Erledigt "Du hast ja Alles" - hmmm vielleicht, wenn ich einen Laptop habe.

Beitrag von „Brumbaer“ vom 26. April 2018, 02:11

AppleUSBNCMControl
Ist das eine Drohung oder eine Versprechen, Erlösung oder Verdammnis, Peek oder Cloppenburg ?
Bei dem Namen liegt die Verbindung nahe, dass der Treiber in AppleUSBNCMIrgendwas.kext zu finden ist.
Es stellt sich heraus es gibt tatsächlich ein Kext und sein Irgendwas ist Nichts.
"AppleUSBNCMNichts.kext, sehr lustig, ha, ha, ha, was haben wir wieder gelacht".
Das *AppleUSBNCM.kext* hat Alles was man erwartet u.a. eine Info.plist und eine Programmdatei names *AppleUSBNCM*.

Persönchen
In der Info.plist schauen wir uns die *IOKitPersonalities* an, denn die sind der Weg zum Herzen des Kextes.

IOKitPersonalities	Dictionary	(3 items)
AppleUSBNCMControl	Dictionary	(5 items)
IOClass	String	AppleUSBNCMControl
IOProviderClass	String	IOUSBHostInterface
bInterfaceClass	Number	2
bInterfaceProtocol	String	*
bInterfaceSubClass	Number	13
AppleUSBNCMData0	Dictionary	(5 items)
IOClass	String	AppleUSBNCMData
IOProviderClass	String	IOUSBHostInterface
bInterfaceClass	Number	10
bInterfaceProtocol	Number	1
bInterfaceSubClass	Number	0
AppleUSBNCMData13	Dictionary	(5 items)
IOClass	String	AppleUSBNCMData
IOProviderClass	String	IOUSBHostInterface
bInterfaceClass	Number	10
bInterfaceProtocol	String	1
bInterfaceSubClass	Number	13

Es gibt drei Persönchen, die einen von zwei Treibern laden. Entweder *AppleUSBNCMControl* oder *AppleUSBData*.
Bei allen ist die *IOProviderClass IOUSBHostInterface*. D.h. sie hängen sich an ein USB-interface. An welches legen bInterfaceClass, bInterfaceSubClass und bInterfaceProtocol fest. Sie entsprechen der Klasse, Unterklasse und dem Protokoll im USB-Interface-Decriptor.

Wer mit wem
Nach einem Neustart wird die Konfiguration des USB Gerätes auf 1 gesetzt und alle Interface Alternativen 0 ausgewählt. Davon ausgehend, dass niemand was daran geändert hat, hätten wir bel L831-EAU
Interface 0 mit Klasse 2, Unterklasse 13 und Protokoll 0.
Interface 1 mit Klasse 10, Unterklasse 0 und Protokoll 1.

AppleUSBNCMControl testet auf Klasse 2, Unterklasse 13 und Protokoll * (egal) und hängt sich deshalb an Interface 0.
AppleUSBNCMData0 testet auf Klasse 10, Unterklasse 0 und Protokoll 1 und hängt sich deshalb an Interface 1.
AppleUSBNCMData13 testet auf Klasse 10, Unterklasse 13 und Protokoll 1 und hängt sich deshalb an kein Interface, denn wir haben keins mit solch einem Tripel.

Dazu hätte wir die *Info.plist* nicht aufmachen müssen, das zeigt uns auch der *IORegistryExplorer*. Allerdings kann uns der *IORegistryExplorer* nicht zeigen, was nicht geladen wurde.
Der Eintrag für *AppleUSBNCMControl* hat kein *CFBundleIdentifier* Feld, also ist der Treiber in der Programmdatei dieses Kextes gespeichert.

Not Cool Man
könnte sein, mit NCM ist an dieser Stelle aber vermutlich *Network Control Model* gemeint. NCM ist eine Unterklasse der USB CDC (*Communication Device Class*) Klasse. Auf USB.org kann man die entsprechenden Spezifikationen downloaden.
In Kürze NCM unterscheidet Daten und Kontroll-Funktionen (Interfaces). Die Daten Funktionen empfangen und senden Pakete. Diese Pakete bestehen aus den Paketen, so wie sie über das Ethernet verschickt werden würden, ergänzt um Informationen, die dem Modem mitteilen, was und wieviel es da bekommt. Wenn man Pakete in Pakete packt, so nennt man das einbetten.

Wie man sich einbettet, so ....
Der Kontroll-Kanal kann ein *Embedded Request and Response Protocol* unterstützen.
Man schickt einen besonderen *Request* an den *Control-Endpoint* (An letztes Mal erinnern: Control Endpoint, Requests - Befehl mit optionalen Daten vom oder zum USB Gerät, schon wieder vergessen ?). Die Daten des *Requests* sind aber keine Daten, sondern ein Befehl, eine Aufforderung (*Request*) an das Modem. D.h. der *Request* an das Modem ist in den Daten des *Requests* an das USB-Gerät eingebettet.
Hat das Modem den Befehl ausgeführt, so schickt es von einem *Interrupt-Endpoint* eine Nachricht. Empfängt der Treiber die Nachricht schickt er einen *Request* an den *Control-Endpoint* , er möchte doch mit den Daten - meist die Antwort (*Response*) auf einen *Request* an das Modem - rausrücken. Was der dann auch tut indem er sie in die Daten des Requests einbettet.
Über den *Interrupt-Endpoint* wird auch eine Nachricht geschickt, wenn das Modem etwas, z.B. dass sich die Empfangsstärke geändert hat, mitzuteilen hat.
Wir erinnern uns an letztes mal, die Interface Deskriptoren für Interface 0 ? Nur einen Endpoint, einen Lese-Interrupt-Endpoint. Also genau das was man für dieses Verfahren braucht.

Dann wollen wir das Ding doch mal eintüten oder besser einbetten.

Schwierige Entscheidung

Nun kann man entweder einen neuen Treiber schreiben oder das was Apple anbietet verwenden. Wäre schon schön, wenn man das nicht neu schreiben müsste,
Mac Treiber werden in C++ geschrieben und die Treiber sind C++ Klassen, die von der Klasse IOService erben.
Im Idealfall hat man eine Headerdatei mit der Klassendefinition. Natürlich gibt es keine Headerfiles oder sonst eineDokumentation zu AppleUSBNCM. Apple stellt allerdings die Quellen zu AppleUSBCDC der Superklasse von AppleUSBNCM, zur Verfügung. Allerdings in der Version 4.5. Inzwischen sind wir bei 5.0 und die verwendeten USB Interface- und Device-Klassen haben sich geändert.
Aber wir haben ja eine ausführbare Datei, die unsere Klasse enthält. Man kann nun aus der ausführbaren Datei die Informationen gewinnen, die man zur Erstellung einer Headerdatei benötigt.
Einige recht einfach, andere mit Aufwand.
Denen, die interessiert wie das geht, sei gesagt, dass das den Rahmen dieser Berschreibung sprengen würde, es ist genug Stoff für eine eigene Artikelserie. Es sei nur gesagt, dass Hopper eine große Hilfe dabei ist und dass er einen Python Interpreter enthält mit dem einige Abläufe automatisieren kann.
Bei meinen Reisen durch AppleUSBNCM kam ich an eine Stelle an der ich einen Funktionsnamen gebraucht hätte, den Hopper auch brav ausgab, der aber nicht über Python abzufragen war. Eine Anfrage beim Entwickler bescherte mir 3 Tage später eine Hopper Version mit der das möglich war - wow, besser, WOW.

So nicht

Nachdem die AppleUSBNCM und AppleUSBCDC (Superklasse von AppleUSBNCM) soweit analysiert waren, dass ich wusste, was sie warum taten, versuchte ich eine Kommunikation mit dem Modem herzustellen.
Ich konnte zwar die Embedded Commands senden, aber es gab nie eine Rückmeldung vom Interrupt Endpoint.
Also habe ich einen neuen Treiber geschrieben. für den Fall, dass ich bei der Analyse von AppleUSBNCM etwas übersehen habe. Nein, habe ich
wohl nicht.
Also nochmals in die USB Deskriptoren geschaut, ob es denn einen Hinweis auf eine andere Zugriffsvariante gibt, und siehe da: Es gibt keinen.

Wer lesen kann, ist klar im Vorteil

Wir hatten gesehen, dass unser Control Interface (USB Interface 0, Alternative 0) vom Typ

Klasse 2, Unterklasse 13 und Protokoll 0 war.
Klasse war Kommunikation, Unterklasse NCM und Protokoll laut Spec *No encapsulated commands / responses*.
Tufftäääh, Tufftäääh, Tufftäääh. Da steht laut und deutlich, na eher kontrastreich und gut leserlich *No encapsulated commands / responses*.
Das Ding kann gar keine Embedded Commands/Responses also wird der Embedded Commands/Responses Treiber (AppleUSBNCMControl) zwar geladen, aber das Gerät unterstützt die Funktion nicht. Kann also nicht gehen. Hätte ich auch gleich lesen können. Obwohl, ich nehme an, dass ich es gelesen hatte, aber es ist halt nicht hängengeblieben.
Es ist ärgerlich, die Zeit für die Analyse des Apple Treibers und das Schreiben eines eigenen Treibers verschwendet zu haben. Aber eigentlich spielt es keine Rolle, denn irgendwann würde ich es ja doch tun müssen.

Wat nu, sprach die Kuh

Na ja wenn man nicht weiter weiß schaut man, wie es die machen, die wissen wie es geht. In unserem Fall sind das die Windowers.
Es gibt Tools mit denen kann man die Kommunikation zwischen Computer und Geräten überwachen. Auf dem Mac gibt es z.B. Wireshark um sich Netzwerkpakete anzuschauen und zu analysieren.

Trüffelschweine

Hier geht es darum ein Tool zu finden, das den USB Verkehr unter Windows überwacht und ausgibt. Solche Tools nennt man oft Sniffer, also USB Sniffer gegoogelt. Und man findet auch ein paar. Sie sind nicht billig, aber meist gibt es eine Demo Version - einen Monat Prüfung bevor man sich ewig bindet. Ein Monat sollte langen.
Verschiedene ausprobiert und alle bringen Windows beim Booten zum Absturz. Na super. "Na, super" mit sarkastischem Unterton sagt man ja nicht mehr. Heute sagt man ja: Na, Diesel.
ie Sniffer arbeiten nach dem selben Prinzip, sie hängen einen Schnüffel-Treiber in den USB Treiberstapel. Während der Schnüffeltreiber sich in den USB Stack wühlt, wie das Trüffelschwein seine Nase in die feuchte Erde, funktioniert USB nicht. Dummerweise ist mein Windows auf einer USB Platte und das Trüffelschwein zieht dem Windows die Platte weg und Windows landet mit dem Schwein Nase an Nase im Dreck.

Neues Fenster

Das Miix hat für Massenspeicher genau einen M.2 Slot. Das macht die Wahl des Ortes für die Windows-Installation leicht, mit auf die M.2. An einem runden Tisch ist immer noch in Platz, alle müssen nur etwas zusammenrücken. Dankbarer-weise geht das auch bei M.2 SSDs, obwohl die nicht rund sind. Das Festplattendienstprogramm geöffnet und 128GB von der HFS+ Partition abgezwackt.

Wie pflegen die Helden zu sagen, "heute ist ein guter Tag, mal wieder die EFI Partition auf einen USB Stick zu kopieren", denn "Kopierst du in der Zeit, bootest du in der Not".
Dann WIndows von eine USB Stick installiert. Die Seriennummer des Miix hat das Windows ungefragt übernommen.
Und Windows geht einwandfrei, Sniffer installiert, geht auch. Nur macos geht nicht mehr. Nicht mal an Clover komme ich ran. Na, Diesel.

Eiskalt

Da läuft's einem kalt den Rücken runter. Aber mit *BOOTICE* bekommt man schnell warme Füße.
Mit *BOOTICE* kann man die EFI Booteinträge manipulieren. Das Programm hat ein etwas altmodisches Aussehen, erfüllt aber seinen Zweck. Man kann alle notwendigen Schritte mit *BOOTICE* erledigen.

Zuerst mountet man die EFI Partition (Reiter Physical Disk, Button Parts Management. Partition anwählen, Button *Assign Drive Letter*).
Dann fügt man einen Eintrag für \EFI\BOOT\BOOTX64.efi hinzu (Reiter *UEFI*, Button *Edit boot entries,* Button *Add*, Datei auswählen).
Dann bewegt man den Eintrag an den Anfang der Liste (Button *Up*),
und speichert die Änderung (Button *Save current boot entry*).

Programm beenden, Neustart, fettig.

Und das nächste mal wird geschnüffelt.