Chapter 6 - EAP Authentication

This chapter describes using Extensible Authentication Protocol with FreeRADIUS. The following topics are discussed in this chapter:

- EAP Overview
- Types/Methods
- Testing with eapol_test
- TLS based EAP methods
- Certificates

6.0 EAP Overview

EAP stands for Extensible Authentication Protocol. The extensible portion of the EAP acronym describes the intended use of the protocol: EAP is a simple wrapper that can transport other authentication methods. These other methods are independent of the encapsulating EAP layer and are independent of each other.

EAP provides only for authentication. There is no way to transport authorization data in EAP (as opposed to RADIUS, which can send authorization data); therefore, only success and failure can be transported in EAP - other information such as IP address assignments and session timeouts cannot be transported in this manner.

The protocol originally started out in PPP, where it was intended to replace the use of PAP and CHAP. Since then, it has gained wider use in wired and wireless network authentication, especially with 802.1X

When used in conjunction with RADIUS, EAP requires the following participants (the names used here are taken from the IEEE 802.1X standard):

- **Supplicant**
  The supplicant is the machine that requests network access to a Local Area Network (LAN). It contains the credentials used for authentication, and it is one end of the EAP exchange.
- **Authenticator**
  The authenticator is the Access Point or switch that controls access to the LAN. It allows or denies network access based on the status returned by EAP. However, it does not directly authenticate the user itself. Instead, it hands the authentication over to the RADIUS server by encapsulating EAP inside of RADIUS.
  As it may be confusing to call something an “authenticator” when it simply relays packets and does not do any authentication, few administrators use the term “authenticator”. The terms “switch” or “access point” are more common. In this document the term NAS is used, which is also the RADIUS term for “switch” or “Access Point”.
- **RADIUS Server**
  The RADIUS server receives the Access-Request from the NAS and decodes the EAP data. It performs authentication and returns an EAP Success or Fail message, which is encapsulated in a RADIUS packet. In theory, the system performing the EAP authentication is an EAP Server.

Since the supplicant is connected to the NAS via LAN, the Ethernet must be used to communicate between the two. Communication is accomplished using an Ethernet type assigned for EAP, called EAP over LAN or EAPoL. Similarly, the NAS uses RADIUS to communicate with the RADIUS server.

The EAP system has become increasingly complex, with ten layers of protocols between the user name and password and the Ethernet IP. By the time the data arrives at the RADIUS server, the user name and

password is carried over the following protocols: Ethernet, IP, UDP, RADIUS, RADIUS Attributes, EAP, EAP-TTLS, TLS, Diameter Attributes, and finally the user name and password. In addition, the NAS performs one kind of packet retransmission using EAPoL to the supplicant and a different retransmission method using RADIUS to the RADIUS server. With this level of complexity, it's a bit of a surprise it works at all.

6.1 Types/Methods

The EAP methods are presented, below, in order of increasing complexity rather than in alphabetical order.

- **EAP-GTC**
  The Generic Token Card (GTC) method provides a challenge-response mechanism that can be used for token cards. It is analogous to the PAP authentication method. It is not secure, and it should be used only for wired 802.1X.
  It cannot be used in wireless settings because it does not provide for the encryption keys required by wireless 802.1X.
- **EAP-MD5**
  The EAP-MD5 method is CHAP wrapped inside of EAP. The same security issues as found with EAP-GTC apply here.
- **EAP-MSCHAPv2**
  The EAP-MSCHAPv2 method is MS-CHAPv2 wrapped in EAP. The same security issues as found with EAP-GTC apply here.
- **LEAP**
  The Lightweight EAP (LEAP) method is based on MS-CHAPv2. The same security issues as found with EAP-GTC apply here. However, some access points allow LEAP for wireless authentication. LEAP is vulnerable to off-line dictionary attacks. For this reason, it is not recommended to use LEAP.
- **EAP-IKEv2**
  The Internet Key Exchange v2 (IKEv2) protocol is used to perform authentication and key exchange for IPSec systems. Because if has not been widely deployed, its sample configuration is in `experimental.conf`.
- **EAP-TLS**
  The Transport Layer Security (TLS) method provides a TLS (or SSL) tunnel between the supplicant and RADIUS server. It requires multiple certificates to operate. A root certificate is used to create and sign both a server certificate and a client certificate. The server certificate is used by the RADIUS server, and the client certificate is used by the supplicant. Both have copies of the root certificate. When the supplicant attempts to authenticate, the server sends its certificate to the supplicant. The supplicant then verifies the server certificate against the root certificate. If the server certificate passes verification, the supplicant then supplies its certificate to the server, and the server verifies the client certificate against the root certificate. If that verification succeeds, the server returns an authentication success.
- **EAP-TTLS**
  The Tunneled TLS (TTLS) method is similar to EAP-TLS, in that it provides a TLS tunnel between the supplicant and the server. However, in this case, client certificates are not necessary. Instead, the TLS tunnel is used to transport additional authentication data such as a user name and password, a CHAP password, or an MS-CHAP authentication sequence.
- **PEAP**
  The Protected EAP (PEAP) method is broadly similar to EAP-TTLS. It was created by Microsoft for use with its Windows operating system. While there are differences between the two methods, most of those differences are relevant only to developers writing EAP methods. The difference that matters

to system administrators is that the authentication method carried inside of the TLS tunnel in PEAP is identical to MS-CHAPv2.

- **EAP-SIM**
  The Subscriber Identity Module (SIM) method is used to provide authentication for Global System for Mobile Communications (GSM) systems. The credentials are stored on a SIM card, such as used in 3G telephones.
  While the server supports EAP-SIM, it has not been widely deployed. Using EAP-SIM also requires access to a Signaling System #7 (SS7) gateway. As a result, EAP-SIM is not discussed any further in this book.
- **EAP-FAST**
  The Flexible Authentication via Secure Tunneling (FAST) method was developed by Cisco and is not widely used outside of Cisco environments. The server does not support EAP-FAST, though that may change in the future. There is experimental support for EAP-FAST in the `eap2` module. The module is documented in the `experimental.conf` file and is not discussed any further in this book.
- **Other EAP Methods**
  A large number of additional EAP methods have been defined. Most of them are not widely used. As Windows supports only PEAP, there are few reasons for a RADIUS server to support anything else. The `eap2` module discussed above contains experimental support for a large variety of EAP methods. It does this by leveraging the HostAP `libeap` library, which implements pretty much every EAP method that has been defined. That library is an impressive piece of work, but it has not been well integrated into FreeRADIUS. Perhaps a future release of the server will change t o accommodate more EAP methods.

6.2 Testing With eapol_test

The server does not supply client test tools that support complex EAP authentication methods. Network RADIUS SARL recommends `eapol_test`as a client test tool. It is part of the `wpa_supplicant` program. It supports all commonly used EAP types and the majority of the rare types. It has excellent debugging output, which is invaluable for tracking down EAP problems.

6.2.1 Downloading and Installing eapol_test

The default installation of `wpa_supplicant` does not include the `eapol_test` program, so it will need to be built from source. Below are instructions that describe how to build the `eapol_test` program:

1. To use `wget` to build `eapol_test`, enter the following:

```
$ wget http://hostap.epitest.fi/releases/wpa_supplicant-0.7.3.tar.gz
```

If you do not have `wget`*, you can download the program manually from* **http://hostap.epitest.fi/wpa_supplicant/**

2. Extract the archive, and set up the build configuration file:

```
$ tar -zxf wpa_supplicant-0.7.3.tar.gz
$ cd wpa_supplicant-0.7.3
$ cp defconfig .config
$ vi .config
```

3. To enable eapol_test, find the line containing:

```
#CONFIG_EAPOL_TEST=y

Change it to:

CONFIG_EAPOL_TEST=y

Save the file.
```

4. Type the following at the shell prompt:

```
$ make eapol_test
```

5. Copy the file to a directory in your PATH, such as `/usr/local/bin` or `~/bin`, using the following command:

```
$ cp eapol_test ~/bin
$ PATH=$PATH:~/bin
$ export PATH
```

6.2.2 Testing with eapol_test

EAP-MD5

The previous EAP-MD5 test can be repeated with eapol_test.

1. Save the following file as `eap-md5.conf`:

```
network={
      key_mgmt=NONE
      eap=MD5
      identity="bob"
      password="hello"
}
```

2. Perform an EAP-MD5 test by running the following command:

```
eapol_test -c eap-md5.conf -s testing123 -n
```

This test uses the default server address of 127.0.0.1. Also be sure to use the `-n` parameter. If this parameter is not used, the eapol_test will expect to receive wireless encryption keys, which are not available for EAP-MD5. The program will then print FAILURE, even if the authentication succeeded.

You should see a number of lines of output, followed by:

```
...
MPPE keys OK: 0 mismatch: 0
SUCCESS
```

EAP-MSCHAPv2

To test EAP-MSCHAPv2, perform the following steps:

1. Save the following as `eap-mschapv2.conf`:

```
network={
      key_mgmt=WPA-EAP
      eap=MSCHAPV2
```

```
        identity="bob"
        password="hello"
}
```

2. To run the test, enter the following in the command line:

```
eapol_test -c eap-mschapv2.conf -s testing123
```

The same output as seen in the previous test will appear if the authentication is successful. If authentication has failed, the following output will be displayed:

```
...
MPPE keys OK: 0 mismatch: 0
FAILURE
```

6.3 TLS Based EAP Methods

Most of the widely used EAP methods are based on TLS, which used to be called SSL. These methods set up an encrypted tunnel between the supplicant and the RADIUS server. The supplicant verifies the server's identity by checking the server certificate, and then it passes the user's name and password inside the TLS tunnel. This step protects the password from third parties and is similar to logging into a secure web page via HTTPS.

This section describes how to create certificates for testing these EAP methods and also shows how to test EAP-TTLS, PEAP, and EAP-TLS. For the purposes of this section, it is assumed that the first line of the users file contains this test entry:

```
bob Cleartext-Password := "hello"
```

6.3.1 Test Certificates

To use TLS-based EAP methods, certificates must be created on the server. The certificates are located in the directory `raddb/certs`. The first time that the server is run in debugging mode, testing certificates are created. These certificates should be used for initial testing and then replaced by real certificates as soon as possible.

The certificates can be created by the following commands:

```
$ cd raddb/certs
$ ./bootstrap
```

If nothing is printed, it means that the certificates have already been created. Otherwise, there should be a lot of output showing how the certificates are being created. The default server configuration files point to these certificates, so there is nothing more to do to enable TLS-based EAP methods.

6.3.2 EAP-TTLS

We can now test the simplest method, EAP-TTLS.

1. Save the following file as `eap-ttls-pap.conf`:

```
network={
        key_mgmt=WPA-EAP
        eap=TTLS
```

```
        identity="bob"
        anonymous_identity="anonymous"
        password="hello"
        phase2="auth=PAP"
}
```

2. Test it via eapol_test:

```
eapol_test -c eap-ttls-pap.conf -s testing123
```

The above test is useful for a number of reasons. The `eapol_test` configuration shows whether or not the server is responding to EAP requests, as the configuration does not verify the server certificate against a list of authorized or known servers. Skipping the certificate verification step is insecure, but very useful for testing.

The data inside of the TLS tunnel is PAP authentication (i.e. `phase2="auth=PAP"`). As seen earlier, PAP is the authentication method that is the most likely to work across all possible databases and password formats.

The PAP authentication protocol can be replaced by any one of CHAP, MSCHAP, or MSCHAPV2. Alternately, the supplicant can tunnel EAP inside of EAP-TTLS by replacing the `auth=PAP` text with either `autheap=MSCHAPV2` for EAP-MSCHAPv2 or `autheap=MD5` for EAP-MD5.

6.3.3 PEAP

The Protected EAP (PEAP) authentication method is used primarily by Windows operating systems. It is similar to EAP-TTLS, except that it uses the configuration `phase2="autheap=MSCHAPV2"`. There are other differences between the two protocols, but these are not relevant here.

Running the test shown below before performing any authentication with Windows machines is recommended. If this test does not work, then no amount of tinkering with certificates or Windows configuration will help. Please look at the server debug output and the messages generated by `eapol_test` to figure out where the problem exists.

1. Save the following file as `peap-mschapv2.conf`:

```
network={
        key_mgmt=WPA-EAP
        eap=PEAP
        identity="bob"
        anonymous_identity="anonymous"
        password="hello"
        phase2="auth=MSCHAPV2"
        phase1="peapver=0"
}
```

2. Run `eapol_test`:

```
eapol_test -c peap-mschapv2.conf -s testing123
```

6.4 Certificates

The previous section described how to create test certificates and how to use them for authentication. This section covers Transport Layer Security (TLS) in greater detail.

The `openssl` command is run against the sample configuration files included with the server and creates a self-signed certificate authority (i.e. root CA) and a server certificate. The root CA should be installed on any client machine needing to do EAP-TLS, PEAP, or EAP-TTLS.

If the FreeRADIUS certificate build system is used, then Microsoft XP Extensions will automatically be included in the server certificate. Without those extensions, Windows clients will not work.

If FreeRADIUS was configured to use OpenSSL, then starting the server in root in debugging mode should also create test certificates; i.e.:

```
$ radiusd -X
```

This command will cause the EAP-TLS module to run the `bootstrap` script to create the certificates. The script will be executed only once: the first time the server has been installed on a particular machine. This bootstrap script should be run on installation of any pre-built binary package for your OS. In any case, the script will ensure that it is not run twice and that it does not over-write any existing certificates.

6.4.1 Certificate Creation

It is suggested that test certificates be used for initial tests on new installations and then "real" certificates be created to use for normal user authentication. This section details the process of creating the various test certificates.

Deleting the Test Certificates

The old test certificates can be deleted by running the following command:

```
$ rm -f *.pem *.der *.csr *.crt *.key *.p12 serial* index.txt*
```

After deleting old test certificates, the instructions below can be followed to create the real certificates. Once the final certificates have been created, the "bootstrap" command can be deleted from this directory; the "make_cert_command" configuration from the "tls" subsection of eap.conf should also be deleted.

If EAP-TLS, PEAP, or EAP-TTLS are not required, then the relevant sub-sections from the `eap.conf` file should also be deleted.

Making a Root Certificate

To create a root certificate, perform the following steps:

1. Open the file:

   ```
   $ cd raddb/certs
   $ vi ca.cnf
   ```

2. Edit the `input_password` and `output_password` fields to be the password for the CA certificate.

3. Edit the `[certificate_authority]` section to have the correct values for country, state, etc. Then, create the CA certificate:

   ```
   $ make ca.pem
   ```

4. Create the DER form of the self-signed certificate, which can be imported into Windows:

   ```
   $ make ca.der
   ```

Making a Server Certificate

To create a server certificate, perform the following steps:

1. Open the file:

```
$ cd raddb/certs
$ vi ca.cnf
```

2. Edit the `input_password` and `output_password` fields, inserting the password for the CA certificate.
3. Edit the `[server]` section, inserting the correct values for country, state, etc. Be sure that the `commonName` field here is different from the `commonName` for the CA certificate. Each certificate has to be unique, so no two certificates can share identical values for all fields.
4. Create the server certificate:

```
$ make server.pem
```

If you have a pre-existing certificate authority and wish to create a certificate signing request for the server certificate, then edit `server.cnf` as above and type the following command:

```
$ make server.csr
```

This command will create the certificate signing request, which then has to be sent to the certificate authority for signing.

Making a Client Certificate

Client certificates are used primarily by EAP-TLS and optionally by EAP-TTLS and PEAP. Most supplicants can use client certificates for EAP-TLS and not for EAP-TTLS or PEAP. The following steps outline how to create a client certificate that is signed by the server certificate created above. The password for the server certificate must be placed in the `input_password` and `output_password` fields of the `server.cnf` file.

1. Open the file:

```
$ cd raddb/certs
$ vi client.cnf
```

2. Edit the `input_password` and `output_password` fields, inserting the password for the client certificate. These passwords will have to be given to the end user, who will be using the certificates.
3. Edit the `[client]` section, inserting the correct values for country, state, etc. Be sure that the `commonName` field here is the `User-Name` that will be used for logins:

```
$ make client.pem
```

The users certificate will be place in the file named for the `commonName` field, which will be `emailAddress.pem` (e.g. `user@example.com.pem`.)

To create another client certificate, repeat the steps for making a client certificate, being sure to enter a a different password and a different login name for the `commonName` field. An error will be produced by OpenSSL if two certificates have the same values for all fields.

6.4.2 Potential Issues

This section details potential issues with certificates created using the FreeRADIUS tools. Potential issues with certificates fall into one of more of the following categories: compatibility, performance, and security.

Compatibility

The certificates created using the FreeRADIUS tools are known to be compatible with all operating systems. However, occasionally problems arise.

Some common issues and notes are noted below. When problems arise, please check the list below carefully for possible causes and solutions. For example, in none of the cases will Windows give the end user any reasonable error message describing what went wrong. The RADIUS server is often mistakenly blamed for these problems; that blame is misplaced.

- Windows requires certain OID's in the certificates. If these are not present, EAP will be stopped. The most visible effect is that the client starts EAP, gets a few Access-Challenge packets, and then a little while later re-starts EAP. If this happens, see the FAQ and the comments in `raddb/eap.conf` for how to fix the problem.
- Windows requires the root certificates to be on the client PC. If the root certificates are not on the client PC, the result will be the same issue as above.
- Windows XP post SP2 contains a bug that causes problems with certificate chains exists. For example, if the server certificate is an intermediate rather than a root certificate, then authentication will silently fail, as above.
- Some versions of Windows CE cannot handle 4K RSA certificates. They will (again) silently fail, as above.
- Prior to Vista, Windows had one working EAP implementation for wireless authentication and a completely separate implementation for wired authentication, which did not always work. For those trying to implement wired authentication using Windows, there is a simple solution: upgrade to Vista. The alternative is to avoid Windows as being too much of a headache.
- Certificate chains of more than 64K bytes are known to not work. This is a problem in FreeRADIUS. However, most clients cannot handle 64K certificate chains. Most Access Points will shut down the EAP session after about 50 round trips, while 64K certificate chains will take about 60 round trips. The solution is to not use large certificate chains.
- All other operating systems are known to work with EAP and FreeRADIUS. This includes Linux, *BSD, Mac OS X, Solaris, and Symbian, along with all known embedded systems, phones, and WiFi devices.

For companies who are thinking of writing their own EAP implementation, there is only one recommendation: Don't, it's a bad idea. The many layers of EAP methods, such as PEAP, are a rich source of implementation-specific behavior and inter-operability problems. The major RADIUS server and supplicant vendors have spent years working together to track down and resolve issues. There is little incentive for them to resolve problems created by yet another EAP implementation with a vanishingly small install base.

Our recommendation instead is to use the `wpa_supplicant` code base. It is simple, clean, BSD licensed, and it supports all common and most of the rarely used EAP types. Using `wpa_supplicant` will decrease costs and avoid interoperability issues.

Performance

EAP performance for EAP-TLS, TTLS, and PEAP is dominated by TLS calculations. That is, a normal system can handle PAP authentication at a rate of 10k packets/s. However, TLS involves RSA calculations, which are very expensive. To benchmark a system, do the following:

```
$ openssl speed rsa
```

or, to test 2048 bit keys:

```
$ openssl speed rsa2048
```

To stop the above test, it may be necessary to press CTRL-C.

To get the most accurate results, try to run the following program when the computer is idle:

```
Doing 2048 bit private rsa's for 10s: 914 2048 bit private RSA's in 9.98s
```

The above message shows that the system is doing about 90 RSA calculations per second. That number is the upper bound on EAP authentications that can be done on the system, and it does not include additional delays such as round trips for EAP packets.

In general, the maximum number of EAP authentications per second that can be done is half of the above number. So the system tested here will be capable of 40-45 EAP authentications per second. Owing to the limits of the underlying CPU, that is also the number of authentications per second FreeRADIUS can handle.

Since the TLS calculations are CPU bound, the only way to achieve a higher performance is more CPU power. Quad-core systems can parallelize the work. Four cores with the above performance may be able to get 4x45 (or 180) authentications per second.

In most situations, performance will not matter. When performance does matter, it is usually because a large NAS is rebooting. When that happens, the server receives a large number of authentication requests in a very short time. The only way the server can cope with that kind of overload situation is if it has been massively over-provisioned for the normal case.

In short, if running 802.1X, a small number of users per NAS is recommended.

With an NAS managing 802.1X for 10,000 users, the RADIUS server will pretty much melt down when the NAS reboots and all 10,000 users try to log in at the same time.

Security

In general, self-signed certificates should be used for 802.1X EAP authentication. When root CAs from other organizations are listed in the "CA_file", it gives those users permission to masquerade as the server, to authenticate users, and to issue client certificates for EAP-TLS. This level of permission is not likely what was desired.

Older certificate configuration files use MD5 for message digests in order to maintain compatibility with network equipment that supports only this algorithm. However, MD5 has known weaknesses and is discouraged in favor of SHA1 (see **http://www.kb.cert.org/vuls/id/836068** for details). Newer certificate configurations use SHA1, which is more secure.