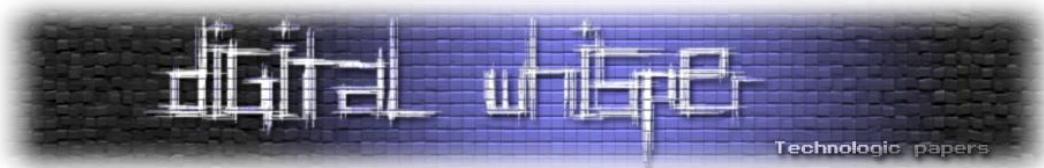
קובוצה זו לא בשימוש תדייר ברוב ה-Drivers ו ורק במקרים ספציפיים יהיה צורך בה. Hardware Abstraction Layer היא שכבה תוכנה שמומומשת ברכיב sys.HAL ונונתנת משיק הרבה יותר נוח עבור מפתחים לגישה ישירה לחומרה ספציפית. במקרה שהמתכונת יצטרך להבין את פרוטוקול התקשרות עבור כל רכיב חומרה שקיים במערכת ולהתychס לכל מקרה קצה הכרחי, שכבת ה- HAL נונתנת לנו API נוח עבור גישה לכל רכיב חומרה:



:OB

בנוסף למבנים הבסיסיים במערכת הפעלה כמו רשימת ה-EPROCESS, קיימת גם רשימה כללית של אובייקטים עבור אובייקטים כמו תהליכיים, תהיליכונים וקבצים. בגלל שימוש מערכת הפעלה צריכה להשתמש בהרבה אובייקטים שונים בזמן ריצתה, היא מנהלת רשימה של kernel objects עבור כל אובייקט קצה ומattaרת את כל התוכנות שלו ואת המידע שצරיך לדעת עליו. רשימה זו קוראים ObjectTable והרשומה שהיא שומרת עבור כל אובייקט נקראת HANDLE_TABLE_ENTRY:





זה ההסבר עליה:

nt!_HANDLE_TABLE_ENTRY

Processes in Windows have their own private handle table which is stored in the kernel virtual address space. HANDLE_TABLE_ENTRY represents an individual entry in the process's handle table. Handle tables are allocated from Paged Pool. When a process terminates the function ExSweepHandleTable() closes all handles in the handle table of that process. The **Object** field points to the object structure i.e. File, Key, Event etc., for which the handle has been created. The **GrantedAccess** field is a bitmask of type ACCESS_MASK which determines the set of operations that the particular handle permits on the object. The value of this field is computed by SeAccessCheck() based on the access requested by the caller (Desired Access) and the access allowed by the ACEs in the DACL in the security descriptor of the object. The debugger's "!handle" command can be used to examine the handle table of any process. The "!htrace" command can be used to display stack trace information about handles, if handle tracing is enabled.

API : ObReferenceObjectByHandle(), ObReferenceObjectByHandleWithTag().

[מקור: https://codemachine.com/articles/kernel_structures.html]

אקשר בסוף המאמר מאמר מעולה של עמית מושל שמתאר את הנושא בהקשר של תהליכי, אבל בעצם הדבר שחייב להבין מההסבר הזה הוא שהמבנה הזה עוזר למפות בין handles מובוסי handles שאפליקציות משתמשות בהם לבין האובייקטים הפנימיים שבפועל משומשים בקורסן.

ב-Driver נוכל לבצע אינטראקציה עם ה-ObjectTable באמצעות קבוצת הפעולות עם ה-prefix של "Ob.." כגון השגת המידע על האובייקט הרלוונטי באמצעות הפעולות ObReferenceObjectByName/Handle/Pointer. חשוב לזכור שהמבנה ייחזר יייחדely מוקהה למקורה, לדוגמא אובייקט קובץ ייחזר מבנה FILE_OBJECT עם המידע הרלוונטי:

```
231 PDRIVER_OBJECT
232     general_helpers::GetDriverObjectADD
233     (PUNICODE_STRING DriverName) {
234         PDRIVER_OBJECT DriverObject = NULL;
235         if (!NT_SUCCESS(
236             ObReferenceObjectByName
237             (DriverName, OBJ_CASE_INSENSITIVE, NULL, 0,
238                 *IoDriverObjectType, KernelMode, NULL, (PVOID*)&DriverObject)) ||
239             DriverObject == NULL) {
240             return NULL;
241         }
242         return DriverObject;
243     }
```

:PS

בשונה מקבצת הפעולות זו שמתבססת על השגת מידע מהמיופיע של usermode handles לאותו קיטים הפנימיים של המערכת מתוך ה-ObjectTable, קיים גם מבנה נוסף המכיל את האובייקטים הkernelים עצם PspCidTable לפि הרפרנציה האמיתית שלהם עבור מערכת הפעלה. מבנה נתונים זה נקרא (process) (process specific client identifier) והוא מרכז בתוכו רשימה של אובייקטים kernelים עבור משאים כמו תהליכיים, תהליכונים ועוד. למבנה זה אין קשר למה שקרה ב-usermode, ואם לדוגמא גנסה לחפש במבנה בעזרת ObjectTable handle.

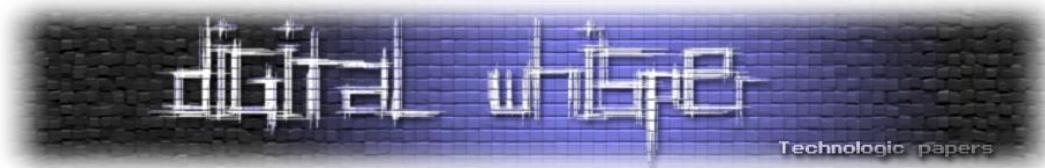
במבנה זה האובייקטים מצוינים על ידי מזהה מיוחד בל' חשיבות לסוג האובייקט ונitin לחפש במבנה זה בעזרת פועלות כמו PsLookupProcessByProcessId שתעביר על כל אובייקט ברשימה ותחזיר את ה-EPROCESS של התהיל' עם ה-ID המתאים:

```
← ...iver/requests.cpp ⌂ ⚙ :
```

```

31
32     // Check for invalid arguments:
33     if (RootkInst->MainPID == 0) {
34         DbgPrintEx(0, 0, "KMDFdrive
35 Requests - Get base address of executable
failed (invalid parameter: MainPID = 0)\n"
36 );
37             return
38
39     // Process EPROCESS:
40     if (!NT_SUCCESS(
41         PsLookupProcessByProcessId
42         ((HANDLE)RootkInst->MainPID, &Process))) {
43             DbgPrintEx(0, 0,
44 KMDFdriver Requests - Get base address of
executable failed (cannot get EPROCESS of
executable %llu)\n", RootkInst->MainPID
45 );
46             return
47         requests_helpers::ExitRootkitRequestADD(NULL
48         , NULL, ROOTKSTATUS_PROCESSEPRC, STATUS_UNSU
49 CCESSFUL, RootkInst);
50     }

```



זה ההסבר עליון

The PspCidTable

The PspCidTable is a "handle table for process and thread client IDs"[7]. Every process' PID corresponds to its location in the PspCidTable. The PspCidTable is a pointer to a HANDLE_TABLE structure.

```
typedef struct _HANDLE_TABLE {
    PVOID          p_hTable;
    PEPROCESS      QuotaProcess;
    PVOID          UniqueProcessId;
    EX_PUSH_LOCK   HandleTableLock [4];
    LIST_ENTRY     HandleTableList;
    EX_PUSH_LOCK   HandleContentionEvent;
    PHANDLE_TRACE_DEBUG_INFO DebugInfo;
    DWORD          ExtraInfoPages;
    DWORD          FirstFree;
    DWORD          LastFree;
    DWORD          NextHandleNeedingPool;
    DWORD          HandleCount;
    DWORD          Flags;
}
```

Windows offers a variety of non-exported functions to manipulate and retrieve information from the PspCidTable. These include:

ExCreateHandleTable

creates non-process handle tables. The objects within all handle tables except the PspCidTable are pointers to object headers and not the address of the objects themselves.

ExDupHandleTable

is called when spawning a process.

ExSweepHandleTable

is used for process rundown.

ExDestroyHandleTable

is called when a process is exiting.

[מקור: <http://uninformed.org/index.cgi?v=3&a=7&p=6>]

:RTL

קובצת הפעולות זו מעדת עבור פעולות קצרות שימוש ככלי שיכל להיות בהן שימוש נפוץ, כגון פעולות להעתיק זיכרון כמו RtlCopyMemory או פעולות הקשורות לניהול מחוזות בקורסן. פעולות אלו הן בעצם utilities שהקורסן מספק לנו במקום שנוצרך למשב עצמנו:

Most of the run-time library (RTL) routines are prefixed with the letters "Rtl"; for a list of the run-time library routines for the kernel, see Run-Time Library (RTL) Routines.

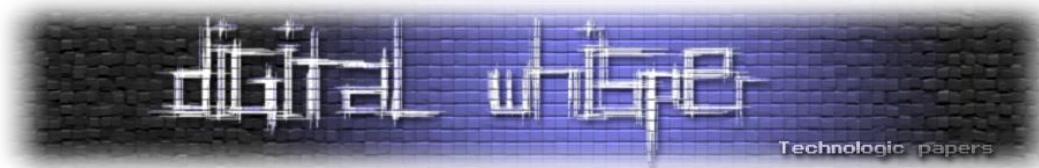
There is also a different kernel-mode library specifically designed for safe string handling. For more information about the safe string library, see Windows Kernel-Mode Safe String Library. Note that safe string library routines are also usually prefixed by "Rtl" but are not part of the run-time library (RTL).

[מקור: <https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/windows-kernel-mode-run-time-library>]

:MM

הקובצת זו מנהלת על ידי memory manager במערכת ההפעלה שמומש ב-`ntoskrnl.exe` עצמו. הפעולות מקובצת זו כוללות כל מה שקשרו לדיזרן, כגון העתקת מידע ממוקם למקום אחר, הקזאה של זיכרון, מיפוי זיכרון מסוים לסוג זיכרון אחר, נעילת מרחב זיכרון מסוים בזיכרון הפיזי של המערכת, האפשריות מאוד מגוונות:

1. `MmAllocateContiguousMemory`
2. `MmAllocateContiguousMemorySpecifyCache`
3. `MmAllocateMappingAddress`
4. `MmAllocateNonCachedMemory`
5. `MmAllocatePagesForMdl`
6. `MmBuildMdlForNonPagedPool`
7. `MmCreateMdl`
8. `MmFreeContiguousMemory`
9. `MmFreeMappingAddress`
10. `MmFreeNonCachedMemory`
11. `MmFreePagesFromMdl`
12. `MmGetPhysicalAddress`
13. `MmGetSystemRoutineAddress`
14. `MmIsValidAddress`



:KE

קברצת kernel executive היא קבוצת פעולות הממומשת ב-`ntoskrnl.exe` ומשמשות להובנה מטרות בסיסיות שהקרナル צריך למשך, כגון סינכרונייזציה, ניהול תהליכיים ותהליכיונים וניהול interrupts. למרות השימוש ה-“נמור” יותר בפעולות הללו, יש בהם שימוש נפוץ גם עבור Driver רגיל ללא שום מטרה מיוחדת:

The `Ke` prefix in Windows kernel functions stands for "Kernel." Functions with this prefix are part of the Kernel Executive, which is responsible for low-level, fundamental operations within the Windows operating system. These operations include synchronization, interrupt handling, thread scheduling, and timing.

:EX

בדומה לקברצת הפעולות RTL, גם קברצת זו נועדה עבור שימוש כפעולות עזר עבור קוד קרנלי שצורך להציג מטרה מסוימת. הפעולות מוממשות בתוך `ntoskrnl.exe` והקברצת אחראית להרבה דברים כגון הקצאת זיכרון, סינכרונייזציה בין חלקיים שונים ושליטה על גישה למשאים:

**Memory Management (e.g.,
`ExAllocatePool`, `ExFreePool`)**

- Purpose:** These functions manage dynamic memory allocation within the kernel. They provide mechanisms to allocate and free memory from different types of memory pools (e.g., non-paged pool, paged pool).

**Synchronization (e.g.,
`ExAcquireFastMutex`,
`ExReleaseFastMutex`)**

- Purpose:** These functions are used for synchronization within the kernel. They provide mechanisms to acquire and release mutexes and other synchronization primitives to ensure safe access to shared resources.

**Resource Management (e.g.,
`ExInitializeResourceLite`,
`ExAcquireResourceExclusiveLite`)**

- Purpose:** These functions manage executive resources, which are used for synchronization and to protect shared resources, providing both exclusive and shared access mechanisms.

**Exception Handling (e.g.,
`ExRaiseStatus`,
`ExSystemExceptionFilter`)**

- Purpose:** These functions handle exceptions and provide mechanisms for structured exception handling within the kernel.

כפי שניתן להבין, כל הפעולות מקובצה זו מモמשות ב-I/O manager של ה kernel והמטרה בשימוש בפעולת קשורה בהכרח להכנת נתונים עבור קלט/פלט או לביצוע I/O מסוים, אם מכיוון ישיר וביצוע ה-I/O בתוך הפעולה עצמה ואם זה לגרום ל פעולה אחרת לבצע את הקלט/פלט בעצמו:

IoCreateDevice, IoDeleteDevice)

- Purpose:** These functions are used to create and manage device objects, which represent physical or logical devices in the system. They allow drivers to create device objects that can be used to interact with hardware or software components.

Driver Management (e.g.,**IoCreateDriver, IoDeleteDriver)**

- Purpose:** These functions manage driver objects, which represent the drivers that handle I/O operations for devices. They enable the creation, initialization, and deletion of drivers.

File Operations (e.g., IoCreateFile,**IoDeleteFile)**

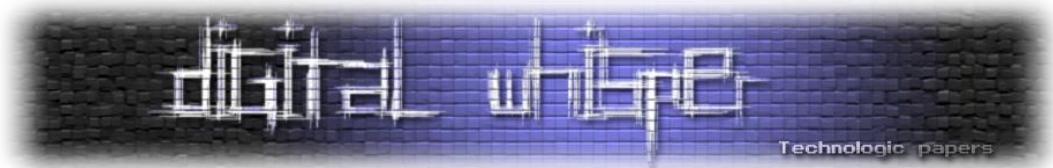
- Purpose:** These functions manage file objects and file operations, such as creating, opening, closing, and deleting files. They provide mechanisms for interacting with the file system.

Completion Routines (e.g.,**IoSetCompletionRoutine,****IoCompleteRequest)**

- Purpose:** These functions manage completion routines, which are callback functions that are called when an I/O operation completes. They allow drivers to specify actions to take when an I/O operation is finished.

از אחרי שעברנו על קבוצות ברורות של פעולות לפי מטרה ומיומש, אפשר מעת על פעולות נוספות שחווב להבין מאותן קבוצות ומקבוצות אחרות. חשוב לומר שאני אכסה פעולות הכרחיות שלא ברורות מלאו, זאת

אומרת שאת הפעולה NtCreateFile לדוגמא אני לא אציג במיוחד



עוד פעולות וモשגים שימושיים לתהיליך הפיתוח

:DbgPrint/Ex

הפעולה הזו היא המקבילה ה-*c*י מתאימה לפעולה המיעדרת לשילוח פלט לגורם מסוים (*printf* בס', *print* בפייטון וcdcומה). בגלל שהוא-*Driver* הוא תוכנה קרנלית ואין לו ממשק גרפי שנייתן לקבל ממנו פלט, הדרך היחידה שלנו לשילוח איזשהו סוג של פלט היא לשילוח ל-*kernel debugger* שמחובר למערכת את הפלט. הפלט זה יופיע בכלים כמו *windbg* או *DbgView* שעל שניהם כבר דיברתי בתחילת המאמר.

הפעולות מקבלות כמה פרמטרים, ומתוכם גם *format string* בדומה ל-*printf* ואת הפרמטרים המתאימים להוסיף לפלט:

The **DbgPrint** routine sends a message to the kernel debugger when the conditions that you specify apply (see the Remarks section below).

Syntax

```
C++  
ULONG DbgPrint(  
    PCSTR Format,  
    ...  
) ;
```

Copy

Parameters

Format

Specifies a pointer to the format string to print. The *Format* string supports most of the *printf*-style [format specification fields](#). However, the Unicode format codes (%C, %S, %lc, %ls, %wc, %ws, and %wZ) can only be used with IRQL = PASSIVE_LEVEL. The **DbgPrint** routine does not support any of the floating point types (%f, %e, %E, %g, %G, %a, or %A).

...

Specifies arguments for the format string, as in *printf*.

Return value

If successful, **DbgPrint** returns the NTSTATUS code STATUS_SUCCESS; otherwise it returns the appropriate error code.

ההבדל בין 2 הגרסאות פשוט מאד:

- **DbgPrint**: שולח את הפלט רק אם תנאים מסוימים קיימים במערכת, אם תנאי אחד או יותר לא מתקיים אז הפלט לא ישלח.
- **DbgPrintEx**: שולח את הפלט לפי התנאים שהקורא סיפק לפעולה שמצוינים תנאים שונים שצרכיכם להתקיים עבור שליחת פלט. לכן נוסףפה 2 פרמטרים: סוג החלק במערכת ההפעלה שקרה לפעולה וכמה שליחת הפלט המסויים קריטית לפעילות המערכת.

:DbgPrint

Remarks

DbgPrint and **DbgPrintEx** can be called at IRQL<=DIRQL. However, Unicode format codes (%C, %S, %Ic, %Is, %wc, %ws, and %wZ) can be used only at IRQL=PASSIVE_LEVEL. Also, because the debugger uses interprocess interrupts (IPIs) to communicate with other processors, calling **DbgPrint** at IRQL>DIRQL can cause deadlocks.

Only kernel-mode drivers can call the **DbgPrint** routine.

DbgPrint sends a message only if certain conditions apply. Specifically, it behaves like the **DbgPrintEx** routine with the DEFAULT component and a message importance level of DPFLTR_INFO_LEVEL. In other words, the following two function calls are identical:

```
C++  
Copy  
DbgPrint ( Format, arguments )  
DbgPrintEx ( DPFLTR_DEFAULT_ID, DPFLTR_INFO_LEVEL, Format, arguments )
```

For more information about message filtering, components, and message importance level, see [Reading and Filtering Debugging Messages](#).

It is recommended that you use **DbgPrintEx** instead of **DbgPrint**, since this allows you to control the conditions under which the message is sent.

:DbgPrintEx

The **DbgPrintEx** routine sends a string to the kernel debugger if the conditions you specify are met.

Syntax

```
C++  
Copy  
NTSYSAPI ULONG DbgPrintEx(  
    [in] ULONG ComponentId,  
    [in] ULONG Level,  
    [in] PCSTR Format,  
    ...  
) ;
```

Parameters

[in] ComponentId

Specifies the component calling this routine. This must be one of the component name filter IDs defined in the Dpfilter.h header file. To avoid mixing your driver's output with the output of Windows components, you should use only the following values for *ComponentId*:

- DPFLTR_IHVVIDEO_ID
- DPFLTR_IHVAUDIO_ID
- DPFLTR_IHVNNETWORK_ID
- DPFLTR_IHVSTREAMING_ID
- DPFLTR_IHVBUS_ID
- DPFLTR_IHVDRIVER_ID

[in] Level

Specifies the severity of the message being sent. This can be any 32-bit integer. Values between 0 and 31 (inclusive) are treated differently than values between 32 and 0xFFFFFFFF. For details, see [Reading and Filtering Debugging Messages](#).

:Bcrypt library

از למרות שהספריה bcrypt היא עצמה קבוצה של פעולות עם מקדם "Bcrypt", רציתו לתת לה יחס מיוחד כי היא מכילה פונקציונליות חשובה בתוכה המאפשרת שימוש בкриптוגרפיה מכל סוגיה בתוך רכיבים קיומיים. הספריה מותאמת לתוכנות ב-C או C++ ומהיה שונה בשמה עבור תוכנות ב-usermode (.kernelmode (cng.lib) או ב-(lib) (bcrypt.lib)).

הספריה וכל ה-apis שהיא מספקת מתועדים בצורה מעולה והספריה מאפשרת לבצע פעולות קרייפטוגרפיות כמו שימוש באלגוריתמים סימטריים או אסימטריים ופעלו hashing מכל הסוגים שלו. תחילה הבדיקה עם bcrypt נראה כך (פחות ביצוע hashing, מופיע גם בפרויקט ה-Minifilter שתיארתי מוקדם יותר):

הפעולה הקרייפטוגרפית הדרישה על נתונים. בפועל נספק בין היתר מצביע לאובייקט ה-provider ומחרוזת המתארת את סוג האלגוריתם בו נרצה להשתמש:

The BCryptOpenAlgorithmProvider function loads and initializes a CNG provider.

Syntax

```
C++ Copy
NTSTATUS BCryptOpenAlgorithmProvider(
    [out] BCRYPT_ALG_HANDLE *phAlgorithm,
    [in]  LPCWSTR          pszAlgId,
    [in]  LPCWSTR          pszImplementation,
    [in]  ULONG             dwFlags
);
```

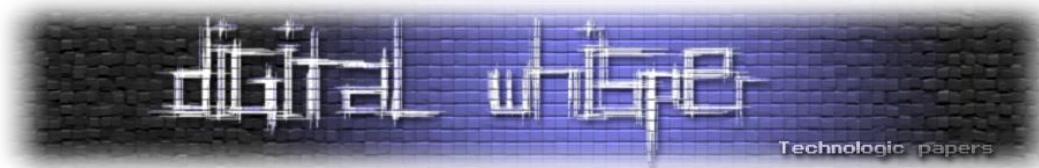
Parameters

[out] phAlgorithm

A pointer to a **BCRYPT_ALG_HANDLE** variable that receives the CNG provider handle. When you have finished using this handle, release it by passing it to the **BCryptCloseAlgorithmProvider** function.

[in] pszAlgId

A pointer to a null-terminated Unicode string that identifies the requested cryptographic algorithm. This can be one of the standard **CNG Algorithm Identifiers** or the identifier for another registered algorithm.



2) אלוקציה של מספיק זיכרון עבור hashing object. זה אובייקט שני שנוצר להשתמש בו עבור ביצוע hashing, ועבור המטרה יש לנו אפשרות להשיג את האורך הדרושים לאובייקט באמצעות הפעולה BCRYPT_OBJECT_LENGTH שמקבלת את הדגל BcryptGetProperty ומחזירה את הגודל:

C++

Copy

```
NTSTATUS BCryptGetProperty(
    [in] BCRYPT_HANDLE hObject,
    [in] LPCWSTR     pszProperty,
    [out] PUCHAR      pbOutput,
    [in] ULONG        cbOutput,
    [out] ULONG       *pcbResult,
    [in] ULONG        dwFlags
);
```

Parameters

[in] hObject

A handle that represents the CNG object to obtain the property value for.

[in] pszProperty

A pointer to a null-terminated Unicode string that contains the name of the property to retrieve. This can be one of the predefined

Cryptography Primitive Property Identifiers or a custom property identifier.

[out] pbOutput

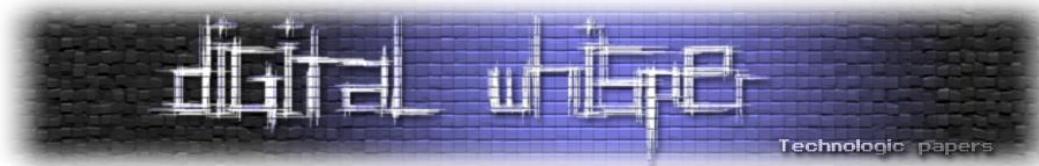
The address of a buffer that receives the property value. The *cbOutput* parameter contains the size of this buffer.

הציגו נכוו לpermeter השני בפעולה, וה-provider נכוו בpermeter הראשון. הפעולה תחזיר את הגודל למשביע :result של ה-provider

BCRYPT_OBJECT_LENGTH

L"ObjectLength"

The size, in bytes, of the subobject of a provider. This data type is a DWORD. Currently, the hash and symmetric cipher algorithm providers use caller-allocated buffers to store their subobjects. For example, the hash provider requires you to allocate memory for the hash object obtained with the **BCryptCreateHash** function. This property provides the buffer size for a provider's object so you can allocate memory for the object created by the provider.



(3) לפעולה את ה-provider, מצביע לזכרון המוקצת עבור האובייקט ומשתנה מティפואן כרך בשימוש שלנו:

```
424     NTSTATUS HelperFunctions::CreateDataHash(PVOID DataToHash, ULONG SizeOfDataToHash, LPCWSTR Ha

// Create the hashing object used to hash the actual data:
Status = BCryptCreateHash(HashAlgorithm, &HashHandle, (PUCHAR)HashObject, HashObjectLength, NULL, 0, 0)
if (!NT_SUCCESS(Status)) {
    goto CleanUp;
}
HashHandleCreated = TRUE;
```

(4) שלב רשות: הקצתה זיכרון עבור תוצאת ה-hash. בגלל שבמקרה זה ניתן לצפות מה יהיה אורך התוצאה ניתן להקצות מערך סטטי עבור התוצאה, אך אם בכלל מקרה יש צורך להקצתה דינמית ניתן להשיג את האורך עם ה-API משלב 2 רק עם הדגל BCRYPT_HASH_LENGTH

(5) ביצוע פועלות hashing בעזרת BcryptHashData וסיום נקי של כל התהליך עם הפעולה :BcryptFinishHash

```
424     NTSTATUS HelperFunctions::CreateDataHash(PVOID DataToHash, ULONG SizeOfDataToHash, LPCWSTR Ha

497         // Hash the actual data:
498         Status = BCryptHashData(HashHandle, (PUCHAR)DataToHash, SizeOfDataToHash, 0);
499         if (!NT_SUCCESS(Status)) {
500             goto CleanUp;
501         }
502
503
504         // Get the hash value (hash handle cannot be reused after this operation) and return it
505         Status = BCryptFinishHash(HashHandle, (PUCHAR)HashedData, HashDataLength, 0);
506         if (!NT_SUCCESS(Status)) {
507             goto CleanUp;
508         }
```

חשוב לוודא שאנו יוצאים מהפעולה כמו שצריך עם כל פקודות הנקוי המתאימות:

```
// Clean up and return successfully:  
CleanUp:  
    if (HashHandleCreated) {  
        BCryptDestroyHash(HashHandle);  
    }  
    if (HashProviderCreated) {  
        BCryptCloseAlgorithmProvider(HashAlgorithm, 0);  
    }  
    if (HashObject != NULL) {  
        ExFreePool(HashObject);  
    }  
    if (HashedData != NULL && !NT_SUCCESS(Status)) {  
        ExFreePool(HashedData); // Note: dont free HashedData if succeeded, will hold the hashed  
        HashedData = NULL;  
        HashedDataLength = 0;  
    }  
    *HashedDataOutput = HashedData;  
    if (HashedDataLength != NULL) {  
        *HashedDataLength = HashDataLength;  
    }  
    return Status;
```

- תהליך השימוש בספרייה עברו פעולות שונות כמו שימוש באלגוריתמים שונים או בסוגים שונים של הצפנות יהיה טיפה שונה מהתהליך שתיארתי כאן, בין אם ב-API שבו משתמשים ובין אם בסדר של השלבים, אך זה שילד טוב עבור רוב השימושים שנctrkr בפיתוח שלנו.

:KeStackAttachProcess/KeUnstackDetachProcess

הפעולות הללו מבוססות על שימוש במנגןון APC. תיארתי את המנגנון במאמרים קודמים אך בקיצור נוון לנו אפשרות לבקש לרוץ בתור תהליך כחלק מתהליך מסוים. מנגןון זה שימושי כשאנו צריכים גישה למשאים השייכים לאוטו תהליך, כגון מרחב הכתובות הווירטואליות של שלוו לא ניתן לגשת מחוץ לאוטו תהליכי:

Asynchronous Procedure Calls

Article • 01/07/2021 • 5 contributors

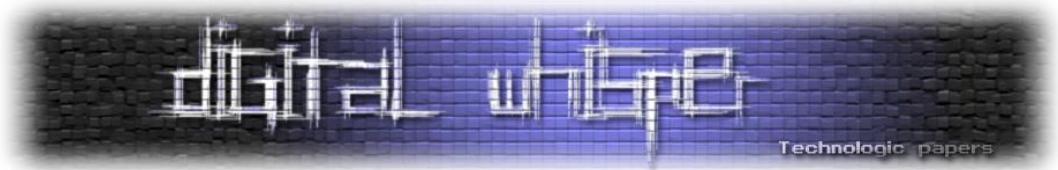
 Feedback

In this article

[Synchronization Internals](#)

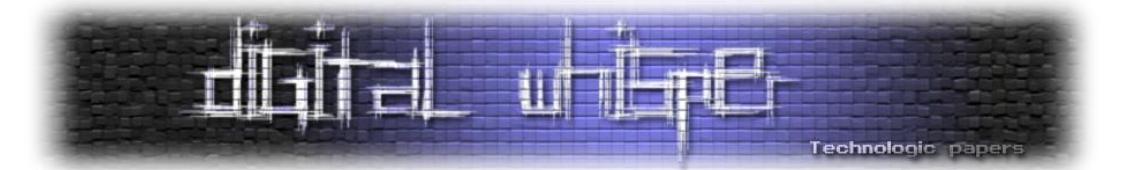
[Related topics](#)

An *asynchronous procedure call* (APC) is a function that executes asynchronously in the context of a particular thread. When an APC is queued to a thread, the system issues a software interrupt. The next time the thread is scheduled, it will run the APC function. An APC generated by the system is called a *kernel-mode APC*. An APC generated by an application is called a *user-mode APC*. A thread must be in an alertable state to run a user-mode APC.



Code Blame Raw ⌂ ⌄ ⌅ ⌆ ⌇

```
270
271 NTSTATUS HelperFunctions::UserToKernel(PEPROCESS SrcProcess, PVOID UserAddress,
272                                         PVOID KernelAddress, SIZE_T Size, BOOL IsAttached) {
273     KAPC_STATE SrcState = { 0 };
274
275
276     // Check for invalid parameters:
277     if (SrcProcess == NULL || UserAddress == NULL || KernelAddress == NULL || Size == 0)
278         return STATUS_INVALID_PARAMETER;
279 }
280
281
282     // Attach to the usermode process if needed:
283     if (!IsAttached) {
284         KeStackAttachProcess(SrcProcess, &SrcState);
285     }
286
287
288     // Perform the transfer:
289     __try {
290         ProbeForRead(UserAddress, Size, sizeof(UCHAR));
291         RtlCopyMemory(KernelAddress, UserAddress, Size);
292         if (!IsAttached) {
293             KeUnstackDetachProcess(&SrcState);
294         }
295     }
296
297
298     __except (STATUS_ACCESS_VIOLATION) {
299         if (!IsAttached) {
300             KeUnstackDetachProcess(&SrcState);
301         }
302         return STATUS_ACCESS_VIOLATION;
303     }
304 }
```



```
main > ShminiFilter / ShminiFilter / helpers.cpp > Top
```

Code Blame Raw ⌂ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉

```
307 NTSTATUS HelperFunctions::KernelToUser(PEPROCESS DstProcess, PVOID KernelAddress, PVOID User
308     SIZE_T Size, BOOL IsAttached) {
309     KAPC_STATE DstState = { 0 };
310
311
312     // Check for invalid parameters:
313     if (DstProcess == NULL || KernelAddress == NULL || UserAddress == NULL || Size == 0)
314         return STATUS_INVALID_PARAMETER;
315     }
316
317
318     // Attach to the usermode process if needed:
319     if (!IsAttached) {
320         KeStackAttachProcess(DstProcess, &DstState);
321     }
322
323
324     // Perform the transfer:
325     __try {
326         ProbeForRead(UserAddress, Size, sizeof(UCHAR));
327         RtlCopyMemory(UserAddress, KernelAddress, Size);
328         if (!IsAttached) {
329             KeUnstackDetachProcess(&DstState);
330         }
331     }
332     return STATUS_SUCCESS;
333 }
334
335     __except (STATUS_ACCESS_VIOLATION) {
336         if (!IsAttached) {
337             KeUnstackDetachProcess(&DstState);
338         }
339         return STATUS_ACCESS_VIOLATION;
340     }
341 }
342
343
```

:ProbeForRead/ProbeForWrite

בפעולות האחרונות ניתן לראות שימוש בפעולה `ProbeForRead`, והמטרה של הפעולה זו ושל הפעולה `ProbeForWrite` היא לבדוק שטוחה כתובות מסוים נגיש מكونടקסט הריצה הנוכחי. פעולה אלו שימושיות במקרים שבהם טוחן כתובות מסוים מבוסס על פרמטרים מקורא לא ידוע או אפילו מ-usermode.

از מה ההבדל בין שתי הפעולות? ניתן להבין אותו מהשם:

This function typically provides no substantial benefit over **ProbeForRead** because a robust driver must always be prepared to handle protection changes in the user mode virtual address space, including protection changes that remove write permission to a buffer passed to a driver after a **ProbeForWrite** call has executed. Because **ProbeForWrite** accesses each page in the supplied buffer, performance may be reduced due to the overhead of accessing each page, especially if the supplied buffer describes a large region of virtual address space. In addition, because **ProbeForWrite** writes to each page, the same buffer may not be used safely with multiple concurrent driver requests. For these reasons, new driver code should always use **ProbeForRead** instead.

The following table outlines the properties of each kernel mode buffer probing routine:

 Expand table

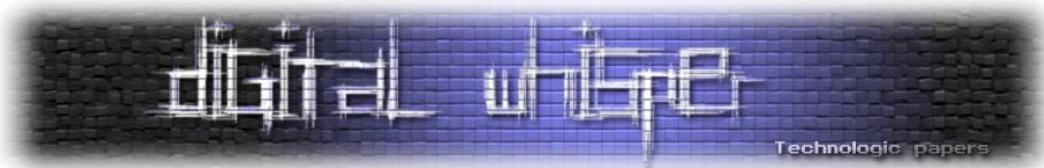
Behavior	ProbeForRead	ProbeForWrite
Confirms that buffer describes a region in user mode address space, if the length is non-zero	x	x
Confirms that buffer base address and length do not wrap past maximum pointer value	x	x
Confirms that buffer is aligned to requested alignment boundary, if the length is non-zero	x	x
Confirms that buffer is initially writable (may change at any time if user mode application reprotects its address space)		x
Accesses each page in the buffer (additional overhead)		x
Modifies each page in the buffer (may cause unexpected behavior if same buffer is used in parallel with multiple driver requests)		x

בזמן ש-**ProbeForRead** רק מנסה לקרוא למרחב הכתובות, **ProbeForWrite** ינסה לכתוב במקום מסוים בכל PAGE של מרחב הכתובות כדי לוודא שקיימות הרשותות כתיבה באותו איזור זיכרון. קיימות

בגישה זו מספר בעיות שבגללן נעדיף להשתמש ב-**ProbeForRead**:

- 1) גישה זו מתעלמת משינוי הרשותות של איזורי זיכרון בזמן ריצת המערכת או אפילו בזמן ריצת הפעולה.
- 2) קיימ overhead שימושי בפעולה בגין שהיא כתובה לתוך כל PAGE במרחב הכתובות, שהוא ממשמעותי מאוד למרחבי כתובות גדולים.
- 3) ניתן לבצע את בדיקת הרשותות זו בעזרת הרצה פשוטה של הפעולה **ZwQueryVirtualMemory** שתחזיר לנו את הרשותות איזור הזיכרון

כדי להשתמש נכון בפעולה נצטרך להכניס אותה לתוך **try statement** ולטפל ב-**exception** של **STATUS_ACCESS_VIOLATION**, מהסיבה ש-**exception** זה יופעל על ידי הפעולה במקרה שהחלק מסוים למרחב הכתובות או כל המרחב לא נגייש בקונטקסט הריצה הנוכחי.



הפעולות מקבלות גם פרמטר שנקרא Alignment, לא נעשה שימוש נפוץ בפרמטר זהה ולכן נספק ערך שיסמל שאין חשיבות לפרמטר זהה:

Syntax

C++

Copy

```
void ProbeForWrite(
    [in, out] volatile VOID *Address,
    [in]      SIZE_T      Length,
    [in]      ULONG       Alignment
);
```

Parameters

[in, out] Address

Specifies the beginning of the user-mode buffer.

[in] Length

Specifies the length, in bytes, of the user-mode buffer. See additional information in the Remarks section.

[in] Alignment

Specifies the required alignment, in bytes, of the beginning of the user-mode buffer.

Return value

None

The screenshot shows a code editor interface with the following details:

- Project:** ShminiFilter / ShminiFilter / helpers.cpp
- File:** helpers.cpp
- Line:** 104
- Code:**

```
104     BOOL HelperFunctions::ChangeProtectionSettings(HANDLE ProcessHandle, PVOID Address, ULONG Si
__try {
    ProbeForRead(Address, Size, sizeof(UCHAR));
    Status = ZwQueryVirtualMemory(ProcessHandle, Address, MemoryBasicInformation, &MemoryInfo,
        if (!NT_SUCCESS(Status)) {
            return TRUE;
        }
    }
__except (STATUS_ACCESS_VIOLATION) {
    return TRUE;
```
- Toolbar:** Includes buttons for Raw, copy, paste, and other file operations.

:RtlPcToFile/Path/Name...

בහינתן pc (כתבת בסיס ל-Kernel Module בזיכרון מערכת), הפעולה מוחזירה את המידע הרלוונטי על ה-
Driver במקורה וקיים כזה:

RtlPcToFile[Name|Path|Header]()

```
NTSTATUS  
RtlPcToFileName(  
    PVOID PcValue,  
    PUNICODE_STRING FileName);  
  
NTSTATUS  
RtlPcToFilePath(  
    PVOID PcValue,  
    PUNICODE_STRING FilePath);  
  
PVOID  
RtlPcToFileHeader (  
    PVOID PcValue,  
    PVOID *BaseOfImage);
```

RtlPcToFileName() and RtlPcToFilePath() enable a driver to get the name and path of a kernel module respectively, from an address that points to the PE image of the module in memory. RtlPcToFileName() and RtlPcToFilePath() walk-through the list of KLDR_DATA_TABLE_ENTRY structures anchored at PsLoadedModuleList and determine if the given address (Pc) falls within the extents in KLDR_DATA_TABLE_ENTRY.Base and KLDR_DATA_TABLE_ENTRY.Base + KLDR_DATA_TABLE_ENTRY.SizeOfImage and if so, return information about the module.

:MmGetSystemRoutineAddress

הפעולה נועדה להשיג כתבות בסיס לפעולות המערכת המוממשות בקורסיל אך לא כלות בספריה שבה אנחנו משתמשים לרוב הפעולות מהקורסיל - ntoskrnl.lib. הפעולה מקבלת מחרוזת של שם הפעולה שאנו רוצים להציג, ואם קיימת פעולה כזו (אחרת הפעולה תחזיר NULL) הפעולה תחזיר את כתובת הבסיס שלה:

```
UNICODE_STRING ExAllocateCallBackName = RTL_CONSTANT_STRING(L"ExAllocateCallBack");  
UNICODE_STRING ExFreeCallBackName = RTL_CONSTANT_STRING(L"ExFreeCallBack");  
PEX_CALLBACK_ROUTINE_BLOCK CallbackBlock;  
INT CallbackContext;  
  
if (!ExAllocateCallBack)  
    ExAllocateCallBack = MmGetSystemRoutineAddress(&ExAllocateCallBackName);  
if (!ExFreeCallBack)  
    ExFreeCallBack = MmGetSystemRoutineAddress(&ExFreeCallBackName);
```

:RtlFindExportedRoutineByName

כפי שציינו, הפעולה MmGetSystemRoutineAddress תחזיר לנו מצביע לפעולה המומומשת בקרNEL שאט שמה סיפקו לפעולה. בעזרה טיפה הנדסה לאחר הפעולה MmGetSystemRoutineAddress ניתן לראותה שהיא בפועל מוצאת את כתובות הבסיס של הKernel ושולחת את אותה כתובות בסיס ושם של הפעולה המיצאת שאנו רוצים להציג לפעולה RtlFindExportedRoutineByName. הפעולה זו מקבלת פרמטר לשם של הפעולה המיצאת שאנו רוצים להציג ואת כתובות הבסיס בזיכרון של התוכנה ממנה אנחנו רוצים להציג את הפעולה המיצאת, ובעזרת הפעולה נוכל להציג גישה לכל פעולה מיצאת על ידי כל תוכנה

הקיים בזיכרון המערכת בצורה נוחה וקלה:

RtlFindExportedRoutineByName()

```
PVOID  
RtlFindExportedRoutineByName (   
    PVOID DllBase,  
    PCHAR RoutineName);
```

The kernel provides the documented function MmGetSystemRoutineAddress() for drivers to retrieve a pointer to an export, given the name of the export. MmGetSystemRoutineAddress() does have a restriction in that it works on exports in NTOSKRNL.exe and HAL.dll. MmGetSystemRoutineAddress() internally calls the undocumented function RtlFindExportedRoutineByName() with the base address of NTOSKRNL.exe stored in nt!PsNtosImageBase and with the base address of HAL.dll in nt!PsHalImageBase. It turns out that RtlFindExportedRoutineByName() is also exported by NTOSKRNL.exe and therefore available for drivers to call directly to get the exports of any kernel module just as applications call GetProcAddress() on any DLL. A notable difference between MmGetSystemRoutineAddress() and RtlFindExportedRoutineByName() is that while the former takes in the RoutineName as a Unicode string, the latter expects a NULL-terminated ASCII string.

סיכום

במאמר CISITI כמה אבני בניין חשובות לכל מי שרצה להיכנס לנושאי פיתוח קרNELים במערכות Windows, מהרמת הסביבה לפיתוח ועד כתיבת קוד איקוטי ומוקדק מטרה. במיוחד בתוכנה צו שobox מואוד שלו מתכונת ידע לפתח את המוצר שלו בצורה הכי מדוקנית וטובה, וכל פרצה או שגיאה שתיה זניחה לחולtein בתוכנה רגילה תוכל להיות קритית ומסוכנת ביותר ב- Driver Kernel ותוכל לגרום למקרים כמו מקרה CrowdStrike המדבר. השגיאות הללו לא יהיו בהכרח מהתקפת סייבר מתוחכמת, אלא יוכל להיות גם על ידי מתכונת מתחילה בחברה המספקת מוצריים למלינוי מערכות מבוססות Windows שלא פעיל בהתאם לחקלים הקיימים של המוצר. בנוסף של דבר המתכונת הוא האשם ככה או ככה וכן צריך לדעת איך לתכנת נכון.

על המחבר

בנ' 18, מהנדס תוכנה ב-software checkpoint. מתעניין מאוד בתחום הփיתוח ומחקר בסביבת lowlevel, מערכות הפעלה וابטחת מידע. מעוניין מאוד לפתח את הידע שלי וללמוד עוד כדי להתפתח בתחום. בין הפרויקטים המרכזיים שלי עבדתי על רוטקיט למערכת הפעלה Windows 10 כדי להחביא תהליכי, קבצים ותובורת רשת, כמו כן גם פיתחתי מערכת להגנה מנוזקות קernelיות כמו שלי ואחרות ופיתחתי וחקרתי דרייברים ומבני נתונים פנימיים רבים.

- ניתן לראות את הפרויקטים האלו ואחרים בעמוד הגיטהאב שלו: <https://github.com/shaygitub>
- ניתן ליצור איתי קשר דרך האימייל שלו: shaygilat@gmail.com
- או דרך עמוד הלינקדאין שלו: <https://www.linkedin.com/in/shay-gilat-67b727281>

ביבליוגרפיה

קישורים למאמרים הקודמים שלי שמספקים ידע חשוב בנושא Windows internals ויכולים לעזור להבין יותר טוב את התוכן במאמר זה:

<https://digitalwhisper.co.il/files/Zines/0xA2/DW162-2-OffensiveWinKernel.pdf>
<https://digitalwhisper.co.il/files/Zines/0xA3/DW163-1-BYODV.pdf>
<https://digitalwhisper.co.il/files/Zines/0xA4/DW164-3-ReverseWinDrivers.pdf>

:Visual Studio 2022 + Windows SDK + Windows WDK

<https://learn.microsoft.com/en-us/windows-hardware/drivers/download-the-wdk>

קישור להורדת Sysinternals Suite

<https://learn.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite>

הסביר על altitude ב-filter Drivers .:filter Drivers altitude

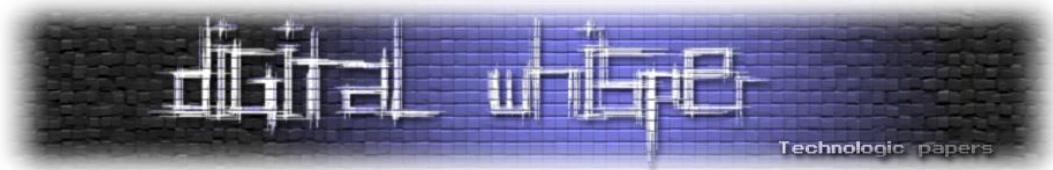
<https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/load-order-groups-and-altitudes-for-minifilter-drivers>

יצירת קובץ INF עבור Minifilter Driver :Minifilter Driver

<https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/creating-an-inf-file-for-a-minifilter-driver>

קישור לדוגמתה Minifilter שיצרתי והדגמתי במאמר:

<https://github.com/shaygitub/ShminiFilter>



קישור לפרויקט שלי שמעבודה עליו למדתי את כל מה שהסבירתי במאמר (והעמוד גיטהאב שלי ☺):

<https://github.com/shaygitub/windows-rootkit>

קישור ל-Handles Internals במערכות מבוססות Windows : Handles Internals

<https://guidedhacking.com/threads/what-are-windows-handles-windows-internals-explained.20345>

קישור למאמר נוסף בנוגע ל-Handles Internals : Handles Internals

<https://medium.com/@amitmoshel70/mysteries-of-the-windows-kernel-pt-1-processes-objects-d677a5afcd9b>