Es tut sich wieder was bei Broadcom WLAN

Beitrag von „schrup21“ vom 25. November 2025, 18:54

~~BcmWLAN DriverKit (Dext) Prototype: Broadcom Wi-Fi Support for macOS Tahoe~~

https://www.insanelymac.com/fo...-support-for-macos-tahoe/

~~Hier auch noch ein paar Infos:~~

~~https://www.insanelymac.com/fo...e/16/#findComment-2844163~~

Edit: wurde gelöscht wie es aussieht.

Beitrag von „DerBeste“ vom 27. November 2025, 14:15

> Zitat von schrup21
>
> ~~BcmWLAN DriverKit (Dext) Prototype: Broadcom Wi-Fi Support for macOS Tahoe~~
>
> https://www.insanelymac.com/fo...-support-for-macos-tahoe/
>
> ~~Hier auch noch ein paar Infos:~~
>
> ~~https://www.insanelymac.com/fo...e/16/#findComment-2844163~~

> **Edit: wurde gelöscht wie es aussieht.**
>
> Alles anzeigen

IIm Moment wird es von mir keine weiteren öffentlichen Aufrufe zur Entwicklung fehlender Treiber geben. Dieses Thema habe ich vorgestern für mich abgeschlossen, nachdem ich das Grundgerüst des DriverKit-Treibers hochgeladen hatte und dafür von „Bin-hacker“ Kommentare erhielt, obwohl ich alles sehr ausführlich erklärt hatte. Ich habe deutlich geschrieben, dass es sich zunächst nur um das Grundgerüst handelt , also wenn es so besser verstehen kann die „Autobahn“, auf der später die eigentlichen „Fahrzeuge“ fahren sollten. Die vollständige Funktion, also das „fahrende Auto“, wollte ich Schritt für Schritt ergänzen und gerne auch im Team entwickeln. Trotzdem wurde das eigentliche Konzept nicht verstanden.

Künftig arbeite ich ausschließlich für mich selbst und nur für meine eigene, nicht kompatible Hardware. Das mache ich bewusst im Hintergrund, um mich nicht erneut angreifbar zu machen.

Die technische Entwicklung läuft dennoch sehr gut. Beim Realtek RTL8111, den ich eigenständig entwickle (hat nichts mit Mieze Realtek8111 zu tun), und beim Broadcom-Treiber gibt es spürbare Fortschritte. Den Realtek RTL802.11ac habe ich inzwischen zu etwa 85 Prozent fertiggestellt. Auch der DriverKit-basierte Text-Treiber lässt sich bereits erfolgreich installieren und laden. Die restlichen rund 15 Prozent sind zurzeit in Arbeit.

Veröffentlicht werden diese Projekte nicht mehr, da es keinen Sinn hat, sich unnötigen Angriffen auszusetzen oder sich die eigene Arbeit schlecht reden zu lassen.

Beitrag von „schrup21“ vom 28. November 2025, 07:31

Hi, ich möchte dir zwar nahe legen, dich nicht von einzelnen abhalten zu lassen, aber natürlich respektieren wir alle deine Entscheidung.

Schade ist es aber schon.

Beitrag von „DerBeste“ vom 28. November 2025, 17:05

> Zitat von schrup21
>
> Hi, ich möchte dir zwar nahe legen, dich nicht von einzelnen abhalten zu lassen, aber natürlich respektieren wir alle deine Entscheidung.
>
> Schade ist es aber schon.

Zur Info:

seit gestern arbeite ich am finalen Compilerprozess und drehe mich seit mehreren Stunden dabei immer wieder um zwei hartnäckige Fehler. Gelegentlich tritt zudem nur ein unerwarteter Compilerfehler auf. Dennoch bin ich überzeugt, dass ich inzwischen etwa 98 % des ersten funktionsfähigen Prototyps des Realtek RTL802.11ac Treibers erreicht habe.

Die Entwicklung ist sehr anspruchsvoll, da Apple im DriverKit Umfeld immer wieder aber auch bewußt viele Hürden hinterläßt, die es uns Hobbybastler schwer machen, stabile und funktionsfähige Treiber nach einem einheitlichen Aufbau zu erstellen. Jede einzelne Komponente muss genau den erwarteten Schnittstellen, Signaturen und Sicherheitsvorgaben entsprechen, sonst verweigert das System den Build oder die Initialisierung.

Trotz dieser komplexen Rahmenbedingungen bin ich zuversichtlich, dass der Prototyp bald vollständig kompiliert und testbereit sein wird. Jede gelöste Hürde liefert wichtige Erkenntnisse, die in die weitere Entwicklung einfliessen und es ermöglichen, zukünftig auch andere Treiberarten effizienter und zuverlässiger zu entwickeln.

Die oben genannten Gedanken mussten einfach einmal ausgesprochen werden, sonst wäre ich innerlich daran explodiert. Mit öffentlichen Äußerungen und Kritik sollte man sich daher so gut wie möglich zurückhalten, denn solche Diskussionen bekommen heute sehr schnell unnötige Dynamik. Ich möchte keine großen Worte schwingen. Erst wenn ich das Projekt mit gutem Gewissen abgeschlossen habe, kann ich eine Version zum Testen bereitstellen. Alles davor ergibt wenig Sinn und macht einen nur unnötig angreifbar, was ich vermeiden möchte. In diesem Sinne.

Beitrag von „DerBeste“ vom 30. November 2025, 02:55

Vielen Dank für die freundlichen Worte.

Nach mehreren Tagen harter Arbeit ist es mir gelungen, den RTL802.11ac.dext-Treiber vollständig zu bauen. Das eine echte Herausforderung! Jetzt habe ich jedoch ein Problem: Der Treiber lässt sich nicht einfach nach /System/Library/DriverExtensions installieren, obwohl alle notwendigen Schritte erfolgreich durchlaufen wurden.

Ich konnte den Treiber bisher nur nach /Library/DriverExtensions übertragen. Das sollte aber eigentlich nicht so sein.

Bisher habe ich auch keinen existierenden Installer für DEXT-Treiber gefunden. Deshalb arbeite ich gerade daran, eine App zu programmieren, die die Installation in /System/Library/DriverExtensions ermöglichen soll. Ob so eine App jemals zuverlässig funktioniert, ist aber ungewiss.

Mit direkten Terminal-Befehlen ist es mir bisher nicht gelungen, den DEXT-Treiber dorthin zu installieren, und mir fällt momentan kein anderer praktikabler Weg ein.

Jeder sinnvolle Ratschlag ist willkommen.

Edit:

Alles claro,

ich habe die Antwort aus dem Netz erhalten und muss die Installation des signierten DEXT ausführen:

1. In das Terminal Fenster öffnen und wechsle in Root-Modus, damit ich alle Systemrechte habe:

bash:

Code

```
1. sudo -s
```

2. Setze die korrekten Rechte für den Zielordner, damit macOS die DEXT akzeptiert:

bash:

Code

```
1. chown -R root:wheel /System/Library/DriverExtensions
2. chmod -R 755 /System/Library/DriverExtensions
```

3. dann kopiere den signierten DEXT-Treiber in das System-Verzeichnis:

bash:

Code

```
1. cp -R /Pfad/zu/meinen/DEXT/*.dext /System/Library/DriverExtensions/
```

4. installiere die DEXTs mit kmutil und erstelle den Kernel-Cache neu:

bash:

Code

1. kmutil install --update-all

5. überprüfe, ob die DEXTs korrekt geladen wurden:

bash:

Code

1. kmutil list
2. kmutil showloaded

6. starte das System neu, damit die DEXTs vom Kernel geladen werden:

bash:

Code

1. reboot

Wichtige Hinweise:

Sicherstellen, dass der KDK zur macOS-Version passt, sonst schlägt `kmutil` fehl.

Dext Driver muss signiert sein, sonst werden sie blockiert.

Arbeite immer als root, damit alle Berechtigungen korrekt gesetzt werden.

Verzichte auf veraltete Befehle wie `bless --bootinfo`, kmutil ersetzt diese Funktion vollständig.

Beitrag von „apfelnico“ vom 30. November 2025, 11:27

Hab zwar keine Ahnung welches Problem gelöst wurde, habe aber große Hochachtung vor der Programmierung einer signierten DEXT. Also modern wie es sein soll. Von so etwas habe ich in unserem Umfeld kein zweites Mal gehört. Meinen Glückwunsch!

Beitrag von „Noir0SX“ vom 30. November 2025, 12:03

Funktionieren Deine Zeilen in macOS 11+ überhaupt noch ? **systemextensionsctl**

Beitrag von „DerBeste“ vom 30. November 2025, 19:57

> Zitat von apfelnico
>
> Hab zwar keine Ahnung welches Problem gelöst wurde, habe aber große Hochachtung vor der Programmierung einer signierten DEXT. Also modern wie es sein soll. Von so etwas habe ich in unserem Umfeld kein zweites Mal gehört. Meinen Glückwunsch!

Hi Apfelnico,

Die folgenden Ausführungen basieren auf umfangreichen Eigenentwicklungen und praktischen Erfahrungen aus mehrwöchiger Arbeit mit macOS-Treibern. Angesichts der spärlichen und oft unzureichenden offiziellen Veröffentlichungen seitens Apple musste ich viele der hier dargestellten Erkenntnisse durch ausgiebiges Experimentieren und intensives Testen gewinnen.

Apple liefert nach meinem Kenntnisstand nur minimale Dokumentation zu den neuen DEXT-Technologien und verschweigt bewusst wichtige Details, was die Entwicklung erheblich erschwert.

Hier eine Gesamtübersicht: KEXT vs. DEXT und die Zukunft von Hackintosh

Die Entwicklung von Treibern unter macOS befindet sich in einem tiefgreifenden Wandel. Apple verfolgt die klare Strategie, die traditionellen Kernel Extensions (KEXT) vollständig abzulösen

und durch moderne Driver Extensions (DEXT) zu ersetzen. Dieser Wandel hat erhebliche Auswirkungen auf alle Entwickler, insbesondere auf Community- und Hobby-Programmierer.

1. Der technische Wandel: KEXT zu DEXT

1.1 Kernel Extensions (KEXT) - Die alte Architektur

KEXTs sind Erweiterungen, die direkt im Kernel von macOS ausgeführt werden. Diese Architektur bietet zwar volle Kontrolle und schnellen Hardwarezugriff, bringt aber erhebliche Nachteile mit sich:

- Direkter Zugriff auf Kernel-Ressourcen
- Hohe Anfälligkeit für Systemabstürze (Kernel Panics)
- Manuelle Nutzerfreigabe seit macOS High Sierra
- Erhöhte Systemanfälligkeit durch fehlerhafte Module

KEXTs gelten heute offiziell als veraltet und werden von Apple langfristig nicht mehr unterstützt.

1.2 Driver Extensions (DEXT) - Die moderne Lösung

Driver Extensions sind Teil des System Extension Frameworks und laufen vollständig im User Space. Sie sind sicherer, stabiler und moderner:

- Keine Kernel-Ausführung → deutlich weniger Systemabstürze
- Abgesicherte, sandbox-basierte Umgebung
- Müssen in ein App-Bundle integriert werden
- Nutzung moderner, restriktiver APIs
- Abhängigkeit vom System Extension Management

DEXTs sind zwar sicherer, aber erheblich komplizierter zu entwickeln und bieten weniger Zugriffsrechte als klassische KEXTs.

2. Apples strategischer Kurs seit 2017

Eigentlich ist seit macOS High Sierra klar erkennbar, dass Apple die KEXTs schrittweise aus dem System entfernt. Die wichtigsten Meilensteine:

- 2017 – High Sierra: Einführung von User-Approved Kernel Extension Loading
- 2019 – Catalina: Start des System Extension Frameworks

- 2020 – Big Sur: Signed System Volume und Klassifizierung von KEXTs als "Legacy"
- 2021–2024: Weitere Einschränkungen und Abschaffung zahlreicher Kernel-APIs

Diese Entwicklung zeigt: Apple will KEXTs vollständig eliminieren.

3. Auswirkungen auf Entwickler und Community-Projekte

Die Umstellung auf DEXTs bringt erhebliche Einschränkungen mit sich:

- Keine direkten Kernelhooks mehr
- Stark limitierte Zugriffsmöglichkeiten auf Hardware
- Treiber müssen in Apps verpackt werden
- Nur eingeschränkt dokumentierte APIs
- Höherer Aufwand beim Debugging
- Zunehmende Abhängigkeit von signierten Entitlements

Damit werden viele etablierte Vorgehensweisen der Hackintosh-Community deutlich erschwert.

4. Das nahende Ende der Hackintosh-Ära

Mit der erwarteten finalen macOS-Generation macOS 26.x, die voraussichtlich die letzte Version mit x86_64-Unterstützung sein wird, steht der Hackintosh-Community ein tiefgreifender Einschnitt bevor.

Die zentralen Punkte:

- Apple stellt seine gesamte Produktlinie auf Apple Silicon um
- Ab macOS 26.x ist sehr wahrscheinlich Schluss mit neuen x86_64-Builds
- Ohne x86_64-Kernel existiert keine Basis mehr für Hackintosh-Installationen
- Aktuelle und zukünftige Treiberentwicklungen (DEXT) sind zunehmend auf Apple Silicon ausgelegt
- Der klassische PC-basierte Hackintosh wird mittelfristig nicht mehr realisierbar sein

Damit ist klar: Wir bewegen uns auf das tatsächliche Ende des Hackintosh-Ökosystems zu.

5. Spezielle Problematik: PCI Netzwerkkarten

Die bisherigen Netzwerk- und PCI-Treiber wie die beliebten Realtek RTL8xxx Karten werden nicht mehr von den modernen DEXT-DriverKit unterstützt. Man kann zwar versuchen, einen Kext dafür zu bauen, aber mit vielen Hürden.

Alternative zu IONetworkController / IOPCIDevice für Realtek-Ethernetkarten

In modernen macOS-Versionen ist die klassische KEXT-Architektur weitgehend abgelöst worden. Besonders Netzwerk- und PCI-Treiber betroffen:

Klassisch (KEXT)	DriverKit (Modern)
IONetworkController	NetworkDriverKit
IOPCIDevice	IOPCI (DriverKit PCI Family)
Kernel-Level Netzwerktreiber	Userspace-basierter Netzwerktreiber
Voller Hardwarezugriff	Starke Sicherheits-Einschränkungen

Die moderne API IOPCI erlaubt grundlegende Geräte-Erkennung, Lesen/Schreiben in PCI-Konfiguration und Mapping von BAR-Memory, aber nicht denselben Tiefgang wie IOPCIDevice im Kernel.

Für Netzwerkkarten hat Apple das Framework NetworkDriverKit eingeführt, das IONetworkController ersetzt. Damit erhält man eine moderne Userspace-taugliche API, Unterstützung für Netzwerk-Interfaces und Queues für RX/TX.

Fazit für Realtek-Chips:

Ein vollständiger RTL8168/8111-Treiber wie früher lässt sich in DriverKit nicht mehr 1:1 umsetzen. Eine realistische Lösung besteht aus PCI-Handling in einer minimalen Kernel-Extension und Ethernet-Interface in einer DriverKit SystemExtension.

6. WLAN-Adapter unter macOS Tahoe

6.1 USB-WLAN-Adapter - höchste Erfolgschance

USB-Adapter sind aktuell die einzige realistische und dauerhaft tragfähige Basis für WLAN-Treiber unter macOS Tahoe, da sie über das User-Space-fähige IOUSBHost-Framework laufen.

Gute Chancen bestehen für:

- Realtek RTL8812AU / BU (802.11ac)
- Realtek RTL8822BU (802.11ac)
- Mediatek MT7612U (802.11ac)
- Mediatek MT7921U (WiFi 6)

6.2 PCIe-WLAN-Karten - möglich, aber extrem schwierig

PCIe-Treiber über DEXT sind theoretisch machbar, aber extrem schwierig, weil:

- DMA, Interrupts und Registerzugriffe stark eingeschränkt sind
- Apple den gesamten WLAN-Stack vollständig blockiert
- AWDL, Handoff etc. nicht nachbildbar sind
- DEXT kein direkter Ersatz für die alten Broadcom-KEXTs ist

Chancen nach Chipsätzen:

- BCM4360: niedrig-mittel (noch am ehesten machbar)
- BCM43602: niedrig (komplexere Firmware, wenig Dokumentation)
- BCM4352: niedrig (bekannte Limitierungen)

6.3 Apple-eigene WLAN-Module - praktisch ausgeschlossen

Alle Chipsätze, die Apple in Macs verbaut, sind:

- tief in das AppleIO80211-Framework integriert
- stark signiert
- vollständig proprietär
- ohne DEXT-Äquivalent
- ohne veröffentlichte APIs

Dazu gehören BCM4350, BCM4364, BCM4377, BCM4387 und alle Apple Silicon internen WiFi/BT-Module.

7. Fazit

- KEXTs waren lange das Fundament vieler Community-Treiber, verlieren aber endgültig ihre Bedeutung

- DEXTs sind die Zukunft, allerdings restriktiver, komplexer und weniger flexibel
- Apples klare Sicherheitsstrategie erschwert freie Treiberentwicklung erheblich
- Mit macOS 26.x nähert sich die Hackintosh-Ära dem Abschluss, da keine neuen x86-Builds mehr zu erwarten sind
- Für Netzwerkkarten und WLAN-Adapter bedeutet dies: nur noch sehr eingeschränkte Möglichkeiten, vor allem über USB-basierte Lösungen

Die Entwicklergemeinschaft steht vor der Herausforderung, sich an diese neuen Gegebenheiten anzupassen oder alternative Lösungen zu finden.

Problemstellung mit dem RTL802.11ac.dext Treiber unter macOS Tahoe:

Leider verliefen die Testergebnisse mit meinem RTL802.11ac.dext unter macOS Tahoe nicht erfolgreich. Während der DEXT-Treiber unter macOS Sequoia problemlos mittels des Installations-Tools installiert wird und korrekt erkannt wird, sieht die Situation unter macOS Tahoe derzeit sehr kritisch aus.

Zwar lässt sich die Installation mit dem Tool ohne Fehlermeldungen durchführen, jedoch kommt es anschließend zu keiner funktionsfähigen Initialisierung des Treibers.

Trotz ordnungsgemäßer Vorgehensweise:

Korrekter Aufbau und Einbindung des DEXT-Pakets

Vollständige Code-Signierung nach Apple-Richtlinien

Fehlerfreie Installation durch das zertifizierte Tool

Ich muss die Ursache für dieses Verhalten unbedingt identifiziert und beheben.

Edit:

So ein Ärger. Ich habe oben ausführlich erklärt, dass man für den Build das passende KDK, also das Kernel Development Kit, installieren muss. Und genau das habe ich selbst völlig übersehen. Mir ist erst jetzt wieder eingefallen, dass ich es noch nicht eingerichtet habe. Das

muss ich unbedingt nachholen. 

Beitrag von „cobanramo“ vom 30. November 2025, 23:34

Gut und schön, jetzt hast du das ganze problematik was der rest der Welt schon seit 4 Jahren weiss beispielhaft redaktionell aufgeschrieben.

Nur um die frage vom Nico wieder aufzugreifen, “Welches teilproblem konntest du da umgehen?”

Hast du ne Ansatz wie man „AWDL“ zum laufen bewegen kann? Den darum gehts ja im grossen und ganzen damit wir die alten vorhandenen Broadcom Chips zum leben erwecken, den Wlan selber können wir ja schon seit etlichen Jahren moderner, schneller In form von Intel sei es per Usb oder Pci haben.

Worin genau wurdest du eigentlich im grossen und ganzen bei den „Verrückten“ nicht verstanden oder missverstanden?

Gruss Coban

Beitrag von „Mieze“ vom 1. Dezember 2025, 00:50

DerBeste Ok, Du hast in Deinem Post Apple's Marketing im Bezug auf Treiber gut zusammengefasst, aber dabei übersehen, dass es sich eben nur um Marketing-Geschwätz handelt. Modern ist der ganze DEXT-Mist nämlich nicht. Treiber in den Userspace zu verlagern wurde auch mal unter Linux versucht, hat sich aber nicht durchgesetzt, weil die Nachteile überwiegen:

- Hoher Overhead durch ständige Prozesswechsel und das Mappen, bzw. Kopieren von Puffern.
- Hohe Latenz von IO-Operationen
- Ungeeignet für Highspeed-IO, weil die oben genannten Faktoren einen Flaschenhals bilden.
- Hohe Systemlast

Das Argument der Sicherheit und Stabilität mag auch nur bei schlecht programmierten Treibern gelten. Im Prinzip ist eine sauber programmierte Kext nicht unzuverlässiger als ein DEXT-Treiber. Wenn man das Marketing-Geschwätz mal beiseite schiebt, dann erkennt man, worum es Apple wirklich geht. Sie möchten das System komplett abschotten und der Kontrolle des Benutzers entziehen, so dass Apple und nicht mehr Du entscheidet, was auf Deinem System läuft. Komischerweise laufen Apple's Treiber, Skywalk eingeschlossen, weiterhin im Kernel. Insgesamt ist die ganze Netzwerkarchitektur von macOS auch im Vergleich mit Linux oder Windows eher primitiv und wenig auf Performance ausgerichtet.

Unter dem Strich ist DriverKit mit seinen DEXT-Treibern Mist, weil es mit zu vielen Einschränkungen und wegen dem hohen Abstraktionsgrad mit einem zu hohen Overhead verbunden ist. Für Hackintoshing sind wir sowieso auch klassische Kernel Extensions angewiesen, weil sich essentielle Dinge, wie z. B. Lilu, etc. nur als Kext realisieren lassen. Es lohnt sich daher nicht, sich zwecks Hackintoshing mit DriverKit auseinanderzusetzen. Abgesehen davon benötigen DEXT-Treiber zwingend AppleVTD, so dass AMD-Boards, und Intel-Boards, die mit AppleVTD inkompatibel sind, ausgeschlossen sind.

> Zitat von DerBeste
>
> Fazit für Realtek-Chips:
>
> Ein vollständiger RTL8168/8111-Treiber wie früher lässt sich in DriverKit nicht mehr 1:1 umsetzen. Eine realistische Lösung besteht aus PCI-Handling in einer minimalen Kernel-Extension und Ethernet-Interface in einer DriverKit SystemExtension.

Von Jumbo-Frames abgesehen, die von DEXT-Treibern grundsätzlich nicht unterstützt werden, stimmt das überhaupt nicht. Natürlich kannst Du mit DriverKit einen NetworkDriverKit-Treiber für PCIe-Netzwerkkarten schreiben, abgesehen von oben genannter Einschränkung. Apple liefert ja solche Treiber für bestimmte Karten von Intel und Mellanox bereits mit.

Für USB-Ethernet-Adapter brauchen wir uns auch garnicht die Mühe machen eigene Treiber zu entwickeln, weil die entweder mit den generischen Treibern von Apple laufen, oder aber Realtek entsprechende Treiber bereitstellt.

Ich habe mich vor einem Jahr mal mit NetworkDriverKit beschäftigt und bin zu dem Ergebnis gekommen, dass dieses Framework, trotz einiger guter Ansätze, eher ein Schritt zurück als nach vorn ist und es sich aus Hackintosh-Sicht nicht lohnt, sich damit zu beschäftigen.

> Zitat von DerBeste
>
> Problemstellung mit dem RTL802.11ac.dext Treiber unter macOS Tahoe:

> Leider verliefen die Testergebnisse mit meinem RTL802.11ac.dext unter macOS Tahoe nicht erfolgreich. Während der DEXT-Treiber unter macOS Sequoia problemlos mittels des Installations-Tools installiert wird und korrekt erkannt wird, sieht die Situation unter macOS Tahoe derzeit sehr kritisch aus.

Hier wird man abwarten müssen, ob und wann Realtek ein Update veröffentlicht. Mangels Dokumentation ist es leider unmöglich native WLAN-Treiber für macOS zu schreiben. Es bleibt also nur der Umweg, dass der WLAN-Treiber ein Ethernet-Interface emuliert. Die WLAN-spezifische Konfiguration muss dann über die Hintertüren durch eigene Utilities oder Konfigurationsdateien vorgenommen werden. Leider ist dieser Weg aufwendig un mit viel Arbeit verbunden.

Beitrag von „mhaeuser“ vom 1. Dezember 2025, 01:42

Mieze War die Erklärung nicht einfach ChatGPT?

Wie auch immer, Treiber im Userland sind eine weitgehend objektiv gute Sache. Was du wetterst, stimmt natürlich alles, ist aber auch eine Niche. Die meisten Treiber haben keine derartigen Performanceanforderungen. Je nach Geräteklasse und Design könnten Kontextswitches sogar verringert werden. Für die Netzwerkdomäne ist das natürlich ein eher ungünstiges Model. Ich frage mich aber generell, wie Benchmarks unter Apple Silicon aussähen.

Gut programmierte Treiber brauchen das nicht… jo und gut designte Hardware mit gut programmierter Software brauchen auch kein Segmenting, RELRO oder Stack Canaries. Korrektheit bei realistischer Software, gab’s nicht, gibts nicht und wird es erstmal nicht geben, fertig. Hände hoch, wer unter Windows noch keinen Bluescreen wegen irgendeinem OEM-Treiber hatte. Oder Privilege Escalation…

Beitrag von „Mieze“ vom 1. Dezember 2025, 13:16

mhaeuser Mag sein, das der Text von ChatGPT stammt, aber soweit wollte ich nicht gehen, ihm das zu unterstellen.

Niemand hat grundsätzlich etwas gegen Treiber im Userland. Für HID-Treiber ist das kein Problem und auch den ganzen Sensor-Kram könnte man dorthin verlagern, aber letzterer läuft weiterhin im Kernel.

Treiber für HighSpeed-IO mögen zwar zahlenmäßig eine kleine Gruppe darstellen, leider zählen sie aber zu den wichtigsten im System: GPU, Massenspeicher und Netzwerk. 3rd-Party-GPUs hat Apple auf AppleSilicon gleich ganz gekillt und Netzwerktreiber weitgehend in den Userspace verlagert. Tolle Entscheidung! Ironischerweise sind es z. T. die Treiber von Apple, welche schlecht programmiert sind und KPs verursachen. So kommt Tahoe mit einer Kext namens AppleEthernetRL.kext, einem Skywalk-Treiber für Realtek RTL8125-Chips. Natürlich werden nur spezielle Chips mit Apple-Firmware unterstützt, aber anstatt bei der Initialisierung zu prüfen, ob der vorhandene Chip kompatibel ist und andernfalls die Initialisierung abzubrechen, verursacht AppleEthernetRL.kext eine KP beim Booten wenn du einen RTL8125 in deinem Hackintosh hast. Ist das Absicht, oder einfach nur Schlampigkeit?

Vielleicht sollte man macOS aus Sicherheitsgründen vollständig ins Userland auslagern und etwas stabileres als Unterbau verwenden, oder lieber gleich komplett in eine VM verlegen?

Erfahrungsgemäß lassen sich die meisten Softwarefehler auf zwei Ursachen zurückführen:

- Zu viel gewollt.
- Mit heißer Nadel gestrickt.

Vielleicht ist das auch der Grund, warum Linux-Treiber in der Regel zuverlässiger arbeiten, als Treiber für macOS und Windows. Aber bei Linux dauert es manchmal auch etwas länger, bis neue Hardware unterstützt wird.

Übrigens was Windows betrifft, so brauchst Du keine OEM-Treiber um einen Bluescreen zu bekommen. Windows zerlegt sich gerne auch selbst, ohne dass da OEM-Treiber ins Spiel kommen müssen.

Früher habe ich durchaus mal mit dem Gedanken gespielt, irgendwann mal eine Mac mit AppleSilicon zu kaufen. Nachdem ich mich 2 Wochen lang durch den Source Code des Kernels von Tahoe gekämpft habe, weiß ich, dass der Hackintosh mein letzter Rechner mit macOS ist. ☹️

Beitrag von „mhaeuser“ vom 1. Dezember 2025, 20:00

Mieze Vom Linux-Kernel höre ich viel Gutes, habe aber noch nie wirklich damit gearbeitet. In jedem anderen “größeren” Projekt, mit dem ich zu tun hatte (EDK II, GRUB, systemd-boot, iPXE EFI, GNU-EFI, Shim, git, rEFInd, Clover, OpenCore/AUDK(!!), rustc, …) steht Schmerz an der Tagesordnung - gelegentlich auch von mir verursacht. Wenn du das alles unter “heiße Nadel” (ist auch regelmäßig der Fall, so ist es nicht) kategorisierst, kann man machen, ist aber dann nur bedingt aussagekräftig. Vielleicht ist Linux auch der eine heilige Gral, mit dem ich eben noch nicht konfrontiert war. Aber, wie in den meisten anderen Ingenieursfeldern ist offensichtlich auch in der Informatik eine Mischung aus Vorsichtsmaßnahmen und etwas Demut angebracht.

(Ein benutzbares Linux-Userland ist mir auch noch nicht untergekommen, Kernel hin oder her…)

Beitrag von „DerBeste“ vom 9. Dezember 2025, 18:21

Ich hab's erst nach Wochen festgestellt, das meine ganze Tipperei für die Katz war. Nicht, weil ich doof bin, sondern weil Apple sein System wie eine Festung abgeriegelt hat, gegen die man einfach keine Chance hat.

Ich dachte erst, es gibt eine Möglichkeit das zu umgehen, aber Pustekuchen. Ich hätte besser auf die anderen gehört, die meinten: Vergiss DriverKit, besonders für WLAN, das ist gelaufen.

Tja, selbst auf die Nase fallen ist die beste Lehre. Apple hat das System so gebaut, dass wir alle verlieren.

DriverKit ist keine Lösung, sondern eine Fessel.

Apple will die Treiber nur noch selbst schreiben. Alles, was nicht ins Konzept passt, fliegt raus.

* USB-WLAN? Tot.

* Eigene Hardware? Egal.

* Broadcom? Pustekuchen.

* Monate Arbeit? Apple lacht.

Fakt ist: Ich durfte gar nicht gewinnen. Und das gilt auch für viele, die viel mehr draufhaben als ich.

Das Schlimmste: Apple redet von Sicherheit und Stabilität. Schön und gut, aber das ist doch alles Quatsch!. Ja, Mieze, ich gebe dir nun 100 % Recht, alles Bullshit!

Es geht um Kontrolle, um mehr Kohle und darum, andere klein zu halten. Wer nicht Apple ist, ist nicht willkommen.

Ich war so blöd zu glauben, Apple weiß, was es tut, und will uns nicht verarschen.

Falsch gedacht.

Das Ganze ist Absicht.

Und das betrifft nicht nur mich, sondern alle:

* Hackintosh-User

* Hardware-Bastler

* Open-Source-Leute

* Entwickler, die einfach nur schaffen wollen

* und alle macOS-Nutzer, die für billige Hardware tief in die Tasche greifen müssen, weil Apple abkassieren will

Apple hat vergessen, warum so viele User macOS mochten: Weil man früher noch selbst was machen konnte und Treiber anpassen. Diese Zeiten sind vorbei.

Die harte Realität ist: Ich bin nicht gescheitert. Der Treiber ist komplett fertig, aber Apple hat den Riegel vorgeschoben.

Wenn du alles DriverKit-tauglich machst, landest du am Ende bei bestimmten Funktionen und Headern, die es in normalen macOS- oder Kernel-Treibern (kext) gab – wie zum Beispiel mach/mach_types.h oder spezielle USBHost-Header. Die sind aber in DriverKit nicht mehr gültig. Das Bauen klappt dann einfach nicht mehr.

DriverKit-Treiber dürfen nur die DriverKit- und USBDriverKit-Header nehmen, die Apple erlaubt. Aber Xcode zwingt dich indirekt dazu, trotzdem verbotene Includes zu benutzen, weil Xcode halt manchmal komische Sachen macht, auf die man keinen Einfluss hat. Ein sauberer Build über ein Makefile ist darum unmöglich: Der Compiler checkt die verbotenen Dateien nicht und gibt Fehlermeldung aus, weil DriverKit zum Beispiel keinen Zugriff auf mach/mach_types.h hat.

Kurz gesagt: Apple führt dich absichtlich in eine ausweglose Situation, in der bestimmte Sachen einfach nicht gehen und der Build-Prozess unmöglich ist.

Und das fühlt sich nicht wie ein technisches Problem an, sondern wie ein fetter Betrug.

Wenn Apple meint, sie können jedes Jahr ein neues System raushauen und alle rennen hinterher, dann irren sie sich.

Viele werden gehen, womöglich zu Linux, und zu anderen offenen Systemen, wo man nicht erst fragen muss.

Ich hab's kapiert, aber es ist ein mieses Gefühl.

Nicht, weil ich verloren habe, sondern weil das Spiel von Anfang an beschissen war.

Und ja, der Text wurde teilweise mit einer KI verfasst!!

Beitrag von „MacGrummel“ vom 9. Dezember 2025, 22:23

Kleiner Scherz: "Doch sicher mit der Apple-Ki?"

Dass das macOS-System immer verschlossener wird und da der Unterschied zum iOS nur noch marginal ist, ist leider nicht grad eine neue Erkenntnis.

Deshalb müssen wir für den Hackintosh ja Treiber für nicht von Apple vorgesehene Hardware schon seit Jahren über OpenCore (oder Clover) in den Boot-Prozess einschleusen. Und jetzt schon mit dem dritten Jahres-Betriebssystem alte Apple-Treiber, die Apple selbst rausgeworfen hat, sogar mit OCLP direkt ins System schleusen/patchen.

Bei Apple war das eigentlich noch nie anders: zwei Jahre nach Einstellung der Produktion der Geräte fliegen auch die dazu gehörenden Treiber aus dem System. Für die meisten normalen Apple-User ist das ja auch nur ein sehr geringes Problem, ein Problem ist es da höchstens, dass es bei Apple faktisch unmöglich ist, wieder auf ein älteres Betriebssystem zurück zu gehen.

Das geschlossene System hat halt bekannte Vor- und Nachteile: ich brauche am Mac nicht nach dem Update oder erst recht einer Neu-Installation noch stundenlang die für meine Hardware passenden Treiber zusammen sammeln, sie sind eben schon dabei. Dass es faktisch keine offenen Schnittstellen für Hardware-Treiber gibt, ist zwar schade, aber eben auch konsequent.

Btw: was kann eigentlich mit den PCI-Schnittstellen im aktuellen MacPro betrieben werden??

Schon im MacPro 6,1 von 2013 war ja in der Original-Software das Betreiben von externen

Grafikkarten deaktiviert, obwohl es mit OC-Hilfe und passender Kext ohne jedes Problem möglich ist. Und jetzt sind bei den M-Macs die externen Thunderbolt-Grafik-Kisten ja offiziell wieder raus. Deshalb dürfte (Miterfinder) Apple den Anschluss eigentlich nicht Thunderbolt nennen..

Wofür der Driver-Kit eigentlich gedacht ist, wenn neue Treiber garnicht geladen werden können, erschließt sich mir allerdings auch nicht.

Beitrag von „Sascha_77“ vom 9. Dezember 2025, 23:00

> Zitat von MacGrummel
>
> Dass das macOS-System immer verschlossener wird und da der Unterschied zum iOS nur noch marginal ist, ist leider nicht grad eine neue Erkenntnis.

Man kann nur hoffen, dass man (im Gegensatz zu iOS) immer mit csrutil und authenticated-root machen kann was man möchte. Sollte das irgendwann mal wie bei iOS sein wo man einen echten Jailbreak braucht kriege ich einen Brechreiz. Das wäre dann sogar ein Grund für mich vom "M" abzuwenden was sehr bedauerlich wäre. Ich hab damals ein iPhone 3GS gehabt. Das war auch das Einzige iPhone in meiner Handylaufbahn weil ich von dem Jailbreak Gedönse und das damit verbundene HickHack nach jedem iOS Update die Nase voll hatte.

Beitrag von „edifant“ vom 10. Dezember 2025, 20:49

Sascha_77 hab mich mittlerweile auch von IOS verabschiedet bis aufs Business - Mobile und hab für Onlinebanking & Co ein billiges Samsung Gedoehns mit Prepaid und für denperönlichen Bedarf ein Pixel mit Graphene. Werde vom Kauf eines Mac Studio absehen, meinenHacky und mein Macbook Pro M2 mit Sequoia solange wie möglich betreiben und in ca. 2Jahren mich wohl von Micro§oft, Apple & Co völlig verabschieden. Kurzfristig werde ich michvon Facebook, What's App & Co verabschieden und all meine eMail - accounts "himmeln" undeinen für Freunde, Familie & Co haben und einen für unwichtige Dinge. Mein Freund Rupert ausWien hat darüber resümiert und festgestellt - die Lage ist hoffnungslos aber nicht ernst 

Beitrag von „mhaeuser“ vom 10. Dezember 2025, 23:33

MacGrummel Selbstverständlich können neue Treiber geladen werden. WiFi ist nur keine unterstützte Geräteklasse - jeder Mac hat bereits WiFi, also wird in die Richtung keine Arbeit gesteckt. Inwieweit es “unmöglich” ist im Vergleich zu den alten Ethernet-Emulatoren, weiß ich nicht - wenn es an mach_types.h scheitert, muss man wohl auf das nächste tolle LLM warten.

Beitrag von „DerBeste“ vom 11. Dezember 2025, 03:03

> Zitat von mhaeuser
>
> MacGrummel Selbstverständlich können neue Treiber geladen werden. WiFi ist nur keine unterstützte Geräteklasse - jeder Mac hat bereits WiFi, also wird in die Richtung keine Arbeit gesteckt. Inwieweit es “unmöglich” ist im Vergleich zu den alten Ethernet-Emulatoren, weiß ich nicht - wenn es an mach_types.h scheitert, muss man wohl auf das nächste tolle LLM warten.

Okay, lass mich eines klarstellen, weil die Diskussion sich im Moment etwas im Kreis dreht: Das Problem ist nicht einfach nur an „mach/mach_types.h“ das ist lediglich ein Symptom, nicht die Ursache.

Als Entwickler solltest du das leicht reproduzieren können. Es geht nicht um eine einzelne

Datei, sondern darum, wie DriverKit aufgebaut ist.

Im Grunde ist es so: DriverKit-Treiber dürfen nur die Header aus DriverKit und USBDriverKit verwenden, die dafür gedacht sind. Aber sobald du anfängst, echte Treiberlogik zu schreiben, wie zum Beispiel für USB, PCI oder Netzwerk nahe Funktionalität dann stößt du zwangsläufig auf Abhängigkeiten, die früher zulässige Kernel-APIs verwendet haben aber heute vollständig isoliert sind.

Das beinhaltet unter anderem:

* mach_*-Typen etc.

* IOUSB/IOService-Strukturen, die in DriverKit nicht übernommen wurden

* Synchronisations- und Memory-Primitive, für die es keinen Ersatz gibt

* Includes, die Xcode selbst über Standard-C++ Includes einzieht, auf die man selbst keinen Einfluss hat

Selbst wenn du jede direkte Referenz sauber entfernst, kommst du nicht weiter: Du ersetzt etwas Verbotenes, brauchst dafür etwas anderes, das auch nicht erlaubt ist, und dann geht der Build woanders schief.

Das ist kein Anfängerfehler und kein „falscher Include“.

Das ist ein unauflösbarer Zyklus. Du kannst es dir so vorstellen: Eine Katze, die ihren eigenen Schwanz hinterher jagt und egal, wie schnell sie hinterher läuft, wird sie niemals zum Ziel gelangen, weil das System bewusst so konstruiert wurde.

Ein weiterer Punkt, der gern übersehen wird:

Ein sauberer Build außerhalb von Xcode, etwa über ein Makefile oder CMake, ist faktisch unmöglich, da der Compiler während der Prüfung der erlaubten Includes scheitert, bevor man überhaupt funktionalen Code erreicht. Das ist kein Zufall, sondern Toolchain-Design.

Zu deiner Aussage „WiFi ist keine unterstützte Geräteklasse“: Genau das ist der Punkt.

DriverKit suggeriert Offenheit, bietet sie aber real nur dort, wo Apple ohnehin vollständige Kontrolle besitzt. Dass alle Macs bereits WLAN haben, ist kein technisches Argument, sondern ein strategisches und erklärt exakt, warum diese Klassen bewusst ausgespart wurden.

Kurz gesagt: Ja, man kann formal neue DriverKit-Treiber laden. Aber in der Praxis sind sie so eingeschränkt, dass echte Hardware-Unterstützung nicht möglich ist.

Aber wenn du das nicht glauben magst, dann kannst du es gern selbst probieren:

* Einen eigenen USB- oder Netzwerk-Treiber schreiben

* Ohne private Frameworks

* Ohne alte Kext-Header

* Nur mit DriverKit

Das Ergebnis ist reproduzierbar und unabhängig vom Skill-Level. Das ist kein Versagen einzelner Entwickler.

Das System ist von Anfang an so gebaut, dass man nicht gewinnen kann.

 Ach, sorry, mit Hilfe der KI verfasst

---

Beitrag von „DerBeste“ vom 14. Dezember 2025, 00:59

Zum Projekt BCMWLANCompanion werde ich den Beitrag vorerst zurückhalten und zu einem späteren Zeitpunkt veröffentlichen. Eine entsprechende Mitteilung folgt.

Beitrag von „Mieze“ vom 14. Dezember 2025, 14:29

> Zitat von Sascha_77
>
> Man kann nur hoffen, dass man (im Gegensatz zu iOS) immer mit csrutil und authenticated-root machen kann was man möchte. Sollte das irgendwann mal wie bei iOS sein wo man einen echten Jailbreak braucht kriege ich einen Brechreiz.

Ich fürchte der Tag, an dem man das nur noch mit einem bezahlten Developer Account machen kann, dürfte nicht mehr weit sein. macOS bewegt sich immer mehr in Richtung iOS und dort war es schon immer so und dann auch nur für den User Space. Der Kernel war da ohne Jailbreak schon immer tabu.

Beitrag von „DerBeste“ vom 23. Dezember 2025, 08:09

Hallo zusammen,

versprochen ist versprochen, und dieses Versprechen halte ich nun ein.

Hier ein kurzes und einfaches Update zu meinem Projekt. Den neuen DriverKit-Treiber für diverse Broadcom-WLAN-Chips konnte ich erfolgreich bauen, linken und signieren.

Der Treiber entspricht vollständig DriverKit Version 24.5 und läuft nur im User-Space (das war

ein harter Kampf). Kernel Extensions werden daher nicht mehr benötigt. Das Build-System kümmert sich automatisch um die Einbindung der passenden Firmware. Unterstützt werden aktuell 23 verschiedene Broadcom-Chipsätze, zum Beispiel BCM4350, BCM4360 und BCM43602 etc. Die grundlegenden Hardware-Funktionen sind umgesetzt. Dazu gehören das MMIO-Mapping, die Verwaltung von DMA-Puffern und das Zurücksetzen der Hardware-Kerne etc. Der komplette Build-Prozess läuft automatisch ab und beinhaltet auch das Code-Signing sowie die anschließende Prüfung.

Der Treiber ist damit bereit für die erste Installation und für erste Tests auf meiner Hardware. Nun muss die passende Hardware her.

Ein paar Bilder befinden sich im Anhang

Beitrag von „karacho“ vom 23. Dezember 2025, 09:30

> Zitat von DerBeste
>
> Unterstützt werden aktuell 23 verschiedene Broadcom-Chipsätze, zum Beispiel BCM4350, BCM4360 und BCM43602 etc.

Super

Jedoch sehe ich keine Firmware für die BCM4360.

Beitrag von „DerBeste“ vom 23. Dezember 2025, 09:39

> Zitat von karacho
>
> Super
>
> Jedoch sehe ich keine Firmware für die BCM4360.

Ich war beim Verfassen meines Textes etwas voreilig, aber das ist zunächst nur ein kleines Problem. Solange ich das entsprechende Data Sheet und die Firmware im Netz finden kann, lässt sich die Implementierung problemlos durchführen, da das Grundgerüst bereits steht.

Vielen Dank für den Hinweis.

Beitrag von „Azteca“ vom 23. Dezember 2025, 09:46

Und wie sieht es aus mit **BCM94360CS2?**

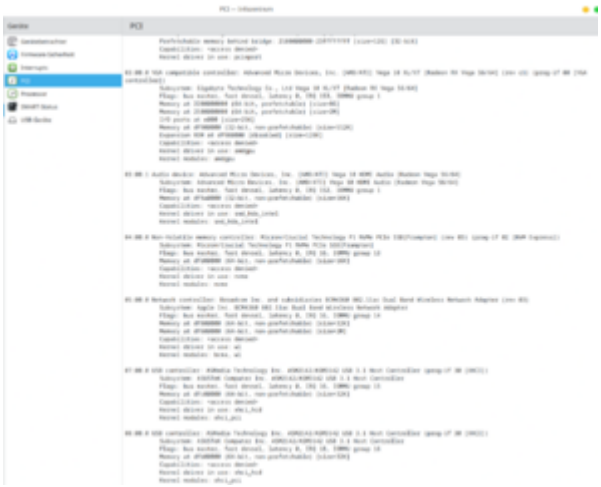
Beitrag von „DerBeste“ vom 23. Dezember 2025, 10:06

> Zitat von Azteca
>
> Und wie sieht es aus mit **BCM94360CS2?**

Ich benötige die Data Sheet und die passende Firmware, dann wäre eine Umsetzung unter Umständen möglich.

Beitrag von „karacho“ vom 23. Dezember 2025, 10:47

DerBeste Die Firmware habe ich noch nirgends gefunden. Ich habe so eine PCIe Karte im Rechner. Ist es nicht möglich die Firmware auszulesen und als .bin zu speichern?



Beitrag von „Azteca“ vom 23. Dezember 2025, 10:56

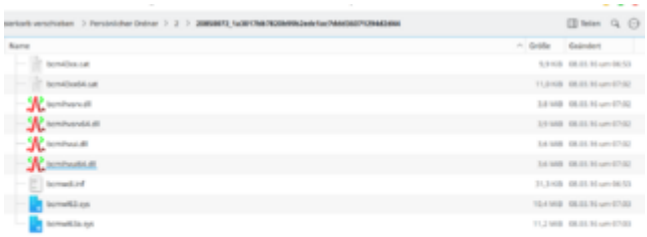
Würde gerne die Sachen liefern, jedoch hab keine Ahnung wie man das aus der Karte kopiert.

Beitrag von „karacho“ vom 23. Dezember 2025, 11:03

Die KI sagt mir:

> Zitat
>
> Für die Auslesung der Firmware eines solchen Chips ist ein spezielles Hardware-Tool erforderlich, wie z. B. ein JTAG- oder SPI-Programmiergerät, da die Firmware im Flash-Speicher des Chips gespeichert ist. Ohne Zugriff auf den physikalischen Chip und die entsprechende Hardware ist eine direkte Auslesung nicht möglich. Es gibt jedoch Ansätze, bei denen die Firmware aus Windows-Treibern extrahiert werden kann, da diese oft .CAB-Dateien enthalten, die .HEX-Dateien mit der Firmware enthalten. Diese Dateien können dann mit Tools wie 7-Zip entpackt werden. Eine solche Firmware kann dann in eine Binärdatei konvertiert werden, falls nötig.

Die Windowstreiber hätte ich vorrätig. Im Screenshot zu sehen sind die extrahierten Dateien aus der .cab Datei.



Beitrag von „schrup21“ vom 23. Dezember 2025, 11:07

Es gibt keine Firmware für BCM4360.

Diese Module werden in Linux mit der SMAC Firmware betankt.

https://wireless.docs.kernel.o...rs/drivers/brcm80211.html

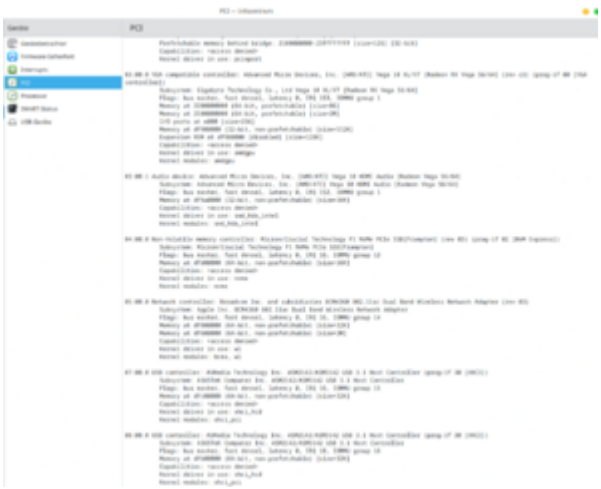
Code

1. /lib/firmware/brcm/bcm43xx-0.fw

Apple verwendet eine eigene, proprietäre Firmware, die ist in AirPort_BrcmNIC enthalten (auch die FW für die BCM943602 und BCM94350 ist da integriert).

Beitrag von „DerBeste“ vom 23. Dezember 2025, 12:42

> Zitat von karacho
>
> DerBeste Die Firmware habe ich noch nirgends gefunden. Ich habe so eine PCIe Karte im Rechner. Ist es nicht möglich die Firmware auszulesen und als .bin zu speichern?



> [Zitat von Azteca]
>
> Würde gerne die Sachen liefern, jedoch hab keine Ahnung wie man das aus der Karte kopiert.

> [Zitat von schrup21]
>
> Es gibt keine Firmware für BCM4360.
>
> Diese Module werden in Linux mit der SMAC Firmware betankt.
>
> https://wireless.docs.kernel.o...rs/drivers/brcm80211.html
>
> Code
>
> 1. /lib/firmware/brcm/bcm43xx-0.fw
>
> Apple verwendet eine eigene, proprietäre Firmware, die ist in AirPort_BrcmNIC enthalten (auch die FW für die BCM943602 und BCM94350 ist da integriert).

Hier eine Anleitung wie ihr die Firmware-ROM eines Broadcom-WLAN-Chips mit Nexmon extrahiert

Beispielchip: BCM43602

Ziel: Erzeugung einer originalen "rom.bin" direkt aus dem WLAN-Chip

1. Voraussetzungen

Bevor ihr beginnt, müssen alle folgenden Bedingungen erfüllt sein:

- Betriebssystem: Linux (Ubuntu wird empfohlen)
- Rechte: Root-Zugriff (`sudo`)
- Hardware: Broadcom-WLAN-Chip, der von Nexmon unterstützt wird (z. B. BCM43602)
- Systemstatus: Der WLAN-Chip wird vom System korrekt erkannt

Ohne diese Voraussetzungen ist der Vorgang nicht erfolgreich durchführbar.

2. System vorbereiten

Öffne ein Terminal und aktualisiere dein System:

bash

sudo apt update

sudo apt upgrade -y

Installiere anschließend alle benötigten Werkzeuge:

bash

sudo apt install -y git build-essential make gcc bc python3

Diese Pakete sind notwendig, um Nexmon zu kompilieren und Patches zu bauen.

3. Nexmon herunterladen

Wechsle in dein Home-Verzeichnis:

bash

```
cd ~
```

Klone das offizielle Nexmon-Repository:

bash

```
git clone https://github.com/seemoo-lab/nexmon.git
```

Wechsle in das Nexmon-Verzeichnis:

bash

```
cd nexmon
```

4. Nexmon-Umgebung einrichten

Initialisiere die Nexmon-Umgebung:

bash

```
source setup_env.sh
```

Wichtig:

Dieser Schritt setzt notwendige Umgebungsvariablen. Ohne ihn schlagen alle weiteren Befehle fehl.

5. Passenden Chip auswählen

Navigiere in das Patch-Verzeichnis:

bash

cd patches

Suche den Ordner deines Chips, z. B.:

bash

bcm43602

Wechsle in das Nexmon-Unterverzeichnis des Chips:

bash

cd bcm43602/nexmon

6. ROM-Dump-Patch bauen

Baue das ROM-Dump-Patch:

bash

make dump-rom

Dabei wird ein kleines Patch-Programm erzeugt, das später den ROM-Inhalt des WLAN-Chips ausliest.

7. WLAN-Interface vorbereiten

Deaktiviere das WLAN-Interface, um Konflikte zu vermeiden:

bash

sudo ifconfig wlan0 down

Falls dein Interface anders heißt (zB. "wlp2s0"), passe den Namen entsprechend an.

8. ROM-Dump ausführen

Starte nun den ROM-Dump:

bash

sudo make run-dump-rom

Was dabei passiert:

- Der WLAN-Chip wird initialisiert
- Der ROM-Inhalt wird in den RAM gespiegelt
- Die Daten werden ausgelesen
- Der Dump wird in eine Datei geschrieben

Der Vorgang dauert nur wenige Sekunden.

9. Ergebnis prüfen

Nach erfolgreichem Abschluss findest du eine Datei wie:

bash

rom.bin

oder

bash

fw_rom.bin

Prüfe die Dateigröße:

bash

ls -lh rom.bin

Erwartet:

- Größe zwischen 500 KB und 1 MB

Fehlerfall:

- Datei ist leer oder 0 Byte → Dump fehlgeschlagen

10. ROM-Datei sichern

Kopiere die Datei auf den USB-Stick

bash

cp rom.bin ~/firmware_bcm43602_rom.bin

Diese Datei ist nun die Grundlage für die Analyse- oder Patch-Arbeiten.

11. ROM-Datei analysieren (optional)

Zur Analyse der Firmware kannst du "binwalk" verwenden:

bash

sudo apt install -y binwalk

binwalk firmware_bcm43602_rom.bin

Damit lassen sich Code-Bereiche und Strukturen identifizieren – insbesondere relevant für Reverse Engineering.

12. Rechtliche Hinweise

- Die Firmware ist proprietär

- Nutzung ausschließlich zu Forschungs- und privaten Zwecken

- Keine öffentliche Weiterverbreitung !

- Broadcom- und ggf. Apple-Lizenzen gelten weiterhin

Kurzfassung

- Linux vorbereiten

- Nexmon herunterladen und initialisieren

- Broadcom-Chip auswählen

- ROM-Dump-Patch bauen

- ROM aus dem WLAN-Chip auslesen

- "rom.bin" sichern und optional analysieren

Ergebnis:

Eine originale WLAN-Firmware direkt aus der Hardware.

Edit:

Lasst euch aber Zeit. Ich arbeite derzeit an einem anderen Projekt und kann mich erst dann darum kümmern, wenn ich das neue Projekt abgeschlossen habe.

Beitrag von „Azteca“ vom 23. Dezember 2025, 13:17

OK. Wenn ich es schaffe melde ich mich wieder.

Beitrag von „karacho“ vom 23. Dezember 2025, 13:44

Azteca Ein Live Linux wird nicht funktionieren, weil du die Abhängigkeiten installieren musst.

DerBeste Hab nexmon mal geclont, aber da im Repository gibts leider keine patchverzeichnisse für bcm4360 oder bcm43602.

Beitrag von „schrup21“ vom 23. Dezember 2025, 15:28

WLAN Module enthalten keine Firmware, die wird vom OS in das Modul geladen - warum sollte man also eine Firmware aus dem Modul extrahieren?

Das bedeutet doch die Firmware ist vorhanden.

In Linux liegt die in /lib/firmware

Beitrag von „DerBeste“ vom 23. Dezember 2025, 15:39

Hier ist das vollautomatisierte Shell-Skript für Linux.

Dieses Skript führt den gesamten Prozess (Abhängigkeiten installieren, Nexmon klonen, Umgebung einrichten, Chip auswählen, Dump durchführen) automatisch durch.

Anleitung:

Kopiere die Datei extract_firmware_linux.sh auf deinen Linux-Rechner (zB. per USB-Stick).

Mache sie ausführbar: chmod +x extract_firmware_linux.sh

Führe sie als Root aus: sudo ./extract_firmware_linux.sh

Das Skript fragt dich interaktiv, welchen Chip du auslesen möchtest und wie dein WLAN-Interface heißt, und erledigt den Rest für dich.

Beitrag von „mhaeuser“ vom 24. Dezember 2025, 00:51

DerBeste Das ist jetzt wirklich nicht böse gemeint, aber probiere doch, die Sache ordentlich zu lernen. LLMs sind nicht auf dem Level, realistische Treiber zu entwickeln. Wenn ein erfahrener Entwickler alles vorkaut, kommt vielleicht wenigstens *etwas* raus, aber so nicht. Ich mache gerade Compilerentwicklung - deutlich angenehmer als Treiberentwicklung - und nichtmal dort sind die LLMs eine große Hilfe, wenn es nicht gerade um Boilerplate oder sehr isolierte Bugs geht.

Außerdem ist so der Lerneffekt fast 0. Deine Posts widersprechen sich immer wieder und dir fällt es entweder nicht auf oder es ist dir egal (wäre schade). Den allgemein bekannten Fakten wie der FW-Geschichte von schrup21 widersprechen sie ebenfalls. Erst gibt es einen halbfertigen Prototypen, der lädt, dann auf einmal können DriverKit-Treiber gar nicht geladen werden, dann geht es auf einmal doch wieder. Ich hoffe, das fällt nicht nur mir auf.

Es gibt kompetente Netzwerkentwickler wie Mieze in der Community und es täte allen gut,

eher auf sie als auf halluzinierende KIs zu hören. Soll jeder sein Lieblingsspielzeug nutzen, aber meine Studiokopfhörer machen mich auch nicht zum Spezialisten für Mix&Master…

Beitrag von „DerBeste“ vom 24. Dezember 2025, 03:55

> Zitat von mhaeuser
>
> DerBeste Das ist jetzt wirklich nicht böse gemeint, aber probiere doch, die Sache ordentlich zu lernen. LLMs sind nicht auf dem Level, realistische Treiber zu entwickeln. Wenn ein erfahrener Entwickler alles vorkaut, kommt vielleicht wenigstens *etwas* raus, aber so nicht. Ich mache gerade Compilerentwicklung - deutlich angenehmer als Treiberentwicklung - und nichtmal dort sind die LLMs eine große Hilfe, wenn es nicht gerade um Boilerplate oder sehr isolierte Bugs geht.
>
> Außerdem ist so der Lerneffekt fast 0. Deine Posts widersprechen sich immer wieder und dir fällt es entweder nicht auf oder es ist dir egal (wäre schade). Den allgemein bekannten Fakten wie der FW-Geschichte von schrup21 widersprechen sie ebenfalls. Erst gibt es einen halbfertigen Prototypen, der lädt, dann auf einmal können DriverKit-Treiber gar nicht geladen werden, dann geht es auf einmal doch wieder. Ich hoffe, das fällt nicht nur mir auf.
>
> Es gibt kompetente Netzwerkentwickler wie Mieze in der Community und es täte allen gut, eher auf sie als auf halluzinierende KIs zu hören. Soll jeder sein Lieblingsspielzeug nutzen, aber meine Studiokopfhörer machen mich auch nicht zum Spezialisten für Mix&Master…

Hallo mhaeuser,

danke für deine offenen Worte. Ich nehme das "nicht böse gemeint" mal so an, hier aber Klartext von meiner Seite:

LLMs & Technik: Ich nutze KI als Werkzeug für Boilerplate und Ideen, nicht als Ersatz für meinen Kopf. Dass KI bei Treibern halluziniert, ist mir absolut bewusst; das Debugging und die Logik liegen bei mir.

Widersprüche: Beim Reverse Engineering und Prototyping geht es oft einen Schritt vor und zwei oder drei zurück. Das kenne ich aus meiner jahrzehntelangen Tätigkeit als Konstrukteur nur zu gut. Wer Fehler nicht wagt und öffentlich macht, macht auch keinen Fortschritt.

Experten & "Geht nicht": Ich habe großen Respekt vor Mieze & Co. Mein Projekt ist ein paralleler Versuch, kein Angriff. Dass Dinge als "unmöglich" gelten, war damals beim AMD-Kernel auch hier und in anderen deutschen Foren ein tägliches Gerede, und genau das ist mein Antrieb gewesen.

Zum Thema Kritik: Da ich meinen Source Code bisher noch gar nicht veröffentlicht habe, ist deine vernichtende Kritik für mich faktisch nicht nachvollziehbar. Es wirkt eher wie ein Ausdruck von Missgunst als eine fachliche Einschätzung. Hättest du den Code im Ganzen gesehen und dann deine konstruktive Kritik geäußert, um die Entwicklung womöglich voranzutreiben, wäre das eine Hilfe für die Community gewesen. Aber so, ohne jede Basis, ist es das auf keinen Fall.

Ich nehme das exakt so für mich auf und beende die Diskussion auf diesem Portal hiermit.

Beste Grüße!

Beitrag von „Azteca“ vom 24. Dezember 2025, 10:02

@DerBeste Sei nicht so empfindlich, es gibt immer Trödler die nichts bringen aber kritisieren, das können die.

Von deiner Seite hat man zumindest die Hoffnung, dass man die Broadcom WIFI-Module wieder nutzen kann. Mach weiter!

Beitrag von „schrup21“ vom 24. Dezember 2025, 12:55

Marvin Häuser, u.A. im Team von Acidanthera.

Azteca ernsthaft?

Beitrag von „Azteca“ vom 24. Dezember 2025, 14:59

Ist mir egal, wenn er so gut ist, kann er doch helfen und nicht kritisieren. Und wenn er besser weisst wie das gemacht wird, warum macht er das nicht. Rhetorische Frage.

DerBeste macht etwas worauf wir warten, und **mhaeuser** führt dazu dass er das womöglich nicht mehr weiterführt.

Beitrag von „mhaeuser“ vom 24. Dezember 2025, 15:26

Azteca Welche Kritik? Der jedem erfahrenen Entwickler offensichtliche Fakt, dass sich sowas mit LLMs nicht schultern lässt, ist ein Hinweis zum Vorgehen. Wenn dir (wie mir letztens) im Baumarkt vom Fachmann gesagt wird, dass du mit einer Hieb-3-Feile nicht weit kommst, bekommst du dann auch einen Trotzanfall und lädst ihn zu dir nach Hause ein, er soll selber machen? Ich habe eine Hieb-2 dazu geholt und habe es nicht bereut, ganz ohne zertrümmertes Ego. Danke, Herr Fachmann!

Im “Prototypen” von InsanelyMac waren doch schon im reinen Boilerplate-Code halluzinierte APIs und oben ging es auf einmal um Kernel-Header, obwohl der einzige Existenzzweck von DriverKit der Umzug weg vom Kernel ist. Jetzt auf einmal wird eine Firmware von einem Gerät ohne persistenten Speicher extrahiert. Sowas muss einem auffallen, wenn man weiterkommen will…

Beitrag von „Mieze“ vom 24. Dezember 2025, 15:35

Azteca Eigentlich wollte ich mich aus dieser Diskussion heraushalten, weil jeder machen kann

was er will, ob es Sinn macht oder nicht, aber Marvin hat 100% Recht!

Beitrag von „Azteca“ vom 24. Dezember 2025, 16:12

Ja aber dir **mhaeuser** ist doch die Reaktion von DerBeste aufgefallen, im Endeffekt ist deine Belehrung doch kontraproduktiv.

Das kann man doch anders machen oder? Ein Hinweis hätte gereicht.

Beitrag von „iMarc“ vom 25. Dezember 2025, 23:28

ich kann es nicht glauben was hier passiert. wie können sich fähige entwickler nur erlauben das offensichtliche anzusprechen? wo kommen wir da nur hin..

Beitrag von „Arkturus“ vom 26. Dezember 2025, 09:34

Es gibt hier nur wahr und falsch, so wie eins und null. Ich sehe nur, das die Kritik an der Wahrheit vor allem deren Urheber trifft. Vor allem dürfte man ernsthafte Zweifel haben, ob diejenigen die eigentlichen Hintergründe überhaupt verstehen.
Es wäre schön und hilfreich, hier die Schärfe rauszunehmen und drüber nachzudenken.

In diesem Sinne wünsche ich einen besinnlichen zweiten Feiertag.