```
# kldload ndis
# kldload if_ndis
# kldload wifi1234_sys
```

Auch hier gilt es wieder zu beachten, dass die Reihenfolge der kldload-Kommandos sehr wichtig ist! Bei Missachtung droht ein „kernel panic“. Wie im vorherigen Abschnitt beschrieben sollte man auch wieder Eintragungen in */boot/loader.conf* vornehmen:

```
ndis_load=“YES“
if_ndis_load=“YES“
wifi1234_sys_load=“YES“
```

Nun kann das System neu gestartet und die Netzwerkkarte konfiguriert werden.

Wichtig: „Project Evil“ funktioniert sowohl auf Systemen mit 32-bit oder 64-bit Prozessoren von AMD oder Intel. Voraussetzung ist allerdings, das die Treiber entweder 32-bit oder 64-bit sind.

WPA – Sicherheit für das WLAN

Um sich vor unliebsamen Gästen im eigenen Funknetz zu schützen, ist WPA eine gute Ausgangsbasis. WPA (Wi-Fi protected access) ist ein Sicherheitsprotokoll, das in IEEE802.11 Netzwerken eingesetzt wird. Es ist im Vergleich zu WEP ein Sicherheitsgewinn, da es Mechanismen zur Authentifizierung implementiert.

Die Kontrolle der Verbindungen und Authentifizierung wird durch das Programm *wpa_supplicant* abgewickelt.

Konfiguration: Die Konfigurationsdatei ist */etc/wpa_supplicant.conf*. Ein Beispiel, bei dem sich ein Client am Access-Point „Bsp_AP“ mit der Benutzerkennung „username“ und dem Passwort „password“ anmeldet:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
eapol_version=1
ap_scan=1
fast_reauth=1
network={
  ssid="Bsp_AP"
  proto=WPA
  key_mgmt=WPA-EAP
  auth_alg=LEAP
  identity="username"
  password="password"
  priority=1
}
```

Damit beim Systemstart das WLAN-Interface korrekt konfiguriert wird, muß in der zentralen Konfigurationsdatei /etc/rc.conf ifconfig_ath0=“WPA DHCP“ eingetragen sein, falls DHCP gewünscht ist, ansonsten ist ifconfig_ath0=“WPA <IP-Adresse> <Netzmaske>“ zu schreiben.

Weitere Sicherheitsfeatures

Neben dem Standard-WPA gibt es noch Erweiterungen, die die Sicherheit noch steigern können, sofern die Gegenstation dies unterstützt!

WPA mit EAP-TLS: EAP bietet keine Verschlüsselung, sondern stellt eine Schnittstelle für andere Authentifizierungsmethoden und Verschlüsselungstechniken dar. Bei EAP-TLS ist dies Transport Layer Security. Beide Seiten benötigen ein Zertifikat, um sich zu authentifizieren.

WPA mit EAP-TTLS: Hier benötigt der Client nicht unbedingt ein Zertifikat, weil ein SSL-Tunnel aufgebaut wird. Für den Server ist ein Zertifikat aber zwingend! EAP-TTLS benutzt den verschlüsselten TLS-Tunnel für den Transport der Authentifizierungsinformationen.

WPA mit EAP-PEAP: Dieses Protokoll wurde als Alternative zu EAP-TTLS entwickelt. Es gibt zwei Methoden, wobei die am meisten benutzte *PEAPv0/EAP-MSCHAP-v2* ist. Es funktioniert ähnlich wie EAP-TTLS, wobei der Benutzername unverschlüsselt übertragen wird, das Passwort aber wird im TLS-Tunnel zum Server übertragen.

Weitere Informationen

Die beste Unterstützung seitens FreeBSD für Wireless-LAN bietet im Moment der Atheros-Treiber. Bitte lesen Sie auch die manual page des Atheros-Treibers „ath“. Beachten Sie weiterhin, dass die Hersteller teilweise willkürlich die verwendeten Chipsätze ohne Vorankündigung ändern. Wenn Sie beim Kauf sichergehen wollen, dass ein Atheros-Chipsatz integriert ist, dann empfehlen wir einen Blick auf die Homepage von Atheros:

http://customerproducts.atheros.com/customerproducts/ResultsPageBasic.asp

Dort kann man durch Eingabe der Produktdaten prüfen lassen, ob ein Atheros-Chipsatz verwendet wird.

Wichtige manual pages:

wlan(4), wlan_wep(4), wlan_ccmp(4) und wlan_tkip(4)

wpa_supplicant(8) und wpa_supplicant.conf(5), ath(4)

Liste der nativ unterstützten WLAN-Karten: wlan(4)

http://www.freebsd.org/doc/en/books/handbook/network-wireless.html

FreeBSD

Wireless-LAN

Der Netzwerkstandard *Wireless LAN* – kurz WLAN – hat sich in den letzten Jahren als Alternative zu einer drahtgebundenen Vernetzung etabliert. FreeBSD bietet Unterstützung von WLAN in Form von nativen Treibern, aber auch im Rahmen des Projekts „Evil“ über Windows-Treiber.

WLAN basiert auf dem IEEE802.11-Standard und arbeitet in den Frequenzbändern 2,4GHz oder 5,0GHz. WLAN-Netzwerke sind auf zwei Arten organisiert:

infrastructure mode: Ein Master – access point (AP) genannt – und diverse Clients bilden ein Netzwerk, wobei die gesamte Kommunikation über den access point abgewickelt wird (BSS)

adhoc mode: Es gibt keinen Master und alle Stationen kommunizieren direkt miteinander (IBSS).

Bei Funkverbindungen besteht immer die Gefahr des Abhörens durch Unbefugte, oder das Einklinken unberechtigter Benutzer. Um dies zu verhindern, sind im Wesentlichen zwei Mechanismen geschaffen worden:

WEP-Modus: einfacher, aber wirkungsloser Verschlüsselungsmodus, der im IEEE802.11-Standard festgeschrieben wurde.

WPA-Modus (WiFi-Protected-Access): wesentlich besserer Verschlüsselungsmodus mit Authentifizierung. Der Modus ist im Standard IEEE802.11i festgeschrieben.

Um Multimedia-Anwendungen eine bessere Plattform zu bieten, wurde *WME/WMM* entwickelt. Zur besseren Ausnutzung der Bandbreite sind Protokolle für die Qualitätsüberwachung der Signale (QoS) entwickelt worden.

Seit FreeBSD 6.0 werden WLAN-Netzwerke nach IEEE802.11 unterstützt genauso wie der WPA-Modus nach IEEE802.11i, WME/WMM und QoS. Wobei letzteres nicht von allen WLAN-Treibern genutzt wird. Das Betriebssystem kann dabei auch als Access-Point oder als Bridge zum Internet arbeiten.

Nativer WLAN-Support in FreeBSD

FreeBSD bietet für viele WLAN-Chipsätze nativen Treiber-Support an. Neben der direkten Implementierung in den Kernel gibt es auch das einfachere Laden eines Kernel-Moduls mittels kldload. Am Beispiel einer WLAN-Karte mit Atheros-Chipsatz soll die Implementierung in den Kernel beschrieben werden:

cd /usr/src/sys/<Architektur>/conf

cp GENERIC WLANKERNEL

Ergänzungen im FreeBSD-Kernel für WLAN-Fähigkeit:

device ath

device ath_hal

device ath_rate_sample

device wlan

device wlan_wep

device wlan_ccmp

device wlan_tkip

device wlan_acl

device wlan_xauth

Speichern der Konfiguration, anschließend kompilieren und installieren:

cd /usr/src

make buildkernel KERNCONF=WLANKERNEL

make installkernel KERNCONF=WLANKERNEL

WLANKERNEL ist nur ein frei wählbarer Platzhalter. Beim Kompilieren und Installieren ist auf Fehlermeldungen zu achten! Vor dem Erstellen eines neuen Kernels bitte Sicherheitskopie des alten Kernels nicht vergessen!

Einfacher ist das Laden von Kernelmodulen. Am Beispiel oben genannter WLAN-Karte soll dies gezeigt werden:

In der Datei /boot/loader.conf:

if_ath_load="YES"

wlan_wep_load="YES"

wlan_ccmp_load="YES"

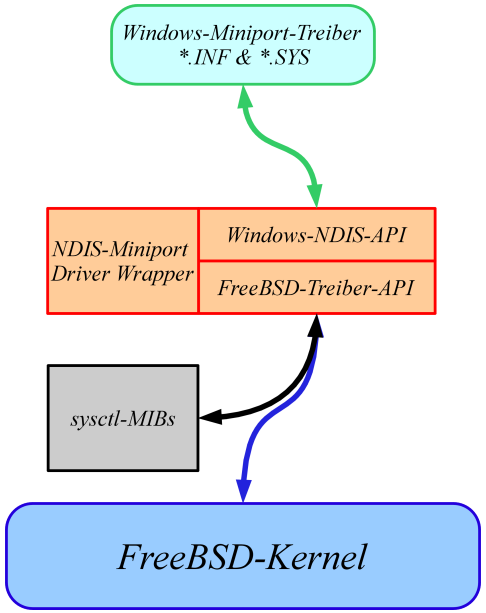
wlan_tkip_load="YES"

Alle zusätzlich benötigten Module werden automatisch mit geladen.

„Project Evil“

Falls keine nativen FreeBSD-Treiber zur Verfügung stehen, hilft „Project Evil“. Eines der Probleme, welche die freie Software-Gemeinde plagen, ist die Verfügbarkeit von Gerätetreibern. Solange ein Betriebssystem keine deutliche Marktdurchdringung hat, macht es keinen ökonomischen Sinn, Gerätetreiber dafür zu entwickeln. Viele Hersteller stellen noch nicht einmal eine Dokumentation zur Verfügung, die es erlaubt Treiber zu schreiben.

Im Fall von WiFi-Karten kann dies zu einem Problem werden. Es ist sehr schwierig zu sagen, welcher Chipsatz zukünftig für eine Karte verwendet wird, manche Hersteller ändern die Hardware ohne die Produktbezeichnung zu modifizieren. So kann die Suche nach einer WiFi-Karte für FreeBSD schwierig werden.



Die FreeBSD-Gemeinde hat daher das „Project Evil“ ins Leben gerufen. Das ist eine teilweise Einbindung des Windows-APIs, welche es erlaubt, Windows-Treiber für Netzwerkkarten zu verwenden (siehe Abbildung).

Wie funktioniert „Project Evil“?

„Project Evil“ stellt Basisfunktionen zur Verfügung, die von Windows Netzwerkkarten-Treibern verwendet werden. Diese Funktionen werden intern in das FreeBSD-Treibermodell umgesetzt. Für den Treiber sieht es so aus, als ob er in einer Windows-Umgebung laufen würde.

Aus Sicht des Betriebssystems ist es ein natives FreeBSD-Kernelmodul. Unter Windows besteht ein WiFi-Treiber aus drei Komponenten. Der Treiber selber, der meist die Erweiterung .sys hat. Eine .inf -Datei, die Informationen zum Treiber enthält und schließlich gibt es noch eine Datei mit einer Kopie der Karten-Firmware.

Traditionell wird die Firmware einer Netzwerkkarte in einem ROM gebrannt. Später wurde erkannt, dass eine Möglichkeit einer Aktualisierung gegeben sein muss, und so wurde die Firmware in einem Flash-ROM gespeichert. In modernen Billigkarten wird kein Flash-ROM mehr verwendet und so wird die Firmware im RAM hinterlegt. Das bedeutet, dass der Treiber diese erst laden muss, bevor die Karte benutzt werden kann. Um die Sache noch komplizierter zu machen, haben einige Treiber separate Firmware für den Ethernet-Controller und für den WLAN-Teil. In den meisten Fällen haben Firmware-Dateien die Endung .bin.

Erstellen der Kernelmodule

Es wird eine Kopie des Windows-Treibers benötigt. Dieser befindet sich auf der Treiber-CD des Herstellers oder auf dessen Internet- Seite. Nun alle Dateien mit der Erweiterung .sys , .inf und .bin in ein Verzeichnis kopieren. Für dieses Tutorial verwendet der Autor eine Wifi-Karte WIFI1234, so dass man man die Treibernamen durch die eigenen ersetzen muss.

Diese Treiber werden mitgeliefert:

wifi1234.bin: Firmware der Netzwerkkarte

wifi1234funk.bin: Firmware des Funkteils

WIFI1234.INF: Treiberinformation

wifi1234.sys: Der Treiber selber

Vorgehensweise

Ab FreeBSD 5.4 werden für „Project Evil“ keine Kernel-Quelltexte mehr benötigt. Das ndis- und if_ndis -Modul sollten bereit installiert sein. Es muss nur noch ein Modul generiert werden, welches den Treiber und die Firmware enthält. Dies wird vom Assistent *ndisgen* gehandhabt.

 # ndisgen

Das Programm fragt nach den Verzeichnissen, in denen Treiber und Firmware abgelegt sind. Bitte beachten, dass zwischen Groß- und Kleinschreibung unterschieden wird und es muss auch der volle Verzeichnisname angegeben werden. Als Ergebnis erhält man ein einzelnes Modul (.ko). In diesem Fall ist es wifi1234_sys.ko. Das Modul ist nur noch nach */boot/kernel* zu kopieren.

 # cp wifi1234_sys.ko /boot/kernel/