Load all modules with kldload:

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

The sequence of kldload commands is very important! Otherwise a kernel panic is the result! As described in the last section it is better to put everything in the file */boot/loader.conf*:

```
ndis_load="YES"
if_ndis_load="YES"
wifi1234_sys_load="YES"
```

Now you may restart the system and configure the the network card.

Important: Project Evil works on both 32-bit and 64-bit systems of AMD and Intel. The drivers have to be 32-bit or 64-bit though.

WPA – Security forWLAN

WPA is a good way to protect your wireless network from intruders. Compared to WEP it offers much enhanced security and also authentication.

The tool *wpa_supplicant* controls the connections and authentification.

Configuration is done in the file '*/etc/wpa_supplicant.conf*'. In following example you will see a client connect to the access point '*EXPL_AP*', with the login '*username*' and the password '*password*'.

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

To make sure the WLAN interface is correctly configured at system startup the following line needs to added to the central configuration file '*/etc/rc.conf*':

```
ifconfig_ath0="WPA DHCP"
```

If you want DHCP, otherwise please add a line like for wired LAN:

```
ifconfig_ath0="WPA IP-Address Netmask"
```

Additional security features:

Besides of standard WPA there exist some extensions which improve the security, but the receiving station has to support this!

WPA with EAP-TLS: EAP doesn't offer encryption but offers an interface for other authentication and encryption methods. In this case it's Transport Layer Security (TLS). Both sides need a certificate for mutual authentication.

WPA with EAP-TTLS: Here the client doesn't require a certificate because an SSL-tunnel is created. But for the server a certificate is mandatory. EAP-TTLS uses the encrypted TLS-tunnel to transport the authentication information in EAP.

WPA with EAP-PEAP: This protocol was developed as an alternative to EAP-TTLS. There are two methods, the one used most often is PEAPv0/EAP-MSCHAP-v2. It's similar to EAP-TTLS, but the user name is sent in clear text, while the password is sent over the TLS-tunnel to the server.

Additional Information

The Atheros driver in FreeBSD offers most features available in the WLAN standards. Please also refer to the manual pages for the Atheros driver (ath). Be aware that vendors often change the chipsets in their cards without announcement! If you want to make sure you have an Atheros chipset in your card, please visit this Atheros' homepage:

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

Important manual pages:

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

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

List of natively supported WLAN cards: wlan(4)

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

FreeBSD

Wireless-LAN

The network standard Wireless LAN (WLAN) has established itself in the past few years as a viable alternative to wired networking. FreeBSD supports WLAN with native drivers and with Windows drivers in the connection with "Project Evil".

WLANs are organized in two ways:

infrastructure mode: A master, called Access Point (AP), and multiple clients form a network where all the communication goes through the access point. This is also known as Basic Service Set (BSS).

adhoc mode: There's no master and all stations communicate directly with each other. This is known as Independent Basic Service Set (IBSS).

Tracing of radio links and connections of unauthorized persons is a great risk. Mainly two mechanisms are created to protect this:

WEP mode: simple but ineffective encryption mode which is described in the IEEE802.11 standard.

WPA mode (Wifi protected access): much better encryption mode with authentification. This mode is described in the IEEE802.11i standard.

To improve support for multimedia applications WME and WMM were developed, as were protocols to improve bandwidth usage with Quality of Service (QoS) functionality.

FreeBSD 6.0 supports either WLAN networks based on the IEEE802.11 standard or the WPA mode, the WME/WMM and QoS as described in IEEE802.11i. The last one is not supported by all drivers. The operating system works either as access point or as a bridge to the internet.

Native WLAN support in FreeBSD

FreeBSD offers native driver support for many WLAN chipsets. Beside the direct implementation in the kernel there is an easier way to load a kernel module with kldload. The following shows an example for the implementation of a device driver for an WLAN card with Atheros chipset:

```
# cd /usr/src/sys/<platform>/conf
# cp GENERIC WLANKERNEL
```

Now edit the file 'WLANKERNEL' and add following lines:

```
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
```

Now save the configuration, compile and install it:

```
# cd /usr/src
# make buildkernel KERNCONF=WLANKERNEL
# make installkernel KERNCONF=WLANKERNEL
```

You may also use a different name than 'WLANKERNEL'. Make sure there are no error messages during compilation and installation. Do not forget to backup the old kernel before creating a new one!

Loading of kernel modules is much easier. Below you find an example with the same card as above:

Put in the file /boot/loader.conf:

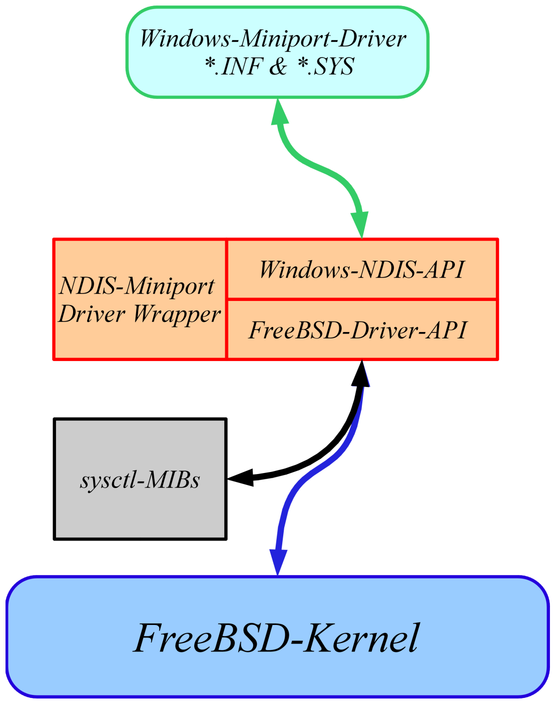
```
if_ath_load="YES"
wlan_wep_load="YES"
wlan_ccmp_load="YES"
wlan_tkip_load="YES"
```

All additional modules that are needed will be loaded automatically when required.

„Project Evil“

If no native FreeBSD drivers are available maybe “Project Evil” can help. One of the problems plaguing the free software community is the availability of device drivers. As long as an operating system doesn't have sufficient market penetration it makes no economic sense to develop drivers for it.

Many vendors don't even offer sufficient documentation to allow writing drivers. In the case of WLAN-cards this can be quite a problem because it's very hard to predict which chipset will be used in a future card. Many vendors change the chipsets without changing the card's name or version.



That's why the FreeBSD community has started “Project Evil”. This is an integration of part of the Windows-API which allows using Windows drivers for network cards (not only WLan cards).

How does „Project Evil“ work?

It offers basic functions used by Windows network device drivers. These functions are translated internally to the FreeBSD driver model. The network driver thinks it's in a Windows environment. FreeBSD though sees it as a FreeBSD kernel module. In Windows a WLAN driver consists of three components: the driver itself (often with the filename extension '.sys'), an information file (often with the filename extension '.inf') and a file with a copy of the card's firmware.

Traditionally the firmware of a network card is burned into the ROM of the card. Later the need for being able to update was recognized and the firmware was stored in a Flash-ROM.

In modern cheap cards no Flash-ROM is used anymore and the firmware is stored in RAM instead. That means that the driver first has to load it before the card can use it. Some drivers have separate firmware for the Ethernet controller an WLAN parts. In most cases the firmware files have the filename extension '.bin'.

Creation of kernel modules

A copy of the Windows driver is needed. It can be found on the driver CD of the manufacturer or on its internet site. Now copy all files with the extension .sys, .inf and .bin in one directory.

In this tutorial we will use the card WIFI1234 as an example, so replace the driver names with the ones of your card's. Following files were used here:

wifi1234.bin: firmware of the card

wifi1234funk.bin: firmware of the transmitter

WIFI1234.INF: driver information

wifi1234.sys: the driver itself

Procedure

As of FreeBSD 5.4 no kernel source code is required anymore for Project Evil. The ndis- and if_ndis-modules should already be installed. You just still need to generate a module containing the driver and firmware of your specific card. This is done with the program ndisgen.

```
# ndisgen
```

The program will ask for the directory containing your driver and firmware. Please be aware to give the full path to the directory you created. As a result you should get a single module (.ko). In our example we would get the module 'wifi1234_sys.ko'. Copy this file to '/boot/kernel'.

```
# cp wifi1234_sys.ko /boot/kernel/
```