Open1x User's Guide

Chris Hessing

Nick Petroni

Bryan Payne

Nick Petroni

Terry Simons

Open1x User's Guide

by Chris Hessing, Nick Petroni, Bryan Payne, Nick Petroni, and Terry Simons

Table of Contents

List of Tables

Chapter 1. About Open1x And This Guide

1.1. General Overview

This work funded by a grant from National Institute of Standards and Technology Critical Infrastructure Grants Program.

This software allows a GNU/Linux or BSD workstation to authenticate with a RADIUS server using 802.1X and various EAP protocols. The intended use is for computers with wireless LAN connections to complete a strong authentication before joining the network.

Note: BSD support is not yet complete.

This provides a good complement to WEP, which provides confidentiality. Even though it is well documented that WEP has technical flaws, it is still better than simply sending data in the clear. Therefore, we recommend using this software (802.1x) for authentication *and* WEP for confidentiality. And, as always, be prepared to update your network(s) as better security solutions become available.

Note: At this time Xsupplicant can be used in conjunction with wpa_supplicant to provide WPA keying, but does not currently provide its own WPA hooks.

1.2. Do I Need Open1x?

The short answer is that if you need to authenticate to an 802.1X-enabled network using Linux, then Open1x is probably for you. The Open1x project provides 802.1X functionality for the Linux operating system. 802.1X is an IEEE standard (ratified in 2001) that provides port-based authentication at layer 2 of the OSI model. 802.1X prevents unauthorized network access until appropriate credentials are supplied to access the network. The Open1x project provides the necessary software to connect to an 802.1X-enabled network.

1.3. Supplicant versus Authenticator

The Open1x project contains source code for both the "Supplicant" and "Authenticator" pieces of the 802.1X standard. This document will only focus on the Open1x Supplicant (xsupplicant), as the Authenticator isn't being actively worked on at this point in time.

1.4. Purpose Of This Guide

This guide is aimed towards both the general user, and the system administrator with the intent of explaining how to install and configure xsupplicant.

Chapter 2. Supported Platforms

2.1. About Xsupplicant

Xsupplicant is designed to work with Linux. Early versions of xsupplicant also supported *BSD and Mac OS X, but this support was pulled out when xsupplicant was rewritten.

Mac OS X support was pulled because Apple is now providing a built-in supplicant as of Mac OS X 10.3 (Panther). *BSD support was initially removed largely due to a lack of active *BSD development. Some *BSD code does remain, however, and we encourage any *BSD developers out there to test xsupplicant and submit patches or file bug reports to improve *BSD support. There has been talk of adding Mac OS X support back into xsupplicant, because Apple's client only works with their own Airport cards. Xsupplicant could potentially fill the gap left by Apple for those users that wish to use 3rd party cards, or wireless standards not supported by Apple, such as 802.11a, but such support would require 3rd party APIs to properly handle Dynamic WEP.

The Open1x team would like to reprovide support for *BSD platforms, but doing so will require some additional hacking on the codebase. This project is maintained in our spare time, and we already feel stretched, so we hope you understand our current dilemma in providing *BSD support. If you are interested in helping us with *BSD support, please let us know.

Xsupplicant releases are primarily tested and developed on Slackware Linux.

2.2. EAP Support

Xsupplicant 1.0.1 supports the following EAP types:

- EAP-AKA
- EAP-GTC
- EAP-MD5
- EAP-MSCHAPv2
- EAP-OTP
- EAP-SIM
- LEAP
- PEAP (MSCHAPv2)
- EAP-TLS
- EAP-TTLS (CHAP, MSCHAP, MSCHAPv2, PAP)

Future versions of Xsupplicant may include support for:

- EAP-FAST
- PEAP-GTC

See Chapter 5: EAP Options for details on configuring a specific EAP type with Xsupplicant.

2.3. Authentication Server Compatibility Matrix

The following is a compatibility matrix that lists tested EAP/Authentication Server combinations. Servers listed with a "+" mean that a particular EAP type is supported by that server. A Server/EAP combination listed with a "*" indicates xsupplicant compatibility. A Server/EAP combination listed with a "!" indicates an incompatibility with Xsupplicant.

Lack of a "*" in an entry does not indicate that a particular combination will not work, only that it has not been tested. This list is not meant to be a complete list of tested combinations, as we do not have the resources to keep the list up to date.

Table 2-1. Supported Authentication Servers

	Cisco ACS	FreeRADIUS	Funk SBR	Infoblox	Meetinghouse AEGIS	Microsoft IAS	Radiator	Roving Planet CSD
EAP-AKA	-	-	-	-	-	-	*	-
EAP-FAST	+	-	-	-	-	-	-	-
EAP-GTC	-	-	-	-	-	-	*	-
EAP-MD5	*	*	*	*	*	-	*	*
EAP-OTP	-	-	-	-	-	-	*	-
EAP-SIM	-	*	-	-	-	-	*	-
EAP-TLS	+	+	+	+	+	+	*	+
LEAP	+	+	+	-	+	-	*	+
PEAP-GTC	+	-	-	-	+	-	+	-
PEAP-MSCHAPv2	*	*	*	!	*	*	*	+
TTLS-CHAP	-	*	+	+	+	-	*	+
TTLS-MSCHAP	-	*	+	+	+	-	*	+
TTLS-MSCHAPv2	-	+	+	+	+	-	*	+

	Cisco ACS	FreeRADIUS	Funk SBR	Infoblox	Meetinghouse AEGIS	Microsoft IAS	Radiator	Roving Planet CSD
TTLS-PAP	-	+	*	!	*	-	*	+

2.4. Supplicants

The goal of this section is to help inform the reader of general 802.1X compatibility. The following list is probably not complete, but should provide a fairly comprehensive supplicant/operating system list.

Table 2-2. Supplicant Support Matrix

	*BSD	Linux	Mac OS X 10.2.x	Mac OS X 10.3.x	Pocket PC 2002	Pocket PC 2003	Windows 98	Windows Me	Windows 2k	Windows XP
Alfa+Ariss SecureW2 (http://www.securew2.com/)	-	-	-	-	-	+	-	-	+	+
Funk Odyssey (http://www.funk.com/)	-	-	-	-	+	+	+	+	+	+
Meetinghouse Aegis (http://www.mtghouse.com/)	-	+	+	+	+	+	+	+	+	+
Native	-	-	-	+	-	+	-	-	-	+
Open1x (http://www.open1x.org)	-	+	-	-	-	-	-	-	-	-
wEAP (http://weap.sourceforge.net/)	-	-	-	-	-	-	-	-	+	+
Wire1x (http://wire.cs.nthu.edu.tw/wire1x)	-	-	-	-	-	-	+	+	+	+

Chapter 3. Getting The Software

3.1. Download Stable Releases

Users should normally download a stable release from http://sourceforge.net/project/showfiles.php?group_id=60236

3.2. Pre-Packaged Releases for Some Distibutions

At some point we plan on releasing pre-packaged versions of xsupplicant for distributions such as Debian, RedHat, SuSe, Slackware, etc... If you would like to see your favorite distribution supported, please let us know (Or better yet, send us the package)!

3.3. CVS

We do not recommend CVS downloads for people unfamiliar with it. CVS often contains new features that have not yet made it into a stable release, but it can also be extremely unstable. We also do not have the time to explain how CVS works (there is good documentation on the web regarding CVS and its many features/options). We encourage users to submit patches against CVS, but patches may not necessarily be commited immediately.

To download xsupplicant via CVS you can use the following command:

```
cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/open1x co xsupplicant
```

Chapter 4. Installation

4.1. Prerequisites

You will need to have the following installed on your system before installing xsupplicant:

OpenSSL - http://www.openssl.org/

libpcsclite (For EAP-SIM/EAP-AKA support) - http://www.linuxnet.com/

Pthreads Support (For EAP-SIM/EAP-AKA support)

4.2. Quick Start Guide

Generally speaking you should be able to type: **`./configure; make; make install`** to configure and install the Open1x supplicant.

If you run into problems, please read the rest of this chapter.

4.3. Running the Configure Script

The following configuration options are available for xsupplicant:

`--enable-eap-sim` - Enables EAP SIM/AKA authentication. (Requires libpcsclite)

`--enable-experimental` - Enable the use of experimental features/code.

`--with-ossl-root` - Specify the path to the directory holding the OpenSSL source tree.

`--enable-radiator-test` - Enable use of the AKA test vectors from Radiator. (Doesn't require a SIM card).

`--enable-static-openssl` - Statically link OpenSSL into the xsupplicant binary.

4.4. When Things go Wrong.

If you experience a problem building xsupplicant, you should:

- Check config.log for any interesting error messages.
- Double check the xsupplicant prerequisites.
- E-mail the xsupplicant mailing list with a detailed explanation of the problem you are experiencing, including any error messages you got and please include a copy of your configuration file and any command line options you used to run xsupplicant. The more information you provide to us, the easier it will be for us to help you fix your problem.

Chapter 5. Configuration

5.1. Overview

5.2. Commandline Options

`-W`

Allow Xsupplicant to work with wpa_supplicant.

`-c /path/to/config_file`

You can specify a configuration file to be used with the "-c" option. Xsupplicant will automatically look for and use "/etc/xsupplicant.conf" if it exists.

`-i device`

Provide the interface on which to listen for EAPoL packets. Please note that xsupplicant will currently look for any valid interfaces and fork to handle each valid interface it finds. You do not need to run multiple instances of xsupplicant yourself.

`-d debug_level`

<debug_level> can be any of:

- 0 - 7 Old style debug flags.
- A - Enable ALL debug flags.
- c - Enable CONFIG debug flag.
- s - Enable STATE debug flag.
- a - Enable AUTHTYPES debug flag.
- i - Enable INT debug flag.
- n - Enable SNMP debug flag.
- e - Enable EVERYTHING debug flag.
- x - Enable EXCESSIVE debug flag.

Debug flags can be stacked such as: **`xsupplicant -d csai`**. This would provide CONFIG, STATE, AUTHTYPES, and INT debug output.

5.3. System Wide Config File

Xsupplicant will look for a file in /etc/xsupplicant.conf by default, unless the -c option is used at runtime. The xsupplicant configuration file consists of several global variables and a set of network-specific configurations. Each network

profile definition contains one or more EAP sections, which contain specific configuration information for the network in question.

5.4. Global Configuration Options

The following is a complete list of the global options available in xsupplicant 1.0.1. These options should be defined outside of any xsupplicant network profiles contained in your configuration file.

5.4.1. Global Options

The following list is a complete list of the global configuration options for the xsupplicant configuration file.

Xsupplicant Config - Global Options

`allow_interfaces`

Defining an interface in "allow_interfaces" will bypass the rules that xsupplicant uses to determine if an interface is valid. For most people this setting shouldn't be needed. It is useful for having xsupplicant attempt to authenticate on interfaces that don't appear to be true physical interfaces. (i.e. Virtual interfaces such as eth0:1)

Example 5-1. Global Option "allow_interface"

```
allow_interfaces = eth0:1, eth0:2
```

`default_netname`

If this option is not used, xsupplicant will attempt to read the profile "default" from the configuration file. Some users may actually have a network named "default", so this option can be used to redefine which profile is used as the default profile.

Example 5-2. Global Option "default_netname"

```
default_netname = default
```

`deny_interfaces`

Defining an interface in "deny_interfaces" will prevent xsupplicant from attempting to authenticate on a given interface. This is useful if you know that you will never do 802.1X on a specific interface, such as a VMWare virtual interface. However, allows will take priority over denies, so defining the same interface in the allow_interfaces, and deny_interfaces will result in the interface being used.

Example 5-3. Global Option "deny_interfaces"

```
deny_interfaces = eth1, vmnet0
```

`first_auth_command`

This directive instructions xsupplicant to run the following command when xsupplicant authenticates to a wireless network for the first time. This will usually be used to start a DHCP client process.

Example 5-4. Global Option "first_auth_command"

```
first_auth_command = <BEGIN_COMMAND>dhclient %i<END_COMMAND>
```

`logfile`

When running in daemon, or non-foreground mode, you may want to have the output of the program. So, define a log file here. Each time XSupplicant is started, this file will be replaced. So, there is no need to roll the log file.

Example 5-5. Global Option "logfile"

```
logfile = /var/log/xsupplicant.log
```

`network_list`

This directive defines all of the network profiles which should be kept in memory and used.Comma delimited list or "all" for keeping all defined configurations in memory. For efficiency, keep only the networks you might roam to in memory. To avoid errors, make sure your default network is always in the network_list. In general, you will want to leave this set to "all".

Example 5-6. Global Option "network_list"

```
network_list = all
```

`reauth_command`

This command is executed when xsupplicant reauthenticates to a wireless network. This may be used to have the dhcp client rerequest it's IP address. This option will probably not be useful except in situations where the layer 3 network might change on a wireless network where the SSID remains the same, such as in the case of the University of Utah's wireless network.

Example 5-7. Global Option "reauth_command"

```
reauth_command = <BEGIN_COMMAND>echo "authenticated user %i"<END_COMMAND>
```

`startup_command`

This command is executed when xsupplicant is first started. This command can do things such as configure the card to associate with the network properly.

Example 5-8. Global Option "startup_command"

```
startup_command = <BEGIN_COMMAND>echo "some command"<END_COMMAND>
```

5.4.2. State Machine Variables

- auth_period
- max_starts
- held_period

The auth_period, held_period, and max_starts modify the timers in the state machine. (Please reference the 802.1X spec for info on how they are used.) For most people, there is no reason to define these values, as the defaults should work.

5.5. Network Profile Configuration Options

A network profile consists of a declaration, such as "default". A complete profile definition is defined by a beginning and ending brace, "{" and "}" respectively. The profile defines network specific attributes for 802.1X operation, such as allowed EAP-types, the username for a given EAP-type, and any EAP-type specific options.

Each network profile section may include one or more valid EAP-types. When the Authentication Server requests a given EAP type, xsupplicant will use that EAP type if a valid configuration exists. Otherwise, xsupplicant will NAK the Authentication Server, and request the first EAP type defined by the "allow_types" configuration option. If xsupplicant and the Authentication server cannot agree on an EAP type, the authentication will not take place, so be sure to define an appropriate EAP-type for your network!

Xsupplicant Config - Network Profile Options

`allow_types`

This option describes which EAP types this network will allow. The first type listed will be requested if the server tries to use something not in this list. Individual EAP types can be specified, or the keyword "all" can be used to specify all EAP types. Additionally, it is legal to use either an underscore (_) or dash (-) for the separator character between "eap" and the type. For instance, "eap-tls" and "eap_tls" are both valid.

Example 5-9. Profile Option "allow_types"

```
allow_types = eap-tls, eap-md5, eap-gtc, eap-otp

allow_types = all
```

`dest_mac`

This option forces xsupplicant to send its packets to this destination MAC address. In most cases, this isn't needed, and shouldn't be defined.

Example 5-10. Profile Option "dest_mac"

```
dest_mac = 00:aA:bB:cC:dD:eE
```

`identity`

This defines the EAP Response Identity, also known as the "outer identity" , or, what xsupplicant will respond with when presented with an EAP Identity Request. This is typically the username for this network.

Example 5-11. Profile Option "identity"

```
identity = <BEGIN_ID>myid@mynet.net<END_ID>
```

`type`

Xsupplicant will attempt to determine if a given interface is wired or wireless, but some drivers misbehave. This option forces xsupplicant to recognize interfaces in a certain way. Use this option if your interface is detected incorrectly by xsupplicant. Valid options are "wired" and "wireless".

Example 5-12. Profile Option "type"

```
type = wireless
```

`wireless_control`

If the profile is forced to wired, this will not do anything. However, if the interface is forced, or detected to be wireless XSupplicant will take control of re/setting WEP keys when the machine first starts, and when it jumps to a different AP. In general, you won't need to define, or set this value. Valid options are "yes" and "no".

Example 5-13. Profile Option "wireless_control"

```
wireless_control = yes
```

5.6. EAP Options

Each network profile in the xsupplicant configuration file may have one or more EAP sections defined. Each EAP section must be associated to a network profile. Each EAP section may also have one or more subsections associated with it.

For instance, a configuration for EAP-TTLS may have any of CHAP, MSCHAP, MSCHAPv2, or PAP subsections defined. Some EAP types do not contain any subsections.

5.6.1. Reused EAP Options

The following options are re-used in many of the EAP types listed below.

Common EAP Options

`chunk_size`

The chunk_size directive specifies the maximum size that a certificate chunk can be. Use this option in EAP

types that use either one or both of client or server certificates (TLS, PEAP, TTLS).

Example 5-14. Common EAP Option "chunk_size"

```
chunk_size = 1398
```

`cncheck`

The cncheck directive provides the ability to verify the CN field of an authentication server certificate for EAP types that use server-side certificates (TLS, TTLS, PEAP).

Use this directive in conjunction with cnexact to control how granular the server certificate check should be.

Example 5-15. Common EAP Option cncheck

```
cncheck = someradius.mynet.net
```

`cnexact`

The cnexact directive forces a failure on authentication if the CN field of the server's certificate does not exactly match the cncheck option in the specified Network/EAP configuration. Set this to "no" to only match the end of the string, which is useful in a situation where there might be mulitple authentication servers for your organization.

For example, a "cncheck = utah.edu" with a "cnexact = no" would match on "foo.utah.edu" and "bar.utah.edu", which might be separate servers on a campus utilizing 802.1X.

Example 5-16. Common EAP Option "cnexact"

```
cnexact = yes
```

`crl_dir`

The crl_dir option is used to specify a directory containing certificate revocation lists. This option can be used in EAP types that use either one or both of client or server certificates (TLS, PEAP, TTLS).

Example 5-17. Common EAP Option "crl_dir"

```
crl_dir = /home/user/certificates/revoked
```

`password`

The password directive is used in EAP types that require a password for authentication.

Example 5-18. Common EAP Option "password"

```
password = <BEGIN_PASS>password<END_PASS>
```

`random_file`

This option is used to specify the random file used to grab random data for certificate something-or-other. Use this option in EAP types that use either one or both of client or server certificates (TLS, PEAP, TTLS).

This option is typically /dev/urandom, but may be different depending on the operating system you are using. You should probably not use /dev/random, since it blocks and can slow authentication down.

Example 5-19. Common EAP Option "random_file"

```
random_file = /dev/urandom
```

`root_cert`

The root_cert option is used to specify the path to the CA public certificate which signed one or both of your server and client certificates. The root_cert option is used in EAP types that use either one or both of client or server certificates (TLS, PEAP, TTLS).

This certificate is used to verify, on the client side, that the server's certificate was signed by the appropriate certificate authority, and on the server side, to verify that the user certificate was signed by the proper certificate authority. This certificate should be the same for both client and server, since it is simply the public key for the certificate authority that signed the client and server certificates.

You can specify a value of "NONE" to prevent xsupplicant from verifying the server certificate, but this is *HIGHLY* frowned upon. If you use this option, you are opening yourself up to a very easy to execute man-in-the-middle attack that could compromise your username and password. Consider yourself warned. :-)

Example 5-20. Common EAP Option "root_cert"

```
root_cert = /home/user/certificates/root_cert.pem
```

`root_dir`

The root_dir option is used to specify a path to a directory containing root certificates. This can be used to force Xsupplicant to allow any combination of root certificates in a given directory to help simplify configuration. This option can be used instead of the root_cert directive in EAP types that use either one or both of client or server certificates (TLS, PEAP, TTLS).

Example 5-21. Common EAP Option "root_dir"

```
root_dir = /home/user/certificates
```

`session_resume`

The session_resume directive is used to specify whether or not to attempt to initiate "TLS Session Resumption" (Also called "Fast Reconnect") when re-authenticating with a server.

Example 5-22. Common EAP Option "session_resume"

```
session_resume = yes
```

`username`

The username option is used in EAP types that require a username for authentication.

Example 5-23. Common EAP Option "username"

```
username = <BEGIN_UNAME>myid@mynet.net<END_UNAME>
```

`user_cert`

This option, which is required for TLS, specifies the path to the user certificate used for TLS authentication. A user certificate in TLS is similar to a username in password-based authentication mechanisms.

User certificates can also be used with PEAP and TTLS, but are not required, and most people will not need this functionality.

Example 5-24. Common EAP Option "user_cert"

```
user_cert = /home/user/certificates/user-cert.pem
```

`user_key`

This option is the key for the user_cert file.

As with user_cert, this option is required for TLS and can be used with TTLS or PEAP if using a user certificate for authentication.

Example 5-25. Common EAP Option "user_key"

```
user_key = /home/user/certificates/user-key.pem
```

`user_key_pass`

This is the password for the user_key.

Example 5-26. Common EAP Option "user_key_pass"

```
user_key_pass = <BEGIN_PASS>password for user-key.pem<END_PASS>
```

5.6.2. EAP-AKA

EAP-AKA allows the following options: username, password, auto_realm.

Example 5-27. Example EAP-AKA Configuration

```
logfile = /var/log/xsupplicant.log

startup_command = <BEGIN_COMMAND>dhclient %i<END_COMMAND>

default {
  allow_types = eap-aka

  identity = <BEGIN_ID>myid@mynet.net<END_ID>

  eap-aka {
      username = <BEGIN_UNAME>akauser<END_UNAME>
      password = <BEGIN_PASS>akauserpass!<END_PASS>
      auto_realm = yes
  }
}
```

5.6.3. EAP-GTC

Example 5-28. Example EAP-GTC Configuration

```
logfile = /var/log/xsupplicant.log

startup_command = <BEGIN_COMMAND>dhclient %i<END_COMMAND>

default {
  allow_types = eap-gtc

  identity = <BEGIN_ID>myid@mynet.net<END_ID>

  eap-gtc {

  }
```

```
}
```

5.6.4. EAP-MD5

EAP-MD5 allows the following option(s): password.

Example 5-29. Example EAP-MD5 Configuration

```
logfile = /var/log/xsupplicant.log

startup_command = <BEGIN_COMMAND>dhclient %i<END_COMMAND>

default {
  allow_types = eap-md5

  identity = <BEGIN_ID>myid@mynet.net<END_ID>

  eap-md5 {
      password = <BEGIN_PASS>password<END_PASS>
  }
}
```

5.6.5. EAP-MSCHAPv2

Valid eap-mschapv2 options are: username (only needed when using mshcapv2 as a phase 2 type), password.

eap-mschapv2 can also be defined as a sub-option inside of a PEAP profile. See the PEAP section for an example.

Example 5-30. Example EAP-MSCHAPv2 Configuration

```
logfile=/var/log/xsupplicant.log

startup_command = <BEGIN_COMMAND>dhclient %i<END_COMMAND>

default {
  allow_types = eap-mschapv2

  identity = <BEGIN_ID>myid@mynet.net<END_ID>

  eap-mschapv2 {
      password = <BEGIN_PASS>password<END_PASS>
  }
}
```

5.6.6. EAP-OTP

Example 5-31. Example EAP-OTP Configuration

```
logfile = /var/log/xsupplicant.log

startup_command = <BEGIN_COMMAND>dhclient %i<END_COMMAND>

default {
  allow_types = eap-otp

  identity = <BEGIN_ID>myid@mynet.net<END_ID>

  eap-otp {

  }
}
```

5.6.7. EAP-SIM

EAP-SIM allows the following options: username, password, auto_realm.

Example 5-32. Example EAP-SIM Configuration

```
logfile = /var/log/xsupplicant.log

startup_command = <BEGIN_COMMAND>dhclient %i<END_COMMAND>

default {
  allow_types = eap-sim

  identity = <BEGIN_ID>myid@mynet.net<END_ID>

  eap-sim {
      username = <BEGIN_UNAME>simuser<END_UNAME>
      password = <BEGIN_PASS>simuserpass!<END_PASS>
      auto_realm = yes
  }
}
```

5.6.7.1. EAP-SIM Specific Options

The following list defines options specific to "eap-sim":

`auto_realm`

The auto_realm option determines whether or not your realm will be automatically appended to your username on authentication, or whether the user will do this manually in the xsupplicant configuration. This option is fairly dependent on how your service is set up, so check with your provider to see if this option should be enabled.

Valid auto_realm options are: yes, no.

Example 5-33. EAP-SIM Option "auto_realm"

```
auto_realm = yes
```

5.6.8. PEAP

Valid options for PEAP are: chunk_size, cncheck, cnexact. crl_dir, random_file, root_cert, root_dir, session_resume, user_cert, user_key, user_key_pass,

PEAP currently requires eap-mschapv2 as a sub-option. Future versions of xsupplicant may include support for other embedded EAP-types such as eap-gtc. In addition, the "username" directive is required for inner-eap types used with PEAP.

Example 5-34. Example PEAP Configuration

```
logfile = /var/log/xsupplicant.log

startup_command = <BEGIN_COMMAND>dhclient %i<END_CLIENT>

default {

  allow_types = all

  identity = <BEGIN_ID>myid@mynet.net<END_ID>

  eap-peap {
      root_cert = /home/user/certificates/root.pem
      chunk_size = 1398
      random_file = /dev/urandom
      cncheck = radiusserver.mynet.net
      cnexact = yes
      session_resume = no

      eap-mschapv2 {
        username = <BEGIN_UNAME>myid@mynet.net<END_UNAME>
        password = <BEGIN_PASS>password<END_PASS>
      }
```

```
  }
}
```

5.6.9. EAP-TLS

Valid options for EAP-TLS are: chunk_size, crl_dir, random_file, root_cert, root_dir, session_resume, user_cert, user_key, user_key_pass,

Example 5-35. Example EAP-TLS Configuration

```
logfile = /var/log/xsupplicant.log

startup_command = <BEGIN_COMMAND>dhclient %i<END_CLIENT>

default {

  allow_types = all

  identity = <BEGIN_ID>myid@mynet.net<END_ID>

  eap_tls {
     user_cert = /home/user/certificates/user-cert.pem
     user_key  = /home/user/certificates/user-key.pem
     user_key_pass = <BEGIN_PASS>password for user-key.pem<END_PASS>
     root_cert = /home/user/certificates/root.pem
     crl_dir = /home/user/certificates/revoked


     chunk_size = 1398
     random_file = /dev/urandom
     session_resume = no
  }
}
```

5.6.10. EAP-TTLS

Valid options for EAP-TTLS are: chunk_size, cncheck, cnexact. crl_dir, random_file, root_cert, root_dir, session_resume, user_cert, user_key, user_key_pass, phase2_type

EAP-TTLS may also have one or more sub-options: chap, mschap, mschapv2, pap.

Example 5-36. Example EAP-TTLS Configuration

```
logfile = /var/log/xsupplicant.log

startup_command = <BEGIN_COMMAND>dhclient %i<END_COMMAND>

default {
  allow_types = eap-ttls

  identity = <BEGIN_ID>myid@mynet.net<END_ID>

  eap-ttls {
      root_cert = /home/user/certificates/root.pem
      chunk_size = 1398
      random_file = /dev/urandom
      cncheck = myradius.radius.com
      cnexact = no
      session_resume = no
      phase2_type = pap

      pap {
        username = <BEGIN_UNAME>myid@mynet.net<END_UNAME>
        password = <BEGIN_PASS>password<END_PASS>
      }
  }
}
```

5.6.10.1. EAP-TTLS Specific Option(s)

The following list defines options specifc to "eap-ttls":

`phase2_type`

The phase2_type directive specifies which phase2 type to use when authenticating with TTLS.

Valid phase 2 types are: chap, mschap, mschapv2, pap.

Example 5-37. EAP-TTLS Option "phase2_type"

```
phase2_type = pap
```

`chap`

Use this option in TTLS to specify a username and password for a CHAP authentication.

Most people will probably want to use PAP with TTLS, however.

Valid chap options are: username, password,

Example 5-38. EAP-TTLS Option "chap"

```
chap {
  username = <BEGIN_UNAME>myid@mynet.net<END_UNAME>
  password = <BEGIN_PASS>password<END_PASS>
}
```

`mschap`

Use this option in TTLS to specify a username and password for an MSCHAP authentication.

Most people will probably want to use PAP with TTLS, however.

Valid mschap options are: username, password,

Example 5-39. EAP-TTLS Option "mschap"

```
mschap {
  username = <BEGIN_UNAME>myid@mynet.net<END_UNAME>
  password = <BEGIN_PASS>password<END_PASS>
}
```

`mschapv2`

Use this option in TTLS to specify a username and password for an MSCHAPv2 authentication. This option is different than eap-mschapv2.

Most people will probably want to use PAP with TTLS, however.

Valid mschapv2 options are: username, password,

Example 5-40. EAP-TTLS Option "mschapv2"

```
mschapv2 {
  username = <BEGIN_UNAME>myid@mynet.net<END_UNAME>
  password = <BEGIN_PASS>password<END_PASS>
}
```

`pap`

Use this option in TTLS to specify a username and password for a PAP authentication.

Most people will probably want to use this option when authenticating with TTLS.

Valid pap options are: username, password,

Example 5-41. EAP-TTLS Option "pap"

```
pap {
  username = <BEGIN_UNAME>myid@mynet.net<END_UNAME>
  password = <BEGIN_PASS>password<END_PASS>
}
```

5.6.11. LEAP

Valid options for LEAP are: password

Example 5-42. Example LEAP Configuration

```
logfile = /var/log/xsupplicant.log

startup_command = <BEGIN_COMMAND>dhclient %i<END_COMMAND>

default {
  allow_types = leap

  identity = <BEGIN_ID>myid@mynet.net<END_ID>

  leap {
      password = <BEGIN_PASS>password<END_PASS>
  }
}
```

5.7. User Config File

We hope to provide the ability to specify both a global configuration file and a more specific user configuration file capability in a future release of xsupplicant.

5.8. Using a GUI for Configuration

The current version of Xsupplicant does not provide a mechanism to configure itself from a Graphical User Interface, except for providing a password. We hope to provide such tools in the future. Fortunately, the 1.0 config file format is much easier to read than older versions.

If you have the QT development tools installed, you can compile and use the qt-gremlin, or xsup_monitor programs, to provided with Xsupplicant for real-time password prompting in X11. We hope to extend this tool so it can also display EAP-Notifications as well.

Chapter 6. Advanced Usage

This chapter is intended to provide examples for making xsupplicant easier to use.

Chapter 7. Troubleshooting

7.1. A Guide to Troubleshooting

If you experience any problems with xsupplicant, please use the following guide to troubleshoot your issues:

- Send us a debug output of xsupplicant, and any relevant xsupplicant options.
- Tell us what card driver you are using, including revision (which can usually be found with dmesg)!
- Send us a gdb backtrace, if possible.
- Send us a copy of your configuration file.
- Please make sure you *DO NOT* include passwords in your configuration file, or in your -d 7 output.

Run xsupplicant in debug mode by using the "-d 7" and "-f" switches and gather the output. If you are segfaulting, run xsupplicant in gdb, if possible and provide a backtrace:

Example 7-1. Getting a GDB Backtrace

```
gdb xsupplicant
(gdb) set args (any args you normally use)
(gdb) run
(gdb) backtrace (after segfault)
```

This will help us find the problem easier.

Send us a copy of your configuration (but remove the passwords please).

7.2. Known Problems

The following is a summary of the known issues with this version of xsupplicant.

- The supplicant may get confused on wired ports that are set up to allow more than one client per port.
- Cisco 340/350 cards do not work correctly with Xsupplicant 1.0. This appears to be due to the driver (or firmware?) hijacking the 0x888e frames, which prevents 802.1X authentication from being possible.

  UPDATE: We have been successful getting a Cisco 350 card to work with the built-in Linux 2.6.7 Aironet driver to the extent that it will authenticate. The driver still has the (common) rekeying bug.

  This means that it should be fairly easy to fix the Kernel driver. We are still unsure what caused this.

Appendix A. Setup for Authenticators and RADIUS Servers

A.1. Authenticators

A.1.1. Open Source Authenticator Projects

- HostAP (http://hostap.epitest.fi/)
- Rose (http://www.rosewlan.com/)

A.1.2. Commercial Authenticators

-

not a complete setup guide for all APs on the market, instead some comments on how to setup the more common APs to work with open1x. also provide pointers to more complete setup & config guides

A.2. Authentication Servers

A.2.1. Open Source Authentication Servers

See the Authentication Server Compatibility Matrix for a complete list of supported EAP types.

- FreeRADIUS

A.2.2. Commercial Authentication Servers

See the Authentication Server Compatibility Matrix for a complete list of supported EAP types.

- Cisco ACS
- Funk SBR
- Infoblox

- Meetinghouse AEGIS
- Microsoft IAS
- Radiator (Source Code Included with Purchase)
- Roving Planet CSD (Based on FreeRADIUS)

Appendix B. Links to Related Resources

B.1. Companies that Support Open1x

The following entities have donated hardware/software to the Open1x project:

Contributions are listed in the order they were received.

The companies listed below do not endorse Open1x.

- Proactive Network Management & Proxim
  - ORiNOCO AP-600 (802.11b/g)
  - http://www.pnmc.com/
  - http://www.proxim.com/
- Radiator
  - Many bug fixes, and added features in a very timely manner.
  - http://open.com.au/
- Hewlett-Packard
  - HP Procurve 420 AP (802.11b/g)
  - http://www.hp.com/
- 3Com
  - 3Com 8200 AP & 802.11a/b/g Wireless Card (3CRPAG175)
  - http://www.3com.com/
- University of Utah Center for High Performance Computing
  - Funding for the Networld + Interop HotStage
  - http://www.chpc.utah.edu/
- Cisco Systems
  - Cisco 1200 AP (802.11b/g)
  - http://www.cisco.com/

B.2. 802.1X Related Open Source Projects

Wire1x (http://wire.cs.nthu.edu.tw/wire1x/) - An Open Source xsupplicant port for Windows.

wEAP (http://weap.sourceforge.net/) - An Open Source project for Windows EAP Plugins.

B.3. 802.1X related proprietary projects

B.4. Standards

The 802.11 Specification (http://grouper.ieee.org/groups/802/11/) [ieee.org]

Wireless Ethernet (http://www.wirelessethernet.com/) [wirelessethernet.com]

IEEE Working Groups (http://grouper.ieee.org/groups/802/dots.html) [ieee.org]

IEEE 802.11b Specification (http://grouper.ieee.org/groups/802/11/index.html) [ieee.org]

IEEE 802.1X (http://www.ieee802.org/1/pages/802.1x.html) [ieee.org]

RADIUS with IEEE 802.1X (http://www.open1x.org/papers/draft-congdon-radius-8021x-10.txt) [local]

Wireless LAN Resources for Linux (http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/) [hpl.hp.com]

B.5. Other Resources

B.5.1. Howtos

Sniffing a wireless network with a Cisco WLAN card. (http://www.cs.umd.edu/~npetroni/airo.html) [umd.edu]

Setting up 802.1X using a WinXP client and a Win2K Radius server. (http://www.cs.umd.edu/~mvanopst/8021x/howto/) [umd.edu]

Setting up 802.1X using Xsupplicant and FreeRADIUS (http://www.missl.cs.umd.edu/wireless/eaptls/) [umd.edu]

Using the Orinoco (Hermes) card with Xsupplicant. (http://www.oreillynet.com/pub/wlg/4602) [oreillynet.com]

B.5.2. Related Links

Ethereal Patches for 802.1X Decoding (http://www.missl.cs.umd.edu/wireless/ethereal/) [umd.edu]

Support for 802.1X in WinXP (http://www.microsoft.com/presspass/press/2001/Mar01/03-26XPWirelessPR.asp) [microsoft.com]

The University of Utah 802.1X Wireless Website (http://wireless.utah.edu/) [utah.edu]