WLAN Devices supported native

3Com: 3Com 3CRPAG175

AZTech: Aztech WL830PC

BayStack: BayStack 650, 660

Cisco: Aironet 802.11b wireless adapters

D-Link: DWL-A520, DWL-A650, DWL-AB650, DWL-AG520, DWL-AG650, DWL-G520B, DWL-G650B

Elecom: LD-WL54, LD-WL54AG

Farallon: SkyLINE

Fujitsu: E5454, E5454, FMV-JW481

Hewlett-Packard: P NC4000

I/O Data: WN-A54, WN-AB, WN-AG

Icom: SL-200

Linksys: WMP55AG, WPC51AB, WPC55AG

Melco: WLI-PCM

NEC: PA-WL/54AG

NEL: SSMagic

Netgear: WAB501, WAG311, WAG511, WG311, WG311T, WG511T

NetWave: AirSurfer, AirSurfer Plus, AirSurfer Pro

Nokia: C020 WLAN

Orinoco: 8470WD, 8480

Proxim: Skyline 4030, Skyline 4032

Raytheon: Raylink 2.4GHz wireless adapters

Samsung: SWL-5200N

SMC: SMC2735W

Sony: PCWA-C300S, PCWA-C500, PCWA-C700

Xircom: CreditCard Netwave

Lucent Technologies: WaveLAN/IEEE 802.11b wireless network adapters and workalikes using the Lucent Hermes, Intersil PRISM-II, Intersil PRISM-2.5, Intersil Prism-3, und Symbol Spectrum24 Chips

NCR / AT&T / Lucent Technologies: WaveLan T1-speed ISA radio LAN cards

Additional Information

The best support for WLAN is provided by the Atheros driver for the AR5210, AR5211 and AR5212 chip sets. If you need more information you may consult the Atheros manual page "ath" on the FreeBSD homepage www.FreeBSD.org. Please also keep in mind that some companies change the chip sets of their devices without notice.

To make sure there's an Atheros chip set in the WLAN adapter you want to buy, we recommend you consult the Atheros homepage before where you can find all companies using Atheros products:

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

Important links

FreeBSD-Homepage: www.FreeBSD.org

BSD-Forum (English): www.BSDForums.org

Notice:
Please make sure you protect your WLAN adequately! In the FreeBSD Handbook there's a whole chapter dedicated to this, called Wireless Networking.

If you see something not listed here or have any suggestions, please contact Daniel.Seuffert@allBSD.de, we will update this flyer. Thanks for helping us!

FreeBSD

WLan and Project Evil

Project Evil: Windows-Drivers for FreeBSD

This howto is based on the following arcticle by David Chrisnall. Thank you David!

www.pingwales.co.uk/tutorials/project-evil.html

Introducing Evil

One of the problems plaguing the Free Software community is the availability of device drivers. Unless an operating system has a significant market share, it does not make economic sense for a manufacturer to write device drivers for that system. Many manufacturers won't even provide documentation allowing open source drivers to be written, claiming that it would require disclosure of valuable intellectual property.

In the case of WiFi cards, this can be a problem. It is very difficult to tell in advance which chipset is used in a given card - some manufacturers change the hardware completely without changing the model number - and so finding a WiFi card compatible with your favourite OS can be difficult.

OpenBSD has a strong ideological attitude in this respect. If a manufacturer is not willing to release documentation, then they will not include closed-source drivers. This argument makes sense from a security point of view - if the drivers are closed then you can't audit them and so they may end up compromising the base system.

FreeBSD is more pragmatic. They include Project Evil, a partial implementation of the Windows driver API, which allows Windows drivers to be used for network cards. While not quite as useful as a native driver, they are a significant improvement over no driver at all.

How Does It Work?

Project Evil provides a set of basic functions commonly used by Windows network drivers. These functions are then translated internally to the FreeBSD driver model. To the driver, it appears that it is running in a normal Windows environment. To the OS, it appears that a native FreeBSD kernel module containing the driver is present.

On Windows, a WiFi driver comes in three components. The driver itself usually has the extension .sys. There is also a .inf file which contains information about the driver, such as the device ID of the hardware. Finally there is a copy of the driver firmware.

Traditionally, the firmware - software embedded in the device - for a network interface would be burned into ROM and shipped with the card. Then it was realised that the ability to update the firmware was desirable and so it was put in Flash, or similar. In modern, low budget, cards, the Flash is left off, and the firmware is stored in RAM. This means that the driver must load it before the card can be used.

To make matters more complicated, some drivers have separate firmware for the ethernet controller and radio portions of the firmware. Firmware files usually have the .bin extension.

Building the Kernel Modules

You will need a copy of the Windows driver. This will probably be on a CD included with your network card, or available from the manufacturer's web site. You should copy everything with a .sys, .inf, or .bin extension to /sys/modules/if_ndis.

I will use the file names of my driver for the rest of this tutorial, but you should substitute your own. The files supplied for my card are:

Fw1130.bin	Network interface firmware.
FwRad16.bin	Radio firmware.
TNET1130.INF	Driver information file.
tnet1130.sys	Driver binary.

The way of generating Project Evil kernel modules changed between FreeBSD 5.3 and FreeBSD 5.4, and unfortunately the documentation shipped with 5.4 still reflects the 5.3 method which no longer works. We will explain the new way only. It might be worth upgrading to -STABLE before you start, as work on Project Evil is constantly in progress.

The New Way

The new way doesn't require the kernel sources installed. The ndis and if_ndis kernel modules should already be installed. You will need to create one module for your card, which will contain the driver and the firmware. This is handled by an undocumented wizard called ndisgen.

```
# ndisgen
```

This will ask you for the location of your driver and firmware files. Note that they are case-sensitive and require full paths. At the end, it will create a single .ko file. In my case, this was tnet1130_sys.ko. You need to move this module to a location where it can be found by kldload, and then load it.

```
# cp tnet1130_sys.ko /boot/kernel/
# kldload ndis
# kldload if_ndis
# kldload tnet1130_sys
```

Note the order of the kldload statements. It is very important that they be performed in this order. Attempting to load the network card driver before the ndis stub driver can result in a kernel panic.

As with the old way, you load the driver at boot by adding it to /boot/loader.conf. You will need to add a line for each module of the three modules, so you should end up with something that looks like this:

```
ndis_load="YES"
if_ndis_load="YES"
tnet1130_sys_load="YES"
```

You can now reboot and have your network card available at boot time. As before, use /stand/sysinstall to set up the interface.

Notice:

„Project Evil“ works with 32- and 64-bit processors from AMD and Intel. You need to have the 32- or 64-bit drivers for Windows respectively!

Important Information

Starting with FreeBSD 6.0 you only need ndisgen!

Compiling Kernel (i386)

You can also compile your ndis driver into your kernel:

```
# cd /usr/src/sys/i386/conf
# cp GENERIC NDISKERNEL
```

You need to set the following options:

```
options NDISAPI
device ndis
device wlan # only if WLAN is used!
```

Save your configuration

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

You may choose any name for your NDIS-capable kernel, NDISKERNEL is just an example. Pay attention to any error messages while compiling and installing. Please don't forget to backup your old kernel and modules before trying to compile the new kernel (copy to /root/kernels/NDISKERNEL or elsewhere)!