NAME

wpa_supplicant - Wi-Fi Protected Access client and IEEE 802.1X supplicant

SYNOPSIS

wpa_supplicant [**-BddfhKLqqsTtuvW**] [**-i***ifname*] [**-c***config file*] [**-D***driver*] [**-P***PID_file*] [**-f***output file*]

OVERVIEW

Wireless networks do not require physical access to the network equipment in the same way as wired networks. This makes it easier for unauthorized users to passively monitor a network and capture all transmitted frames. In addition, unauthorized use of the network is much easier. In many cases, this can happen even without user's explicit knowledge since the wireless LAN adapter may have been configured to automatically join any available network.

Link-layer encryption can be used to provide a layer of security for wireless networks. The original wireless LAN standard, IEEE 802.11, included a simple encryption mechanism, WEP. However, that proved to be flawed in many areas and network protected with WEP cannot be consider secure. IEEE 802.1X authentication and frequently changed dynamic WEP keys can be used to improve the network security, but even that has inherited security issues due to the use of WEP for encryption. Wi-Fi Protected Access and IEEE 802.11i amendment to the wireless LAN standard introduce a much improvement mechanism for securing wireless networks. IEEE 802.11i enabled networks that are using CCMP (encryption mechanism based on strong cryptographic algorithm AES) can finally be called secure used for applications which require efficient protection against unauthorized access.

wpa_supplicant is an implementation of the WPA Supplicant component, i.e., the part that runs in the client stations. It implements WPA key negotiation with a WPA Authenticator and EAP authentication with Authentication Server. In addition, it controls the roaming and IEEE 802.11 authentication/association of the wireless LAN driver.

wpa_supplicant is designed to be a "daemon" program that runs in the background and acts as the backend component controlling the wireless connection. **wpa_supplicant** supports separate frontend programs and an example text-based frontend, **wpa_cli**, is included with wpa_supplicant.

Before wpa_supplicant can do its work, the network interface must be available. That means that the physical device must be present and enabled, and the driver for the device must be loaded. The daemon will exit immediately if the device is not already available.

After **wpa_supplicant** has configured the network device, higher level configuration such as DHCP may proceed. There are a variety of ways to integrate wpa_supplicant into a machine's networking scripts, a few of which are described in sections below.

The following steps are used when associating with an AP using WPA:

- **wpa_supplicant** requests the kernel driver to scan neighboring BSSes
- **wpa_supplicant** selects a BSS based on its configuration
- **wpa_supplicant** requests the kernel driver to associate with the chosen BSS
- If WPA-EAP: integrated IEEE 802.1X Supplicant completes EAP authentication with the authentication server (proxied by the Authenticator in the AP)
- If WPA-EAP: master key is received from the IEEE 802.1X Supplicant
- If WPA-PSK: **wpa_supplicant** uses PSK as the master session key
- **wpa_supplicant** completes WPA 4-Way Handshake and Group Key Handshake with the Authenticator (AP)
- **wpa_supplicant** configures encryption keys for unicast and broadcast
- normal data packets can be transmitted and received

SUPPORTED FEATURES

Supported WPA/IEEE 802.11i features:

- WPA-PSK ("WPA-Personal")
- WPA with EAP (e.g., with RADIUS authentication server) ("WPA-Enterprise") Following authentication methods are supported with an integrate IEEE 802.1X Supplicant:
  - EAP-TLS
  - EAP-PEAP/MSCHAPv2 (both PEAPv0 and PEAPv1)
  - EAP-PEAP/TLS (both PEAPv0 and PEAPv1)
  - EAP-PEAP/GTC (both PEAPv0 and PEAPv1)
  - EAP-PEAP/OTP (both PEAPv0 and PEAPv1)
  - EAP-PEAP/MD5-Challenge (both PEAPv0 and PEAPv1)
  - EAP-TTLS/EAP-MD5-Challenge
  - EAP-TTLS/EAP-GTC
  - EAP-TTLS/EAP-OTP
  - EAP-TTLS/EAP-MSCHAPv2
  - EAP-TTLS/EAP-TLS
  - EAP-TTLS/MSCHAPv2
  - EAP-TTLS/MSCHAP
  - EAP-TTLS/PAP
  - EAP-TTLS/CHAP
  - EAP-SIM
  - EAP-AKA
  - EAP-PSK
  - EAP-PAX
  - LEAP (note: requires special support from the driver for IEEE 802.11 authentication)
  - (following methods are supported, but since they do not generate keying material, they cannot be used with WPA or IEEE 802.1X WEP keying)
  - EAP-MD5-Challenge
  - EAP-MSCHAPv2
  - EAP-GTC
  - EAP-OTP
- key management for CCMP, TKIP, WEP104, WEP40
- RSN/WPA2 (IEEE 802.11i)
  - pre-authentication
  - PMKSA caching

AVAILABLE DRIVERS

A summary of available driver backends is below. Support for each of the driver backends is chosen at wpa_supplicant compile time. For a list of supported driver backends that may be used with the -D option on your system, refer to the help output of wpa_supplicant (**wpa_supplicant -h**).

wext Linux wireless extensions (generic).

wired wpa_supplicant wired Ethernet driver

roboswitch
wpa_supplicant Broadcom switch driver

bsd BSD 802.11 support (Atheros, etc.).

ndis Windows NDIS driver.

COMMAND LINE OPTIONS

Most command line options have global scope. Some are given per interface, and are only valid if at least one **-i** option is specified, otherwise they're ignored. Option groups for different interfaces must be separated by **-N** option.

-b br_ifname
Optional bridge interface name. (Per interface)

-B Run daemon in the background.

-c filename
Path to configuration file. (Per interface)

-C ctrl_interface
Path to ctrl_interface socket (Per interface. Only used if **-c** is not).

-i ifname
Interface to listen on. Multiple instances of this option can be present, one per interface, separated by **-N** option (see below).

-d Increase debugging verbosity (**-dd** even more).

-D driver
Driver to use (can be multiple drivers: nl80211,wext). (Per interface, see the available options below.)

-e entropy file
File for **wpa_supplicant** to use to maintain its internal entropy store in over restarts.

-f output file
Log output to specified file instead of stdout. (This is only available if **wpa_supplicant** was built with the CONFIG_DEBUG_FILE option.)

-g global ctrl_interface
Path to global ctrl_interface socket. If specified, interface definitions may be omitted.

-K Include keys (passwords, etc.) in debug output.

-t Include timestamp in debug messages.

-h Help. Show a usage message.

-L Show license (BSD).

-o override driver
Override the driver parameter for new interfaces.

-O override ctrl_interface
Override the ctrl_interface parameter for new interfaces.

-p Driver parameters. (Per interface)

-P PID_file
Path to PID file.

-q Decrease debugging verbosity (**-qq** even less).

-s Log output to syslog instead of stdout. (This is only available if **wpa_supplicant** was built with the CONFIG_DEBUG_SYSLOG option.)

-T Log output to Linux tracing in addition to any other destinations. (This is only available if **wpa_supplicant** was built with the CONFIG_DEBUG_LINUX_TRACING option.)

-t Include timestamp in debug messages.

-u Enable DBus control interface. If enabled, interface definitions may be omitted. (This is only available if **wpa_supplicant** was built with the CONFIG_DBUS option.)

-v Show version.

-W Wait for a control interface monitor before starting.

-N Start describing new interface.

EXAMPLES

In most common cases, **wpa_supplicant** is started with:

```
wpa_supplicant -B -c/etc/wpa_supplicant.conf -iwlan0
```

This makes the process fork into background.

The easiest way to debug problems, and to get debug log for bug reports, is to start **wpa_supplicant** on foreground with debugging enabled:

```
wpa_supplicant -c/etc/wpa_supplicant.conf -iwlan0 -d
```

If the specific driver wrapper is not known beforehand, it is possible to specify multiple comma separated driver wrappers on the command line. **wpa_supplicant** will use the first driver wrapper that is able to initialize the interface.

```
wpa_supplicant -Dnl80211,wext -c/etc/wpa_supplicant.conf -iwlan0
```

wpa_supplicant can control multiple interfaces (radios) either by running one process for each interface separately or by running just one process and list of options at command line. Each interface is separated with -N argument. As an example, following command would start wpa_supplicant for two interfaces:

```
wpa_supplicant \
-c wpa1.conf -i wlan0 -D nl80211 -N \
-c wpa2.conf -i ath0 -D wext
```

OS REQUIREMENTS

Current hardware/software requirements:

- Linux kernel 2.4.x or 2.6.x with Linux Wireless Extensions v15 or newer
- FreeBSD 6-CURRENT
- Microsoft Windows with WinPcap (at least WinXP, may work with other versions)

SUPPORTED DRIVERS

Linux wireless extensions

In theory, any driver that supports Linux wireless extensions can be used with IEEE 802.1X (i.e., not WPA) when using ap_scan=0 option in configuration file.

Wired Ethernet drivers

Use ap_scan=0.

BSD net80211 layer (e.g., Atheros driver)

At the moment, this is for FreeBSD 6-CURRENT branch.

Windows NDIS

The current Windows port requires WinPcap (http://winpcap.polito.it/).
See README-Windows.txt for more information.

wpa_supplicant was designed to be portable for different drivers and operating systems. Hopefully, support for more wlan cards and OSes will be added in the future. See developer.txt for more information about the

design of wpa_supplicant and porting to other drivers. One main goal is to add full WPA/WPA2 support to Linux wireless extensions to allow new drivers to be supported without having to implement new driver-specific interface code in wpa_supplicant.

ARCHITECTURE

The **wpa_supplicant** system consists of the following components:

wpa_supplicant.conf
: the configuration file describing all networks that the user wants the computer to connect to.

wpa_supplicant
: the program that directly interacts with the network interface.

wpa_cli
: the client program that provides a high-level interface to the functionality of the daemon.

wpa_passphrase
: a utility needed to construct *wpa_supplicant.conf* files that include encrypted passwords.

QUICK START

First, make a configuration file, e.g. */etc/wpa_supplicant.conf*, that describes the netw orks you are interested in. See **wpa_supplicant.conf(5)** for details.

Once the configuration is ready, you can test whether the configuration works by running **wpa_supplicant** with following command to start it on foreground with debugging enabled:

```
wpa_supplicant -iwlan0 -c/etc/wpa_supplicant.conf -d
```

Assuming everything goes fine, you can start using following command to start **wpa_supplicant** on background without debugging:

```
wpa_supplicant -iwlan0 -c/etc/wpa_supplicant.conf -B
```

Please note that if you included more than one driver interface in the build time configuration (.config), you may need to specify which interface to use by including -D<driver name> option on the command line.

INTERFACE TO PCMCIA-CS/CARDMRG

For example, following small changes to pcmcia-cs scripts can be used to enable WPA support:

Add MODE="Managed" and WPA="y" to the network scheme in */etc/pcmcia/wireless.opts*.

Add the following block to the end of **start** action handler in */etc/pcmcia/wireless*:

```
if [ "$WPA" = "y" -a -x /usr/local/bin/wpa_supplicant ]; then
/usr/local/bin/wpa_supplicant -B -c/etc/wpa_supplicant.conf -i$DEVICE
fi
```

Add the following block to the end of **stop** action handler (may need to be separated from other actions) in */etc/pcmcia/wireless*:

```
if [ "$WPA" = "y" -a -x /usr/local/bin/wpa_supplicant ]; then
killall wpa_supplicant
fi
```

This will make **cardmgr** start **wpa_supplicant** when the card is plugged in.

SEE ALSO

wpa_background(8) wpa_supplicant.conf(5) wpa_cli(8) wpa_passphrase(8)

LEGAL