Digital Whisper
Technologic Papers

התפוח על אדן החלון

מאת ליאל חנוכוב

הקדמה

iTunes For Windows הינה תוכנה חינמית של חברת Apple שמאפשרת לבעלי מכשירי אפל לחברם למחשב הווינדוס שלהם ולסנכרן את ספריות המדיה השונות, לעדכן את המכשיר, להתחבר ל-iCloud ולספק מרחב פעולה עם מכשירי אפל. בזכות iTunes אני יכול להוריד קבצי mp3 לאייפון שלי בלי הצורך להשתמש בשירותים כמו ספוטיפיי או אפל מיוזיק. תוכנה זו מותקנת על מחשיבי האישי רוב חיי וכך גם למיליוני אנשים ברחבי כדור הארץ בלי תיעוד נרחב או מחקר שפורסם ברשת, לכן היה לי חשוב להבין איך היא עובדת ולשתף את המחקר.

אם תתקינו iTunes במחשב ווינדוס תראו שמורדות לכם יותר מ-10 אפליקציות עזר שגם נותנות תמיכה לפיצ'רים ש-iTunes צריכה כמו השירות iPodService.exe שנותן לנו לתקשר עם מכשירי אייפוד (איך אני אוהב את האייפוד שאפל שלי) ועד mDNSResponder, פרויקט קוד פתוח שאחראי על יישום Multicast DNS כדי שנוכל לבצע שאילתות למחשבים ברשת הלוקלית ואפילו להתחבר ל-airplay. חשוב לציין שאם תחברו את האייפון שלכם למחשב תוכלו רק לגלוש בתמונות בפרוטוקול PTP.

במאמר זה אציג בפניכם איך מערך הדרייברים בווינדוס עובד ובתוכו הדרייברים של אפל AppleLowerFilter ו-AppleKmdfFilter שמהווים חלק אינטגרלי מאיך שנדבר עם מכשירי אפל, נראה את השירותים השונים שמציע השירות Apple Mobile Device Service שמאפשר לנו לתקשר עם האייפון משכבת ה-User Mode וארחיב על נושאים כמו USB descriptors, Windows Driver Frameworks (WDF) והמסע שפקטת USB תעבור עד שתגיע אלינו.

קריאה מהנה!

זיהוי המכשיר והפעלת הדרייברים המתאימים

כשמכשיר USB חדש מתחבר למחשב, מערכת ההפעלה תיצור Physical Device Object (PDO) חדש וה-PnP Manager ידווח על כך לדרייברים שאחראיים על אותו מכשיר בעזרת מזהיי החומרה (idVendor, idProduct) שישמשו כדי למצוא את הדרייברים הכי מתאימים ב-Driver Store.

מכשירי אפל מציגים את עצמם כ-composite device עם מספר ממשקים כדי שנוכל לגשת לפונקציות נפרדות כמו העברת קבצים, וידיאו ואודיו. כשנחבר מכשיר אפל למחשב ווינדוס מתאם ה-USB יעביר את מזהיו למערכת ההפעלה שתחפש את הדרייבר המתאים ביותר. ופה קבצי inf באים לפעולה, קבצי inf הם קבצי קונפיגורציה שמטרתם להוריד תוכנות נדרשות בעת חיבור המכשיר, במקרה שלנו קובץ זה יהיה AppleUSB.inf. קובץ זה כולל הוראות קונפיגורציה למכשירים רבים (מ-HomePod עד AppleWatch) אך היום נתמקד באייפון.

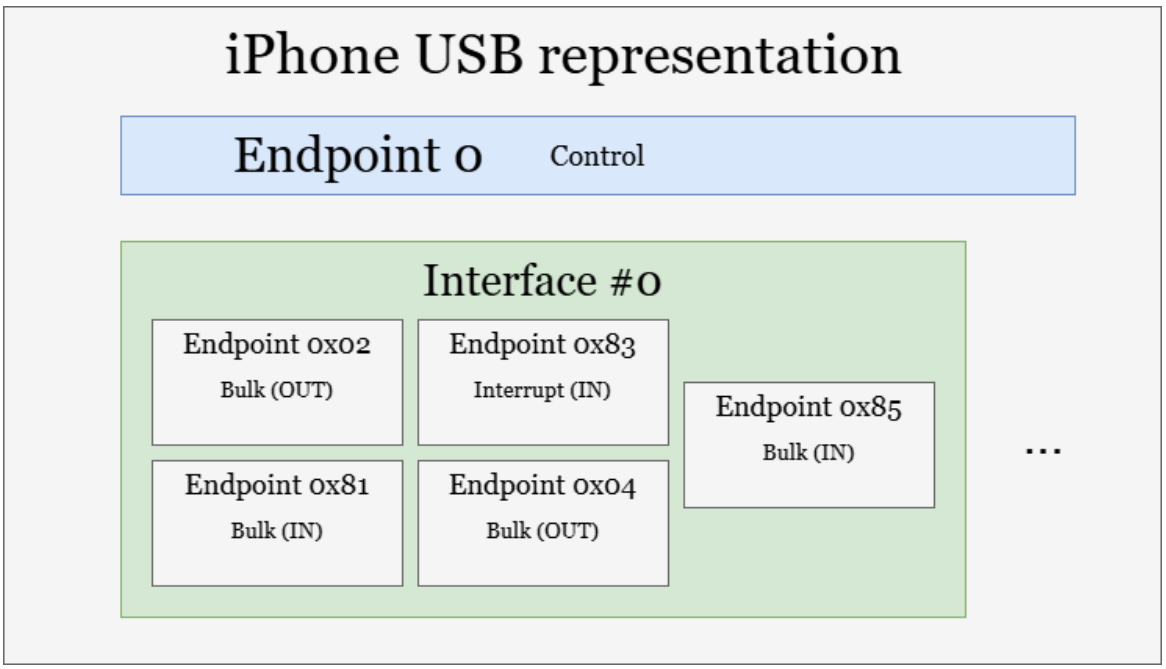
```
[AppleUSB_CCGPDriverInstall_AddReg.HW]
HKR,,"OriginalConfigurationValue",0x00010001,2
HKR,,"UsbccgpCapabilities",0x00010001,0x10
HKR,,FriendlyName,,%iPhone.AppleUSB.DeviceDesc% ; "Apple Mobile Device USB Composite Device"
HKR,,LowerFilters,0x00010000,AppleLowerFilter
```

אפשר לראות בתמונה שלאחר שנחבר את המכשיר יועתק הדרייבר AppleLowerFilter למערכת ההפעלה בנתיב C:\Windows\System32\drivers\. ויוסיף את הערכים הרשומים למפתח המכשיר ברגיסטרי. ערך ה-OriginalConfigurationValue הוא ערך שיוגדר עבור הדרייבר usbccgp.sys במפתחות הרגיסטרי של המכשיר, הוא מחליט באיזו קונפיגורציה המכשיר ישתמש כשהוא יתחבר, כשמכשיר מתחבר בפעם הראשונה מערכת ההפעלה תקרא את הערך ותטען את הקונפיגורציה הספציפית הזו, דבר שיכול להיות רלוונטי למכשירים עם מספר קונפיגורציות. את השלבים הבאים ניתן לראות באתר של מייקרוסופט, המכשיר יאותחל ולאחר האיתחול ה-PnP manager יזהה את הדרייברים הפונקציונלים (drivers function) ואת הדרייברים המסננים (Filter Drivers) הרלוונטים למכשיר וייבנה את הרכב הדרייברים שבמקרה שלנו הדרייבר הפונקציונלי יהיה Usbccgp שמהווה מולטיפלקסר למכשירים כך שנוכל להתממשק עם כמה דרייברים וממשקים לאותו מכשיר, והדרייבר המסנן יהיה AppleLowerFilter. סדר הקריאה הוא כך שפונקציית ה-AddDevice תקרא עבור כל דרייבר ונתחיל עם הדרייברים המסננים ולאחר מכן לפונקציונלים, ה-PnP manager יישלח את הבקשה IRP_MN_START_DEVICE לדרייברים של המכשיר.

קצת על USB

לפני שנתחיל לראות את השדות והממשקים שהאייפון חושף למחשב חשוב לי לציין שפרוטוקול USB הוא פרוטוקול עצום שקיים בהרבה מכשירים אלקטרוניים ולא תקף רק למכשירים, מדובר בכבלים, אותות חשמליים ועוד המון דברים שנמצאים מאחורי הקלעים במוצרים שאנו משתמשים בהם ביום יום. יש סוגים שונים של הפרוטוקול עם מהירויות שונות כמו Low speed, Full Speed, High speed ו-USB 3.0 שמוגדר כ-Super speed וזה אפילו עוד לפני שנגענו ב-USB-C שמביא תכונות רבות איתו. כל תכונה כזו מוסיפה המון סיבוכיות בשכבת החומרה, אנחנו יכולים להתנחם בכך שזה לא כל כך גרוע בשכבת התוכנה ששם אנחנו מקבלים פחות או יותר מראה אחיד. לכן אציג בפניכם את בעיקר את מה שנקבל מאייפון בשכבת התוכנה. למי שרוצה לקרוא מעבר למה שייאמר פה, מומלץ לקרוא את המפרט של USB 2.0 כדי להבין נושאים כמו איך כבל USB בנוי ואיך נבצע איתות דיפרנציאלי.

תחשבו על מצלמה שתחברו למחשב, כשתחברו מצלמה סביר שיהיה לה גם מיקרופון ולכן בדרך כלל יהיה לנו ממשק גם לזרימת וידאו וממשק נפרד לזרימת אודיו. אם נסתכל על האייפון גם לו יש מספר ממשקים, ממשק להעברת קבצים, ממשק להעברת אודיו וממשק להעברת תמונות. לכל ממשק יש מספר נקודות קצה. נקודת קצה היא נקודה שנוכל לשלוח מידע אליה וממנה, כמו פורט במחשב. יש לנו נקודת קצה שמוגדרת OUT ונקודת קצה שמוגדרת IN בשביל קלט ופלט כש-IN אומר לנו שהמידע בפייפ (PIPE) יעבור מהמכשיר למחשב ו-OUT מהמחשב למכשיר בנוסף לנקודת הקצה הראשית שנקראת Endpoint 0 ומחויבת להיות על כל מכשיר USB כשהיא מהווה נקודת שליטה בין המחשב המארח למכשיר ומנהלת את כל תעבורת השליטה, אנומרציה של מכשירים וניהול פקודות הנשלחות אליה.



נוריד את תוכנת Usbview שבאה כחלק מערכת כלי הדיבוג של ווינדוס ונראה את מכשיר האייפון ונראה את תכונותיו. כמו ה-Vendor id שמאפיין את חברת אפל שבמקרה שלנו הערך שלו יהיה 0x05AC, כל חברה יכולה לפנות לארגון USB-IF, לשלם עמלה ולהשיג מזהה מיוחד לחברה שיאפיין את מכשיריה. לאחר מכן נראה את מזהה המכשיר ה-Product ID ועוד מידע גנרי על המכשיר כמו סוג ה-USB, בהרבה רכיבים זולים ניתן לראות שמזהה ה-Vendor שונה ממזהה היצרן ה-iManufacturer עקב העובדה שבדרך כלל חברות קטנות לא יהיו בעלות המפעל בארץ הייצור, שיטה נחמדה לראות מי באמת ייצר את המכשיר. לאורך השדות נראה כאלה שמתחילים ב-b, bcd, i, w, bm כשאלו הרלוונטים למכשיר שחיברנו.

- w - word, השדה יהיה בעל גודל של שתי בייטים.
- b - השדה יהיה בעל ערך בגודל בייט אחד.
- i - index, משמש כמצביע בגודל בייט אחד שיפנה אותנו למערך המחרוזות של המכשיר כמו שם היצרן.
- bm - bitmap, נתייחס לשדה בתור הביטים העצמאיים שלו, לדוגמה אם ביט מספר 6 יהיה שווה 1 נדע שהמכשיר מפעיל את עצמו.
- bdc (Binary-Coded decimal) - שדה קצת מבלבל מכיוון שלא מדובר בשדה שהוא Binary-Coded אלא שדה שהוא Hexadecimal-Coded ובתוכו נראה שתי ערכי הקס שיאפיינו אותו, לדוגמה 0x0300.

קונפיגורציית המכשיר תאפיין את תכונות המכשיר לדוגמה נראה שהמכשיר מפעיל את עצמו ולא חייב לקבל אנרגיה ממתאם ה-USB אך האנרגיה המקסימלית שאנחנו מבקשים שנקבל בכבל היא 500mA. לאחר מכן נראה את ה-Interface Descriptor שבדרך כלל מכיל הרבה פרמטרים של המכשיר אך במקרה שלנו המידע השימושי שנקבל מה-descriptor הזה הוא מספר נקודות הקצה שיש לו, שיהיה 3 ושהמכשיר משתמש בפרוטוקול Picture Transfer Protocol (PTP). החלק המעניין הוא נקודות הקצה שיגידו לנו איזה מידע עובר בנקודה ואיך. יש סוגים שונים של נקודות קצה, אם נחשוב על מקלדת לדוגמה, היא לא מעבירה מידע מהר כמו מצלמה או במקרה שלנו הטלפון שייצטרכו להעביר במהירות ובמידות גדולות. בשביל זה יש לנו את השדה Transfer Type, כפי שראינו בנקודות הקצה של האייפון יש לנו 5 נקודות קצה ש-4 מהן הן מסוג Bulk ו-1 מסוג Interrupt.

לפי השם כבר נוכל להבין ש-Bulk ישומש להעברת רוב המידע, נוכל להקביל זאת למוצר כמו מכשיר אחסון חיצוני שנבקש קובץ מסוים ואם היו שגיאות בדרך המכשיר ינסה לשלוח שוב עד שנקבל את המידע במלואו. העברות יהיו באותו קצב של Interrupt אך ללא polling rate מוגדר וייקחו את כל רוחב הפס הזמין להן, בגרסאות שונות של USB יהיו לנו הגבלות שונות על גודל הפקטות, ב-full-speed נוכל להעביר 8, 16, 32 או 64 בתים בפקטה לעומת 512 ב-high-speed שנראה שהאייפון מקבל. אם נשווה זאת למצלמות נראה שמצלמות יידרשו Transfer Type מסוג Isochronous כדי להעביר מידע גדול בקצב אחיד בלי ניסיון לזהות שגיאות.

Interrupt ישומש להעביר פקודות או מידע ספציפי. למי שהתעסק עם בקרים או תכנות low-level למערכות הפעלה יודע ש-Interrupt הוא דבר שאתה לא שולט בו, הוא בא למערכת ואתה חייב להתמודד איתו כשהוא מגיע, זה לא המקרה כשאנחנו מתקשרים עם USB.

ב-USB נציין בנקודת הקצה כל כמה זמן אנחנו רוצים שהמחשב ייבדוק אם קיים אצלנו Interrupt לפי השדה bInterval ויראה אם יש מידע חדש שהמכשיר רוצה לשלוח לפי גודל הפקטה המקסימלי wMaxPacketSize. ה-Transfer Type שנשאר לנו לעבור עליו הוא הוא Control Transfer שהוא היחיד שיכול לתקשר לשני הכיוונים (IN ו-OUT) מאותו פייפ וחייב להיות בכל מכשיר בדרך כלל ב-Endpoint 0, לסוג זה שלושה שלבים:

1. שלב ה-SETUP שכולל 8 בתים ומרכיב את פקטת ה-SETUP מגדירים את הבקשה וכמה מידע צריך להיות מועבר בשלב ה-DATA.
2. שלב ה-DATA הוא רשות מכיוון שלא כל הבקשות צריכות מידע ואם כן נצטרך, השלב יתחיל תמיד עם העברת מידע הכוללת את הפקטה DATA1 ולאחר מכן סוג העברת המידע קופצת עם כל פקטה בין DATA0 ל-DATA1. ככה שאם פקטה תאבד נוכל לדעת זאת לפי הזוגיות של מספר הפקטות.
3. שלב ה-STATUS ייכלול פקטת DATA1 ריקה ואם ה-DATA stage היה IN אז הסטטוס יהיה OUT ולהפך.

וככה זה ייראה בתוכנה Usbview:

```
[Port8]  :  Apple Mobile Device USB Composite Device


Is Port User Connectable:         yes
Is Port Debug Capable:            no
Companion Port Number:            2
Companion Hub Symbolic Link Name: USB#ROOT_HUB30#4&394145fb&0&0#{f18a0e88-c30c-11d0-8815-00a0c906bed8}
Protocols Supported:
 USB 1.1:                         yes
 USB 2.0:                         yes
 USB 3.0:                         no

Device Power State:               PowerDeviceD0

       ---===>Device Information<===---
English product name: "iPhone"

ConnectionStatus:
Current Config Value:              0x03  -> Device Bus Speed: High (is not SuperSpeed or higher capable)
Device Address:                    0x0D
Open Pipes:                           5

          ===>Device Descriptor<===
bLength:                           0x12
bDescriptorType:                   0x01
bcdUSB:                          0x0200
bDeviceClass:                      0x00  -> This is an Interface Class Defined Device
bDeviceSubClass:                   0x00
bDeviceProtocol:                   0x00
bMaxPacketSize0:                   0x40 = (64) Bytes
idVendor:                        0x05AC = Apple
idProduct:                       0x12A8
bcdDevice:                       0x1405
iManufacturer:                     0x01
     English (United States)  "Apple Inc."
iProduct:                          0x02
     English (United States)  "iPhone"
```

```
          -------->Open Pipes<------

          --->Endpoint Descriptor<---
bLength:                            0x07
bDescriptorType:                    0x05
bEndpointAddress:                   0x02  -> Direction: OUT - EndpointID: 2
bmAttributes:                       0x02  -> Bulk Transfer Type
wMaxPacketSize:                   0x0200 = 0x200 max bytes
bInterval:                          0x00

          --->Endpoint Descriptor<---
bLength:                            0x07
bDescriptorType:                    0x05
bEndpointAddress:                   0x81  -> Direction: IN - EndpointID: 1
bmAttributes:                       0x02  -> Bulk Transfer Type
wMaxPacketSize:                   0x0200 = 0x200 max bytes
bInterval:                          0x00

          --->Endpoint Descriptor<---
bLength:                            0x07
bDescriptorType:                    0x05
bEndpointAddress:                   0x83  -> Direction: IN - EndpointID: 3
bmAttributes:                       0x03  -> Interrupt Transfer Type
wMaxPacketSize:                   0x0040 = 1 transactions per microframe, 0x40 max bytes
bInterval:                          0x0A

          --->Endpoint Descriptor<---
bLength:                            0x07
bDescriptorType:                    0x05
bEndpointAddress:                   0x04  -> Direction: OUT - EndpointID: 4
bmAttributes:                       0x02  -> Bulk Transfer Type
wMaxPacketSize:                   0x0200 = 0x200 max bytes
bInterval:                          0x00

          --->Endpoint Descriptor<---
bLength:                            0x07
bDescriptorType:                    0x05
bEndpointAddress:                   0x85  -> Direction: IN - EndpointID: 5
bmAttributes:                       0x02  -> Bulk Transfer Type
wMaxPacketSize:                   0x0200 = 0x200 max bytes
bInterval:                          0x00

       -------->Full Configuration Descriptor<------

          --->Configuration Descriptor<---
bLength:                            0x09
bDescriptorType:                    0x02
wTotalLength:                     0x0027  -> Validated
bNumInterfaces:                     0x01
bConfigurationValue:                0x01
iConfiguration:                     0x05
     English (United States)  "PTP"
bmAttributes:                       0xC0  -> Self Powered
MaxPower:                           0xFA = 500 mA

          --->Interface Descriptor<---
bLength:                            0x09
bDescriptorType:                    0x04
bInterfaceNumber:                   0x00
bAlternateSetting:                  0x00
bNumEndpoints:                      0x03
bInterfaceClass:                    0x06  -> This is an Image USB Device Interface Class
bInterfaceSubClass:                 0x01
bInterfaceProtocol:                 0x01
iInterface:                         0x1B
     English (United States)  "PTP"
```

אם תרצו להעמיק אפנה אתכם למדריך הנהדר של חברת arm שיסביר לעומק איך לבנות בקרים של USB והקונספטים השונים שיש בפרוטוקול.

אחרי שהבנו איך הדרייבר AppleLowerFilter מורד ואת הבסיס של מכשירי USB נוכל לצלול ולהבין לעומק איך מכשירי אפל מתקשרים עם מחשבי ווינדוס. מושג שכדאי להבין הוא שהייעוד של דרייבר מסנן הוא להכנס ל-Device Stack, לנטר, לשנות ולהשתלט על בקשות IRP לפני שההיררכיה הרגילה של דרייברים תבצע את העבודה. AppleLowerFilter הוא דרייבר מסנן תחתון (Lower Filter) שמנטר את בקשות ה-USB ו-PTP למכשיר ועורך אותן. לעומת דרייבר מסנן עליון (Upper Filter) שבדרך כלל יישב בסוף היררכיית הדרייברים כשאנחנו הכי קרובים לשכבת ה-User Mode וישמש יותר ללוגים, פרסור פרוטוקולים או הצפנה.

הדבר ראשון שנראה כשנפתח את הדרייבר בדיקומפיילר כמו IDA הוא שמדובר בדרייבר מבוסס על גבי ה-Windows Driver Frameworks (WDF) ולא המודל הרגיל של דרייברים בווינדוס, ה-Windows Driver Model (WDM) מה שייקשה עלינו לבצע חקירה סטטית ראשונית בדיסאסמבלר עם הדרייבר מכיוון שנראה פונקציות רבות של WDF בתור אופסט ממבנה הפונקציות של WDF כפי שניתן לראות בנוסף לשמות משתנים לא מזוהים בתמונה הבאה:

```
13  result = WdfVersionBind(a1, &DestinationString, &unk_140006040, &qword_140006300);
14  if ( (int)result >= 0 )
15  {
16    qword_1400062D8 = *(_QWORD *)(qword_1400062F8 + 1608);
17    v5 = sub_140003DE0(&unk_140006040);
```

לשם כך נוריד את התוסף ida-kmdf למחשב שלנו כך שבמקום שכשנדבג את הדרייבר עם WinDbg לא נצטרך לעצור כל קריאה לפונקציה ולפענח את הפונקציה לפי הכתובת שלה אחרי שנטען את הספריה האחראית על wdf. לדוגמה עם הפקודה ln (List Nearest Symbol) נראה איזו פונקציה נמצאת בכתובת fffff8060c4955f0 ונקבל את הפלט הבא שיראה לנו פונקציה של WDF:

```
2: kd> ln fffff8060c4955f0
Browse module
Set bu breakpoint

 [minkernel\wdf\framework\shared\core\fxdeviceinitapi.cpp @ 1342] (fffff806`0c4955f0)   Wdf01000!imp_WdfFdoInitSetFilter
Exact matches:
    Wdf01000!imp_WdfFdoInitSetFilter (struct _WDF_DRIVER_GLOBALS *, struct WDFDEVICE_INIT *)
```

על מנת שלא נצטרך לבצע זאת כל קריאה לפונקציה שיפרתי את התוסף שבנה thalium והוספתי לו תמיכה ל-IDA גרסא 9, ניתן להוריד את הפורק מגיטהאב. וככה התוצאה תראה כשנשווה לפני ואחרי:

```
13  result = WdfVersionBind(a1, &DestinationString, &unk_140006040, &qword_140006300);
14  if ( (int)result >= 0 )
15  {
16    qword_1400062D8 = *(_QWORD *)(qword_1400062F8 + 1608);
17    v5 = sub_140003DE0(&unk_140006040);
```

↓

```
20  result = WdfVersionBind(DriverObject, &DestinationString, &BindInfo, &WdfDriverGlobals);
21  if ( result >= 0 )
22  {
23    qword_1400062D8 = (__int64)WdfFunctions->pfnWdfDriverMiniportUnload;
24    v7 = sub_140003DE0(&BindInfo);
```

חקירת הדרייבר

כעת אנחנו יכולים להתחיל לחקור, נפעיל את המכונה הוירטואלית שהכנתי עם דיבוג לקרנל מופעל, וברגע שנכנס נראה שהדרייבר לא פעיל, כפי שראינו השירות יתחיל לרוץ ויפעיל את הדרייבר כשנחבר מכשיר USB עם הפרמטרים המתאימים, בפונקציית ה-DriverEntry לא נמצא דבר מעניין במיוחד, אך תשומת הלב צריכה להיות מופנית לפונקציה שמוגדרת לאירוע EvtDeviceAdd. קל מאוד לפספס אותה אם לא נשים לב לקריאה ל-WdfDriverCreate. שבתוך האובייקט WDF_DRIVER_CONFIG נגדיר פונקציה שתקרא כל פעם ש-WDF ייצור מכשיר חדש:

```
kd> dt Wdf01000!_WDF_DRIVER_CONFIG poi(@rsp+20)
+0x000 Size              : 0x20
+0x008 EvtDriverDeviceAdd : 0xfffff807`4be48000     long  +0
+0x010 EvtDriverUnload   : (null)
+0x018 DriverInitFlags   : 0
+0x01c DriverPoolTag     : 0
```

כשנחבר את האייפון למחשב נראה שאכן נקראה הפונקציה, ה-PnP manager עובר על כל הדרייברים שיכולים להיות רלוונטים למכשיר ומפעיל את פונקציונליות ה-Add Device, במקרה שלנו הוא ייקרא לדרייבר של Wdf שיעביר את הבקשה לפונקציה שראינו. דבר ראשון שנבצע בפונקציה היא קריאה ל-WdfdoInitSetFilter מה שיזהה את הדרייבר כמסנן התחתון של המכשיר שהתחבר. ולאחר מכן נבצע את השלבים הבאים:

1. נגדיר פונקציית Callback ל-PnP and power management callback בשביל האירוע EvtDeviceD0Entry שאומר שנבצע פונקציה זו כשהמכשיר במצב D0.
2. נגדיר שתי פונקציות IRP Preprocess שבעצם יעבדו IRP's לפני שהם יעובדו על ידי WDF:
   - IRP_MJ_PNP
   - IRP_MJ_INTERNAL_DEVICE_CONTROL
3. ניצור Device Object חדש עם Type 56 (מספר המייצג IoType_File)

מכיוון שדרייבר בווינדוס לא יכול לתקשר ישירות עם USB נצטרך להשתמש ב-USB Request Block (URB) כדי לתקשר עם המכשיר ולכן נצטרך ליצור בקשות שייתאמו את המבנה של האובייקט. לאחר קישור פונקציות ה-Callback נגיע לפונקציה שאני קראתי לה UsbEvtDevicePrepareHardware_1820, הוספתי את ההיסט של כל פונקציה בסוף השם כדי שתוכלו לעקוב אם תרצו לראות את המחקר בעיניים.

שם אנחנו נקרא את ה-Descriptor מה-USB בצורה הבא, בפונקציה AllocateUSBDHandle_33A4 ניצור USBD Handle שישמש כדי להחליט על גרסת ה-USBDI (603 בשביל Windows 10 ומעלה, 602 בשביל Windows 8 ו-Server 2012 ו-1536 בשביל Windows 7). נקרא ל-IOCTL_INTERNAL_USB_SUBMIT_URB כדי ליצור את הבקשה וגם לקבל את הסטטוס שלה אחרי שהיא הסתיימה.

לאחר שהשגנו Handle ניצור בריכת זיכרון (pool) בקרנל באמצעות ExAllocatePoolWithTag עם התגית "APPL", ונשתמש בכתובת הזיכרון שהוקצתה בקריאה ל-AllocateURBBuffer_3844:

```
20  DidAllocateHandle = AllocateUSBDHandle_33A4(DeviceObject, TargetDeviceObject, 0x602u, 'LPPA', (__m128 **)&pUSBDHandle);
21  if ( DidAllocateHandle >= 0 )
22  {
23    DidAllocateHandle = AllocateURBBuffer_3844(pUSBDHandle, (__m128 **)&URB_HEADER);
24    if ( DidAllocateHandle >= 0 )
25    {
26      *((_WORD *)URB_HEADER + 1) = 11;          // UrbHeader.Function = URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE
27      *(_WORD *)URB_HEADER = 136;               // UrbHeader.Length   = 136 bytes
28      *((_DWORD *)URB_HEADER + 9) = 18;
29      *((_QWORD *)URB_HEADER + 6) = 0;
30      *((_QWORD *)URB_HEADER + 5) = ConfigBuffer;// output buffer
31      *((_BYTE *)URB_HEADER + 131) = 1;         // USBD_TRANSFER_DIRECTION
32      *((_BYTE *)URB_HEADER + 130) = 0;         // direction = IN
33      *((_WORD *)URB_HEADER + 66) = 0;
34      *((_QWORD *)URB_HEADER + 7) = 0;
35      DidAllocateHandle = SubmitUrbSynchronously_2270(Device, (__int64)pUSBDHandle, (__int64)URB_HEADER);
36    }
37  }
```

חשוב לציין איך ידעתי שהפונקציה שנקראת היא URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE, אם נסתכל בדוקומנטציה של מייקרוסופט לאובייקט URB_HEADER אין לנו את הקוד המקורי של הפונקציות, ולכן נצטרך לפתוח את הקובץ usb.h שיותקן על המחשב שלנו עם הורדת ערכת הפיתוח של ווינדוס. נפתח את הקובץ ונאתר את ההגדרות והחתימות של הפונקציות הרלוונטיות:

```
#define URB_FUNCTION_SELECT_CONFIGURATION 0x0000
#define URB_FUNCTION_SELECT_INTERFACE 0x0001
#define URB_FUNCTION_ABORT_PIPE 0x0002
#define URB_FUNCTION_TAKE_FRAME_LENGTH_CONTROL 0x0003
#define URB_FUNCTION_RELEASE_FRAME_LENGTH_CONTROL 0x0004
#define URB_FUNCTION_GET_FRAME_LENGTH 0x0005
#define URB_FUNCTION_SET_FRAME_LENGTH 0x0006
#define URB_FUNCTION_GET_CURRENT_FRAME_NUMBER 0x0007
#define URB_FUNCTION_CONTROL_TRANSFER 0x0008
#define URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER 0x0009
#define URB_FUNCTION_ISOCH_TRANSFER 0x000a
#define URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE 0x000b
#define URB_FUNCTION_SET_DESCRIPTOR_TO_DEVICE 0x000c
#define URB_FUNCTION_SET_FEATURE_TO_DEVICE 0x000d
```

ה-Descriptor שקיבלנו ייתן לנו מידע על המכשיר כמו מזהה היצרן, הפרוטוקול וה-Device class כמו שראינו קודם ב-Usbview, כשנחזור ל-UsbEvtDevicePrepareHardware_1820 נאחסן את הערך של bNumConfigurations באינדקס ה-16 של אובייקט ה-Context, נקרא למשתנה NumConfig שנקבל מהקריאה ל-WdfObjectGetTypedContextWorker.

האובייקט שקיבלנו מ-WdfObjectGetTypedContextWorker יצביע למרחב הזיכרון של ה-Device במקרה שלנו ולכל Device שניצור יהיה זיכרון משלו שניתן לגשת אליו מכל מקום בדרייבר אם נעביר לפונקציה את אותו WdfDevice. אם לפני כן ב-WDM היינו צריכים לאלקץ את ה-Context ידנית בכל קריאה ל-IoCreateDevice, דבר שלקה בבטיחות ופגיע למירב החולשות. היום אנחנו משתמשים ב-WDF שמבצע type-checking בזמן הידור ועוד שלל בדיקות בטיחות כמו נעילה אוטומטית של משאבים, הרצת פונקציות

Callbacks ברמת PASSIVE_LEVEL ב-(IRQL) Interrup Request Level כדי שלא נפריע ל-Interrupts מחומרה ונוכל לקרוא לפונקציות Windows-API בלי בעיות.

בחזרה לקוד, נוכל לראות שלאחר מכן נשווה אם הערך שנמצא ב-NumConfig שווה לאחד, אם הדבר נכון, נשנה את הערך הלוקאלי לאחד. אם לא, נשלח למכשיר בקשה לקבל את הקונפיגורציה המועדפת, אם נכשל בבקשה נשנה את הערך לשלוש ולאחר מכן נשנה את הערך שנמצא ב-NumConfig ל-PreferredConfig. הבדיקה הבאה תהיה אם השירותים של אפל מותקנים על המחשב ואם לא נשנה את הערך לאחד אחר כך נבדוק אם הקונפיגורציה המועדפת גדולה ממספר הקונפיגורציות המצוינות ב-Descriptor. ולבסוף אם הקונפיגורציה המועדפת קטנה מחמש נחזיר אותה. להלן הפסואדו-קוד:

```
  if ( *(_BYTE *)(DevCtx + 16) == 1 )
  {
    v3 = 1;
  }
  else
  {
    GotConfig = GetUSBConfig_1CC0(WdfDevice, (__int64)&PreferredConfig);
    v3 = PreferredConfig;
    if ( GotConfig < 0 )
      v3 = 3;
  }
  PreferredConfig = v3;
  *(_BYTE *)(DevCtx + 17) = v3;
  isAppleServicesRunning = IsAppleServiceRunning_1E44();
  v6 = v3;
  if ( !isAppleServicesRunning )
    v6 = 1;
  v7 = v6;
  if ( v6 >= *(_BYTE *)(DevCtx + 16) )
    v7 = *(_BYTE *)(DevCtx + 16);
  v8 = v7;
  result = 5;
  if ( (unsigned __int8)v8 < 5u )
    return v8;
  return result;
}
```

חשוב לציין ששדה ה-PreferredConfig לא מצוין ב-Descriptor אלא בזיכרון הלא נדיף של המכשיר ונשיג אותו באמצעות הפונקציה URB_FUNCTION_VENDOR_DEVICE מתוך GetUSBConfig_1CC0. נוסיף גם ש-GetUSBConfig_1CC0 ייצור בקשת control-transfer עם סוג בקשה 69 ביישום שלו לשליחת הבקשה (כנראה מספר פנימי של אפל).

מה שראינו עכשיו הוא למה למרות שקובץ ה-INF כותב את הערך 2 ל-OriginalConfigurationValue ברגיסטרי אם נחבר את האייפון למחשב ללא iTunes נראה ש-OriginalConfigurationValue יהיה שווה 0. וזה מכיוון שלאחר שניצור את הערך ברגיסטרי הפונקציה UsbEvtDevicePrepareHardware_1820 תקרא ל-WdfUsbTargetDeviceCreate כדי ליצור אובייקט USB שייצג את מכשיר ה-USB שאיתו יתקשר הדרייבר.

בשביל לבדוק את הקונפיגורציה המועדפת הפונקציה תיצור WDFTIMER שייקרא כל חמש שניות כדי לבדוק אם משהו השתנה, ואם משהו השתנתה, נקרא לפונקציה WdfUsbTargetDeviceCyclePortSynchronously בשביל להפרד מאובייקט ה-USB הנוכחי ולטעון אותו עם הקונפיגורציה החדשה.

יישום Picture Transfer Protocol

השלב הבא הוא להסתכל על שתי הפונקציות שהגדרנו לקריאות IRP Pre-Process שאלו קריאות שייבצעו פעולות על ה-IRP לפני שהוא עובר דרך הדרייברים האחרים.

נתחיל עם הפונקציה שתנהל את הפעולה IRP_MJ_INTERNAL_DEVICE_CONTROL, שתבדוק אם ה-IRP הוא בקשת IOCTL_INTERNAL_USB_SUBMIT_URB כדי להשיג את הקונפיגורציה של ה-USB ואם זה נכון נעביר את ה-IRP בעזרת הפונקציה IoForwardIrpSynchronously לדרייבר הבא. ולאחר מכן נקבל pipe handle שיעביר דרכו בקשות Interrupt, BulkIn ו-BulkOut שיאוחסנו ב-Device Context.

עכשיו נעבור לפונקציות ה-Callback לשני המקרים, IRP_MJ_QUERY_DEVICE_RELATIONS ו-IRP_MN_QUERY_ID. בפונקציה שתנהל את IRP_MN_QUERY_ID ניקח את ערך הקונפיגורציה שקיבלנו לפני כן ואם הוא שווה לאחד נוסיף את המחרוזת &RESTORE_MODE בשביל מידע שיחזור לקריאות BusQueryDeviceID ו-BusQueryHardwareIDs.

נעבור לפונקציה הבא, שתקרא FilterEvtQueryDeviceRelations_1050 תתבצע רק אם השירות של אפל רץ על מכשירינו וה-PtpTimer עוד לא התחיל לרוץ, מה שייקרא לקראת סוף הפונקציה שייתבצע כל 5 שניות ויאחסן משתנה ב-Context שיסמן שהקריאה ל-QueryDeviceRelations לא תעובד יותר בעזרת פונקציית ה-Callback המוגדרת. נחזור לתנאי, אם יש לנו את שירותי אפל וגם הטיימר לא רץ על המכשיר נעביר בקשות BusRelations באופן סינכרוני לדרייברים הבאים, ואם הדרייבר ייראה שהבקשה הצליחה, ושיש מידע שחזר, הדרייבר יסתכל ברשימת המכשירים על מכשירים בעלי ה-CompatibleID USB\Class_06 ואם אכן כן הדרייבר יפענח את אובייקט ה-Device Object של אותו מכשיר, יוריד אותו מהרשימה ויעדכן את כמות המכשירים. כשהמכשיר בעל USB\Class_06 נמצא ו-iTunes מותקן מתחיל הטיימר שהזכרתי קודם הפסקה שאחראי השהיית הראות של מכשיר ה-Windows Portable Device (WPD). אנו נצטרך להשהות את גילוי המכשיר כדי לוודא שהדרייבר אכן זיהה את המכשיר וחלון לשלוח פקטות PTP/MTP למכשיר.

בשביל לבחון פקטות אלו אוריד את תוכנת Wireshark עם התוסף USBPcap ואראה את פתיחת החיבור עם המכשיר:

No.	Time	Source	Destination	Protocol	Length	Operation Code	Transaction ID	Info
4309	19.690260	host	3.4.1	USB	27			URB_BULK in
4310	19.693115	3.4.1	host	USB-PTP	39		0x00000013	RSP 00000013 (2001) OK
4151	19.416661	host	3.4.2	USB-PTP	39	0x1001	0x00000001	CMD 00000001 (1001) GetDeviceInfo
4154	19.426424	3.4.1	host	USB-PTP	39	0x1001	0x00000001	DAT 00000001 (1001) GetDeviceInfo
4124	19.230767	host	3.4.2	USB-PTP	43	0x1002	0x00000000	CMD 00000000 (1002) OpenSession
4146	19.413397	host	3.4.2	USB-PTP	43	0x1002	0x00000000	CMD 00000000 (1002) OpenSession
4132	19.248273	host	3.4.2	USB-PTP	39	0x1003	0x00000002	CMD 00000002 (1003) CloseSession
4295	19.672592	host	3.4.2	USB-PTP	39	0x1004	0x00000012	CMD 00000012 (1004) GetStorageIDs
4298	19.674522	3.4.1	host	USB-PTP	39	0x1004	0x00000012	DAT 00000012 (1004) GetStorageIDs
4303	19.681428	host	3.4.2	USB-PTP	43	0x1005	0x00000013	CMD 00000013 (1005) GetStorageInfo
4306	19.686227	3.4.1	host	USB-PTP	39	0x1005	0x00000013	DAT 00000013 (1005) GetStorageInfo
4128	19.240847	host	3.4.2	USB-PTP	43	0x9008	0x00000001	CMD 00000001 (9008) UNKNOWN
4163	19.453159	host	3.4.2	USB-PTP	43	0x9801	0x00000002	CMD 00000002 (9801) UNKNOWN Undefined
4166	19.454995	3.4.1	host	USB-PTP	39	0x9801	0x00000002	DAT 00000002 (9801) UNKNOWN
4171	19.461723	host	3.4.2	USB-PTP	43	0x9801	0x00000003	CMD 00000003 (9801) UNKNOWN Association
4174	19.464646	3.4.1	host	USB-PTP	39	0x9801	0x00000003	DAT 00000003 (9801) UNKNOWN
4179	19.471066	host	3.4.2	USB-PTP	43	0x9801	0x00000004	CMD 00000004 (9801) UNKNOWN Script
4182	19.476038	3.4.1	host	USB-PTP	39	0x9801	0x00000004	DAT 00000004 (9801) UNKNOWN

הקודים הרלוונטים הם:

1. 0x1002 - OpenSession
2. 0x1003 - CloseSession
3. 0x1004 - GetStorageIDs

אנו לא צריכים Session כדי לקרוא לבקשות כמו GetStorageIDs ולכן החיבור נפתח ונסגר.

נראה בנוסף את הקודים 0x9801 ו-0x9008 שלא מזוהים על ידי התוכנה אך אם נלך לאתר נראה ש-0x9801 הוא הרחבה של מייקרוסופט ל-PTP לפי האתר של Wireshark אך 0x9008 כנראה קוד פנימי של אפל:

```
/* Microsoft / MTP extension codes */
PTP_OC_MTP_GetObjectPropsSupported              = 0x9801,
PTP_OC_MTP_GetObjectPropDesc                    = 0x9802,
PTP_OC_MTP_GetObjectPropValue                   = 0x9803,
PTP_OC_MTP_SetObjectPropValue                   = 0x9804,
PTP_OC_MTP_GetObjPropList                       = 0x9805,
PTP_OC_MTP_SetObjPropList                       = 0x9806,
PTP_OC_MTP_GetInterdependendPropdesc            = 0x9807,
PTP_OC_MTP_SendObjectPropList                   = 0x9808,
PTP_OC_MTP_GetObjectReferences                  = 0x9810,
PTP_OC_MTP_SetObjectReferences                  = 0x9811,
PTP_OC_MTP_UpdateDeviceFirmware                 = 0x9812,
PTP_OC_MTP_Skip                                 = 0x9820,
```

ולאחר ששלחנו את פקטות ה-PTP/MTP ה-PtpTimer ייקרא ל-IoInvalidateDeviceRelations עם הסוג BusRelation שיופעל כשבא IRP חדש מסוג QueryDeviceRelations אבל משום שהטיימר כבר בוצע לא נוריד את מכשיר ה-WPD מהרשימה ולכן ה-PnP manager ייראה את ה-Physical Device Object (PDO) בשביל מכשיר ה-WPD וייבנה עבורו היררכיית דרייברים.

עד כה על הדרייבר AppleLowerFilter, ראינו איך הדרייבר מנטר בקשות שעוברות בקרנל ומוסיף להם פונקציונליות כמו לשלוח פקטה שתגיש את הקונפיגורציה המועדפת של המכשיר שחיברנו בעזרת פקטות URB ואיך אם הורדנו את שירותי אפל הקונפיגורציה תהיה שונה. הוספנו תמיכה בבקשות כמו BusQueryHardwareIDs ו-BusQueryDeviceID לפי הקונפיגורצייה שקיבלנו.

הפתח לשכבת ה-USER MODE

הדרייבר AppleLowerFilter הוא לא הדרייבר היחיד שמותקן ביחד עם iTunes אך הוא הכי משמעותי בהקשר של תקשורת USB, בפרק זה נראה את התוכנות הנוספות שמרחיבות את מרחב הפעולה שלנו עם האייפון. כפי שראינו קודם, ברגע שנתקין את תוכנת iTunes נוכל לבחור קונפיגורציית USB שונה. בלי iTunes אנחנו מוגבלים לקונפיגורציה מספר אחד, וכש-iTunes מותקן נלך לקונפיגורציה מספר שלוש כברירת מחדל ששם הדרייבר usbccgp ייצור את מזהה המכשיר במבנה הבא: USB\VID_05ac&PID_yyyy&MI_01. נראה שבקובץ appleusb.inf קיים מתאים למבנה זה של מכשיר ולכן נטען את הדרייבר AppleKmdfFilter ודרייבר ה-AppleUsbFilter UMDF:

```
[AppleUsbMux_Install]
Include=winusb.inf
Needs=WINUSB.NT
CopyFiles=AppleUsbFilter_CopyFiles
CopyFiles=AppleKmdfFilter_CopyFiles
FeatureScore=0x7F

[AppleUsbFilter_CopyFiles]
AppleUsbFilter.dll

[AppleKmdfFilter_CopyFiles]
AppleKmdfFilter.sys
```

AppleKmdfFIlter הוא דרייבר מאוד קטן קטן שיישמש את אפליקציות ה-User mode לתקשורת עם הקרנל והעברת מידע ל-USB בעזרת I/O control codes (IOCTLs), שיעבירו את הבקשות להיררכיית הדרייברים. לדוגמה התמונה המוצרפת תראה קוד שייבדוק אם קוד ה-IOCTL, שהעברנו תקף ויעביר את הבקשה לדרייברים שמתחתיו:

```
        if ( IoctlCode == 2228257
          || IoctlCode == 2228261
          || IoctlCode == 2228265
          || IoctlCode == 2228269
          || IoctlCode == 2228273
          || IoctlCode == 2228277
          || IoctlCode == 2228281
          || IoctlCode == 2228285 )
        {
          CurrentStackLocation = *(__int128 **)(((__int64 (__fastcall *)(PWDF_DRIVER_GLOBALS, __int64))WdfFunctions->pfnWdfRequestWdmGetIrp)(
                                                  WdfDriverGlobals,
                                                  Request)
                                                + 0xB8);
          v6 = ((v4 - 2228257) & 0xFFFFFFFC) + 32;
LABEL_15:
          CustomIoctlCalc = v6 | 0x220002;
LABEL_43:
          Stack = *CurrentStackLocation;
          MinorFunction = CurrentStackLocation[1];
          Flags = CurrentStackLocation[2];
          Control = CurrentStackLocation[3];
          DeviceObject = *((_QWORD *)CurrentStackLocation + 8);
          DWORD2(MinorFunction) = CustomIoctlCalc;
          v29 = DeviceObject;
          ((void (__fastcall *)(PWDF_DRIVER_GLOBALS, __int64, __int128 *))WdfFunctions->pfnWdfRequestWdmFormatUsingStackLocation)(
            WdfDriverGlobals,
            Request,
            &Stack);
LABEL_57:
          IoTarget = ((__int64 (__fastcall *)(PWDF_DRIVER_GLOBALS, __int64))WdfFunctions->pfnWdfDeviceGetIoTarget)(
                        WdfDriverGlobals,
                        WDFDEVICE);
          return SendWDFIOToObject_1514(Request, IoTarget);// Sends a WDF I/O request (WDFREQUEST) to an IO target
        }
LABEL_56:
        ((void (__fastcall *)(PWDF_DRIVER_GLOBALS, __int64))WdfFunctions->pfnWdfRequestFormatRequestUsingCurrentType)(
          WdfDriverGlobals,
          Request);                          // formats a specified I/O request so the driver can forward it
        goto LABEL_57;
      }
```

בפסאודו קוד שצירפתי אפשר לראות שנשווה קבוצה של IOCTLs ואם אחד מהם מתאים, נכנס לבלוק שיכיל מספר תוויות שיהוו מקום ספציפי בקוד שנוכל לקרוא רק לו בעזרת קריאה ל-goto שייחשב את השדה CurrentStackLocation במבנה ה-IRP ביחד עם השדות הבנים שלו ויעביר את הבקשה באמצעות הפונקציה SendWDFIOToObject_1514 לדרייבר הבא. נוכל לראות גם שאם ידובר בבקשת קריאה או כתיבה. בבקשה לכתיבה נקבל Handle לאובייקט WDFMEMORY שייצג את הזיכרון שקיבלנו בבקשה באמצעות WdfRequestRetrieveInputMemory, נשיג את אובייט ה-I/O Target הרלוונטי, נשנה את הבקשה שתתאים לפורמט של WDF ולבסוף נשלח אותה עם WdfRequestSend.

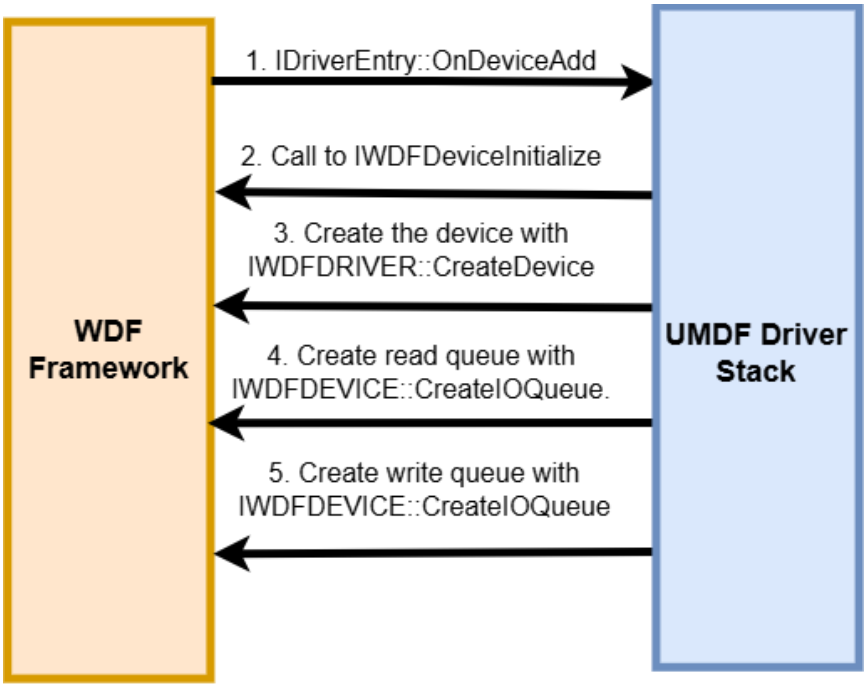
תקשורת עם אייפון מה-User Mode

נצטרך לבדוק את ערכי הקונפיגורציה שה-PnP manager יודע על המכשיר, ונקבל את השדה הבא: "DEVPKEY_Device_DriverInfPath String wpdmtp.inf", הקובץ wpdmtp.inf אחראי להפעיל תמיכה לפרוטוקול MTP ובשונה מהקובץ appleusb.inf הוא ייטען את הדרייברים מסוג User-Mode Driver Framework (UMDF) המתאימים לפי ה-CompatibleID USB\Class_06&SubClass_01&Prot_01, שלושת הקבצים העיקריים שנסתכל עליהם הם WpdMtp.dll שאחראי על הפקת תשובות לפקודות שנשלחו ושליחת בקשות כמו GetDeviceInfo כחלק מהפרוטוקול. WpdMtpUS.dll שאחראי על ניהול ה-pipes שייעברו לדרייבר winusb. ו-WpdMtpDr.dll שייבצע איתחול למשתני ה-WPD ויפענח IOCTLs שיקבל מהאפליקציות השונות, הדרייבר הכי רלוונטי ביישום הפרוטוקול.

אם שמתם לב, הקבצים האחראים על MTP בווינדוס הם דרייברים הפועלים בשכבת ה-User-Mode, אבל למה קוראים להם דרייברים? הרי דרייבר אמור לרוץ בקרנל ולתקשר עם החומרה, וכאן מדובר בקובץ DLL רגיל. ניתן לראות זאת כתהליך User-Mode שיוצר אובייקטי Device וירטואליים ב-Device Manager, מטפל בבקשות PnP דרך ממשקי COM ייעודיים. הכל בתיווך ה-Reflector WudfRd.sys שנמצא בקרנל, ומתקשר עם השירות WudfHost.exe המארח את כל הדרייברים של UMDF ומבטיח בידוד בין דרייברים שונים, תוך שימוש במנגנוני IPC מתקדמים להעברת הודעות בין שכבות. לדוגמה כשנחבר את האייפון למחשב שלנו נראה את WudfHost.exe מתחיל לפעול עם מספר דרייברים:

Name	Base address	Size	Description
WUDFHost.exe	**0x7ff6318...**	**320 kB**	**Windows Driver Foundation - User-mode Driver Framew...**
combase.dll	0x7fff3217...	3.33 MB	Microsoft COM for Windows
devobj.dll	0x7fff308f...	204 kB	Device Information Set DLL
rpcrt4.dll	0x7fff3345...	1.12 MB	Remote Procedure Call Runtime
sechost.dll	0x7fff3206...	636 kB	Host for SCM/SDDL/LSA Lookup APIs
WUDFPlatform.dll	0x7fff2f8f0...	200 kB	Windows Driver Foundation - User-mode Platform Library
AppleUsbFilter.dll	0x7fff28c2...	120 kB	Apple Mobile Device USB Device
dbghelp.dll	0x7fff22ca...	2 MB	Windows Image Helper
kernel32.dll	0x7fff31a9...	776 kB	Windows NT BASE API Client DLL
ntdll.dll	0x7fff335b...	1.97 MB	NT Layer DLL
PortableDeviceClassExtension.dll	0x7fff28a7...	144 kB	Windows Portable Device Class Extension Component
PortableDeviceTypes.dll	0x7fff2e13...	200 kB	Windows Portable Device (Parameter) Types Component
WpdMtp.dll	0x7fff28a3...	244 kB	MTP core protocol component
WpdMtpDr.dll	0x7fff1c67...	932 kB	Windows Portable Device Media Transfer Protocol Driver
WpdMtpUS.dll	0x7fff2087...	180 kB	Usbscan transport layer for MTP driver
WUDFx.dll	0x7fff2070...	604 kB	WDF:UMDF Framework Library
WUDFx02000.dll	0x7fff2092...	748 kB	WDF:UMDF Framework Library

לאחר שהבנתי שפיענוח אובייקטי ה-COM לא יהיה יעיל חיפוש קצר ברשת יוביל אותנו לקוד של WpdWudfSampleDriver שמהווה בסיס לאותו סוג דרייבר שעזר לי להבין את המבנה של הדרייבר ולספק את התרשים הבא:



אפשר לראות שנקודת ההתחלה שלנו היא OnDeviceAdd שבה ניצור אובייקט CDevice שייקח אותנו לפונקציה CDevice::OnPrepareHardware ששם נבצע אתחול לאובייקט WpdBaseDriver בעזרת הפונקציה WpdBaseDriver::Initialize, מפה נעבור לקוד שלא נמצא ברשת ונראה שנעבור ל-WpdMtp שייכלול את המחלקה CMtpDevice שתייצג מכשיר בספרייה ולאחר מכן נתחיל את התקשורת עם המכשיר בעזרת פרוטוקול MTP. לא אכנס לאיך הוא בדיוק עובד מכיוון שנושא זה יכול להיות מאמר בפני עצמו.

אם נסתכל בתמונה נראה גם את דרייבר ה-UMDF של אפל AppleUsbFilter שיהיה אחראי על תקשורת עם הדרייבר הפונקציונלי WinUSB שמספק לנו קריאות נוחות לתקשורת עם מכשירי USB כמו Initialize_WinUsb שתיצור עבורנו DeviceHandle למכשירי USB. הדרייבר משומש כדי ליישם את הפרוטוקול הקנייני של אפל שנחקר לעומק בפרויקט libimobiledevice.

Apple Mobile Device Service

נראה ש-iTunes מורידה את השירות Apple Mobile Device Service ואם ננסה להשבית אותו, לפתוח את iTunes ולחבר את האייפון, iTunes יציגו בפנינו התראה שתבקש להדליק את פעולת השירות ורק איתו נוכל לראות ולתקשר עם ספריית השירים בטלפון, לפי האתר theapplewiki.com נראה שהפרוטוקולים הקיימים בתקשורת בין מחשב ווינדוס או מק ל-iPhone הם בהיררכייה הבאה, מההכי נמוך לגבוה ביותר:

- פרוטוקול USB
- פרוטוקל usbmux שישמש ל-USB multiplexing והשירות שמיישם אותו ויפתח TCP socket לתקשורת עם האייפון
- פרוטוקול lockdownd
- iTunesHelper
- Apple File Conduit - שירות באייפון שייתן לנו לגשת לתמונות ואפליקציות מסוימות.

נסתכל על השירות ונראה את הספריות הבאות שהוא משתמש בהן:

- MobileDevice.dll - תבצע את האימות עם המכשיר, פונקציה לדוגמה היא AMDeviceExtendedPairWithOptions_ B4F5 שתבצע אימות של המפתחות שהועברו.
- CoreFoundation.dll - ספרייה שמכילה את הקוד של CoreFoundation, אפשר להקביל אותה ל-Lib C בלינוקס או ל-C Runtime בווינדוס.
- AppleMobileDeviceService_main.dll - ספרייה שתפתח socket בפורט 27015 ותאזין לבקשות usbmux.

בשביל לתקשר בעזרת הפרוטוקול usbmux וה-Socket שמאזין על המחשב נצטרך לפרמט את הבקשות שלנו לפורמט ה-plist של אפל כמו שכתבתי בסקריפט הפייתון הבא:

```
def send_usbmux_plist(sock, obj, tag):
    """Send a plist inside a usbmuxd packet frame."""
    payload = plistlib.dumps(obj, fmt=plistlib.FMT_BINARY)
    header = struct.pack('<IIII', len(payload) + 16, 1, 8, tag)
    sock.sendall(header + payload)
    print(f" Sent {obj.get('MessageType', '?')} (tag={tag}, {len(payload)}
bytes)")

def recv_usbmux_plist(sock):
    """Receive one usbmuxd plist frame and decode it."""
    header = sock.recv(16)
    if not header:
        raise ConnectionError("socket closed")
    if len(header) < 16:
        raise RuntimeError("incomplete header")

    length, version, msg_type, tag = struct.unpack('<IIII', header)
    payload_len = length - 16
    data = bytearray()
    while len(data) < payload_len:
        chunk = sock.recv(payload_len - len(data))
        if not chunk:
            raise ConnectionError("socket closed while reading payload")
        data.extend(chunk)

    try:
        plist = plistlib.loads(data)
    except Exception:
        plist = {"_raw": data.hex()}
    return plist, tag
```

כדוגמה נשלח את הפקודה ListDevices בעזרת הקוד הבא ונקבל את המכשירים המחוברים למחשב:

```
with socket.create_connection((HOST, PORT)) as s:
    s.settimeout(180)
    tag = 1

    list_req = {
        "MessageType": "ListDevices",
        "ClientVersionString": "PythonClient",
    }
    send_usbmux_plist(s, list_req, tag)
    reply, rtag = recv_usbmux_plist(s)
    print(f"\n Reply to tag {rtag}:")
    pretty(reply)

    tag += 1
    listen_req = {
        "MessageType": "Listen",
        "ClientVersionString": "PythonClient",
    }
    send_usbmux_plist(s, listen_req, tag)
    print("\n=== Waiting for device events ===")
    start = time.time()
    while time.time() - start < 20:
        try:
            msg, rtag = recv_usbmux_plist(s)
            print(f"\n Incoming packet (tag={rtag}):")
            pretty(msg)
        except socket.timeout:
            print("...still waiting for events...")
            continue
        except Exception as e:
            print("Connection ended:", e)
            break
```

ולהלן התוצאה:

```
Incoming packet (tag=2):
{
        "MessageType": "Result",
        "Number": 0
}

Incoming packet (tag=0):
{
        "DeviceID": 1,
        "MessageType": "Attached",
        "Properties": {
        "ConnectionType": "USB",
        "DeviceID": 1,
        "LocationID": 0,
        "ProductID": 4773,
        "SerialNumber": "XXXXXXXX-YYYYYYYYYYYYYYYY"
}
```

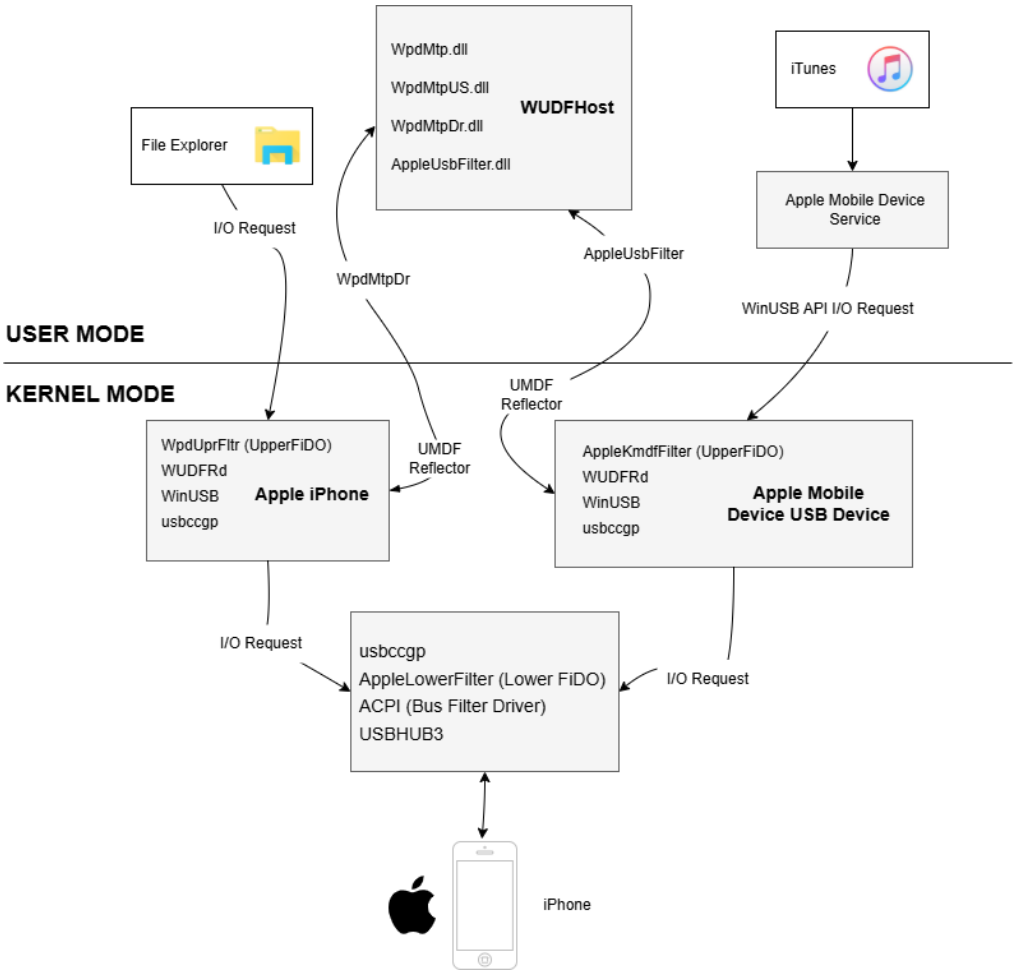
כשנהנדס לאחור את הקוד, נראה ש-AppleMobileDeviceService_main בודק את המחרוזת לפי CFDictionary ומבצע השוואה בין מחרוזות מוגדרות מראש עם תוכן ההודעה בפונקציה שנקראת HandleMessageTypeCommand_867C:

```
if ( CFStringCompare(MessageType_Value, ConnectString, 0) )
{
  if ( CFStringCompare(MessageType_Value, ListenString, 0) )
  {
    if ( CFStringCompare(MessageType_Value, LogsString, 0) )
    {
      if ( CFStringCompare(MessageType_Value, ListDevicesString, 0) )
      {
        if ( CFStringCompare(MessageType_Value, ListListenersString, 0) )
        {
          if ( CFStringCompare(MessageType_Value, ReadBUIDString, 0) )
          {
            if ( CFStringCompare(MessageType_Value, ReadPairRecordString, 0) )
            {
              if ( CFStringCompare(MessageType_Value, SavePairRecordString, 0) )
              {
                if ( CFStringCompare(MessageType_Value, DeletePairRecordString, 0) )
                {
                  return 1;
                }
                else
                {
                  DeletePairRecord(usbmuxClient, dictionary);
                  CheckIfPairingChanged(v30, v29);
                  return 0;
                }
              }
              else
              {
                SavePairRecord(usbmuxClient, dictionary);
                CheckIfPairingChanged(v28, v27);
                return 0;
              }
            }
            else
            {
              ReadPairRecord(usbmuxClient, dictionary);
              return 0;
            }
```

סיכום

עברנו על סמך מה אילו דרייברים נטענים במערכת ההפעלה ווינדוס כשנחבר אייפון, לאחר מכן למדנו את סוגי השדות השונים שיש בפרוטוקול USB ואיך הם משפיעים על ההתקשרות שלנו עם האייפון, ראינו איך נדבר עם המכשיר בעזרת פקטות URB בשכבת הקרנל ונתנהג שונה לפי גרסת הקונפיגורציה שנקבל בעזרת הדרייבר AppleLowerFilter, ראינו איך הוא מפעיל את פרוטוקול PTP באייפון כדי שיוכל לתקשר בקודים שאפל הגדירו.

מצאנו ש-AppleKmdfFilter נועד בשביל לפתוח את התקשורת עם הקרנל לאפליקציות משכבת ה- User Mode בעזרת IOCTL. למדנו מה הם דרייברים שנמצאים בשכבת ה-User Mode ואיך הם מועילים לנו כשתהיה לנו התנהגות שונה אם תוכנות אפל מותקנות, לסיום אציג את הדיאגרמה שמסכמת הכל ומראה את התהליך שנעבור בין אם נפתח את תוכן האייפון ב-File Explorer או ב-iTunes.



על המחבר

אני ליאל חנוכוב, חוקר חולשות בעל תשוקה לטכנולוגיות שמעורבות בחיי היום-יום שלנו כדי להפוך את העולם לבטוח יותר, תחומי המחקר שלי כוללים בין היתר אבטחה של מערכות הפעלה ומערכות האינטרנט של הדברים (IoT).

אשמח לענות על שאלות ולקבל משוב :)

https://www.linkedin.com/in/lielh/

מקורות מידע

- https://www.keil.com/pack/doc/mw/usb/html/
- https://learn.microsoft.com/en-us/windows-hardware/drivers/wdf/
- https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/introduction-to-wdm
- https://github.com/thalium
- https://github.com/ReverseWarrior/ida_kmdf
- https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/usbview
- https://github.com/microsoft/Windows-Driver-Frameworks
- https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdfdriver/nc-wdfdriver-evt_wdf_driver_device_add
- https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdfdriver/nf-wdfdriver-wdfdrivercreate
- https://learn.microsoft.com/en-us/windows-hardware/drivers/install/pnp-manager
- https://github.com/microsoft/Windows-driver-samples/tree/win11-22h2/wpd/WpdWudfSampleDriver
- https://theapplewiki.com/wiki/AFC
- https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/device-power-states
- https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/ns-wdm-_device_object
- https://learn.microsoft.com/en-us/windows-hardware/drivers/usbcon/communicating-with-a-usb-device
- https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdfobject/nf-wdfobject-wdfobjectgettypedcontextworker
- https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdfusb/nf-wdfusb-wdfusbtargetdevicecycleportsynchronously

- https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdfrequest/nf-wdfrequest-wdfrequestretrieveinputmemory
- https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/wdfrequest/nf-wdfrequest-wdfrequestsend
- https://learn.microsoft.com/en-us/windows-hardware/drivers/wdf/overview-of-the-umdf
- https://libimobiledevice.org/
- https://learn.microsoft.com/en-us/windows/win32/windows-portable-devices
- https://github.com/microsoft/Windows-driver-samples/blob/win11-22h2/wpd/WpdWudfSampleDriver/Driver.cpp#L18
- https://github.com/microsoft/Windows-driver-samples/blob/win11-22h2/wpd/WpdWudfSampleDriver/Device.cpp#L80
- https://github.com/microsoft/Windows-driver-samples/blob/win11-22h2/wpd/WpdWudfSampleDriver/WpdBaseDriver.cpp#L152
- https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/types-of-wdm-device-objects
- https://learn.microsoft.com/en-us/windows-hardware/drivers/usbcon/usb-common-class-generic-parent-driver