Extensible Authentication Protocols

Cisco AR supports the Extensible Authentication Protocol (EAP) to provide a common protocol for differing authentication mechanisms. EAP enables the dynamic selection of the authentication mechanism at authentication time based on information transmitted in the Access-Request. (This type of EAP authentication mechanism is called an authentication exchange.)

Extensible Authentication Protocols (EAP) provide for support of multiple authentication methods. Cisco Access Registrar 4.0 supports the following EAP authentication methods:

In general, you enable each EAP method by creating and configuring a service of the desired type. Use the **radclient** test tool to confirm that the EAP service has been properly configured and is operational.

Both versions of Protected EAP (PEAP) are able to use other EAP methods as the authentication mechanism that is protected by PEAP encryption. For PEAP Version 0, the supported authentication methods are EAP-GTC and EAP-SIM. For PEAP Version 1, the supported authentication methods are EAP-MSChapVersion 2 and EAP-SIM.

The PEAP protocol consists of two phases: a mandatory authentication handshake phase and an optional tunnel phase where another complete EAP authentication exchange takes place protected by the session keys negotiated by phase one. Cisco Access Registrar 4.0 supports the tunneling of other EAP methods within the PEAP phase two exchange.

EAP-FAST

Cisco Access Registrar 4.0 supports the EAP-FAST authentication method. EAP-FAST uses the EAP-MSChapV2 method for credential provisioning and EAP-GTC for authentication. Credential provisioning typically occurs only during the client's initial EAP-FAST authentication. Subsequent authentications rely on the provisioned credential and will usually omit the provisioning step.

EAP-FAST is an authentication protocol designed to address the performance shortcomings of prior TLS-based EAP methods while retaining features such as identity privacy and support for password-based protocols. The EAP-FAST protocol is described by the IETF draft *draft-cam-winget-eap-fast-00.txt*.

The EAP-FAST credential is known as a Protected Access Credential (PAC) and contains information used to secure the authentication operations. Parts of the PAC are encrypted by the server and are not visible to other entities. Clients are expected to securely store PACs locally for use during authentication.

Configuring EAP-FAST involves creating and configuring the required EAP-MSChapV2 and EAP-GTC services as well as the EAP-FAST service with the appropriate parameters.

You can use the **radclient** test tool to confirm that the EAP services are properly configured and operational.

Configuring EAP-FAST

To enable EAP-FAST, use **aregcmd** to create and configure a service of type *eap-fast*.

Step 1 Launch **aregcmd** and create an EAP-FAST service.

cd /Radius/Services

add eap-fast-service

Step 2 Change directory to the service and set its type to eap-fast.

cd eap-fast-service

set type eap-fast

Step 3 Set the AuthorityIdentifier:

set AuthorityIdentifier ***authority-identifier***

Step 4 : Set the AuthorityInformation:

set AuthorityInformation ***authority-information***

Step 5 : Set the AuthentitcationService:

set AuthenticationService ***eap-gtc-service***

Step 6 : Set the ProvisionService:

set ProvisionService ***eap-mschapv2-service***

The follow example shows the default configuration for an EAP-FAST service:

```
[ //localhost/Radius/Services/eap-fast ]
    Name = eap-fast
    Description =
    Type = eap-fast
    IncomingScript~ =
    OutgoingScript~ =
    AuthorityIdentifier =
    AuthorityInformation =
    MaximumMessageSize = 1024
    PrivateKeyPassword =
    ServerCertificateFile =
    ServerRSAKeyFile =
    CACertificateFile =
    CACertificatePath =
    ClientVerificationMode = Optional
    VerificationDepth = 4
    EnableSessionCache = True
    SessionTimeout = "5 Minutes"
    AuthenticationTimeout = 120
    CredentialLifetime = Forever
    AuthenticationService =
    ProvisionMode = Anonymous
    ProvisionService =
    AlwaysAuthenticate = True
```

Table 8-1 lists and describes the EAP-FAST service properties.

Table 8-1 EAP-FAST Service Properties

Property	Description
IncomingScript	Optional script Cisco AR server runs when it receives a request from a client for PEAP-v0 service
OutgoingScript	Optional script Cisco AR server runs before it sends a response to a client using PEAP-v0
AuthorityIdentifier	A string that uniquely identifies the credential (PAC) issuer. The client uses this value to select the correct PAC to use with a particular server from the set of PACs it may have stored locally. Ensure that the AuthorityIdentifier is globally unique and that it does not conflict with identifiers used by other EAP-FAST servers or PAC issuers.
AuthorityInformation	A string that provides a descriptive text for this credential issuer. The value can be displayed to the client for identification purposes and might contain the enterprise or server names.
MaximumMessageSize	Indicates the maximum length in bytes that a PEAP or EAP-TLS message may have before it is fragmented.
PrivateKeyPassword	The password used to protect the server's private key.
ServerCertificateFile	The full pathname of the file containing the server's certificate or certificate chain used during the TLS exchange. The pathname may be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are PEM and DER. If an encoding prefix is not present, the file is assumed to be in PEM format.

Table 8-1 EAP-FAST Service Properties (continued)

Property	Description
ServerRSAKeyFile	The full pathname of the file containing the server's RSA private key. The pathname may be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are "PEM" and "DER". If an encoding prefix is not present, the file is assumed to be in PEM format. The following example assumes that the subdirectory **pki** under **/cisco-ar** contains the server's certificate file. The file **server-key.pem** is assumed to be in PEM format. The file extension ***.pem*** is not significant. **set ServerRSAKeyFile PEM:/cisco-ar/pki/server-key.pem**
CACertificateFile	The full pathname of the file containing trusted CA certificates used for client verification. The file may contain more than one certificate but all certificates must be in PEM format. DER encoding is not allowed.
CACertificatePath	The name of a directory containing trusted CA certificates (in PEM format) used for client verification. This parameter is optional, and if it is used there are some special preparations required for the directory it references. Each certificate file in this directory must contain exactly one certificate in PEM format. The server looks up the certificate files using the MD5 hash value of the certificate's subject name as a key. The directory must therefore also contain a set of symbolic links each of which points to an actual certificate file. The name of each symbolic link is the hash of the subject name of the certificate. For example, if a certificate file named **ca-cert.pem** is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in **ca-cert.path.pem** is 1b96dd93, then a symbolic link named 1b96dd93 must point to **ca-cert.pem**. If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extensions as in 1b96dd93.0 and 1b96dd93.1.
ClientVerificationMode	Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional. • RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one. • None will not request a client certificate. • Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.
VerificationDepth	Specifies the maximum length of the certificate chain used for client verification.
EnableSessionCache	Specifies whether TLS session caching (fast reconnect) is enabled or not. Set to True to enable session caching; otherwise set to False.
SessionTimeout	If TLS session caching (fast reconnect) is enabled, SessionTimeout specifies the maximum lifetime of a TLS session. Expired sessions are removed from the cache and will require a subsequent full authentication. SessionTimeout is specified as a string consisting of pairs of numbers and units, where units may be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks, as in the following: **Set SessionTimeout "1 Hour 45 Minutes"**

Table 8-1 EAP-FAST Service Properties (continued)

Property	Description
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out; defaults to 120.
CredentialLifetime	Specifies the maximum lifetime of a Protected Access Credential (PAC). Clients that successfully authenticate with an expired PAC will be re-provisioned with a new PAC. CredentialLifetime is specified as a string consisting of pairs of numbers and units, where units may be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks. Credentials that never expire should be specified as Forever.
AuthenticationService	Specifies the name of the EAP-GTC service is used for authentication. The named service must have the UseLabels parameter set to True.
ProvisionMode	Specifies the TLS mode used for provisioning. Clients only support the default Anonymous mode.
ProvisionService	Specifies the name of the EAP-MSChapV2 service used for provisioning.
AlwaysAuthenticate	Indicates whether provisioning should always automatically rollover into authentication without relying on a separate session. Most environments, particularly wireless, will perform better when this parameter is set to True, the default value.

EAP-FAST Keystores

The EAP-FAST service manages a set of keys used to protect the security and integrity of the PACs it issues. The keys are stored in**/Radius/Advanced/KeyStores/EAP-FAST** and are maintained automatically requiring minimal administration. Administrators can specify the maximum number of keys that are stored and the frequency of key updates.

The following is the default KeyStores settings:

```
[ //localhost/Radius/Advanced/KeyStores/EAP-FAST ]
        NumberOfKeys = 256
        RolloverPeriod = "1 Week"
```

Table 8-2 defines the KeyStores properties.

Table 8-2 KeyStores Properties

Property	Description
NumberOfKeys	Number (from 1-1024) that specifies the maximum number of keys stored for EAP-FAST.
RolloverPeriod	Specifies the amount of time between key updates.

Testing EAP-FAST with radclient

There are two distinct phases to testing EAP-FAST: provisioning and authentication. In the instructions below, Step 2 and Step 3 test provisioning and Steps 4 and Step 5 test authentication. At least one successful provisioning phase must be completed prior to testing authentication. Testing EAP-FAST with **radclient** requires that the EAP-MSChapV2 and EAP-GTC services be configured and functional.

The following instructions and examples assume that the AlwaysAuthenticate parameter has been set to False for testing purposes. This permits the provisioning and authentication steps to be tested separately. Most installations will set AlwaysAuthenticate to True for production use, and **radclient** works with that setting, but might display extra error messages that you can ignore.

Complete the following steps to test EAP-FAST using **radclient**:

Step 1 Start **radclient**.

cd /cisco-ar/usrbin

./radclient –s

Step 2 Specify the inner provisioning method

tunnel eap-mschapv2

The only allowable method for provisioning is eap-mschapv2.

Step 3 Provision a new PAC:

simple_eap_fast_test user-name password

Step 4 Specify the inner authentication method.

tunnel eap-gtc

The only allowable method for authentication is eap-gtc.

Step 5 Authenticate using the PAC.

simple_eap_fast_test user-name password

The **simple_eap_fast_test** command passes its arguments to the inner authentication mechanism which in turn treats the arguments as a username and a password. The command in Step 3 should result in provisioning a new PAC, and Step 5 should result in successful authentication using that PAC. The following examples demonstrate:

PAC Provisioning

The following example provisions a PAC for user bob.

pac show

```
No PAC(s) available to show
```

tunnel eap-mschapv2

```
PEAP tunnel method is eap-mschapv2
EAP-FAST tunnel method is eap-mschapv2
```

simple_eap_fast_test bob bob

```
EAP-FAST authentication status:
    [0x0e07] TLS authentication succeeded
Response to EAP-FAST message was not an Access-Accept
p012
```

pac show

```
PAC 1 version 1 (219 bytes)
    A-ID      : AR-4.0
    A-ID-Info : Cisco Systems Access Registrar
    I-ID      : bob
    Expires   : Never (0)
    Key#      : 12
    TLV  1    : PAC-Key (1) mandatory (32 bytes)
    TLV  2    : PAC-Opaque (2) mandatory (120 bytes)
    TLV  3    : PAC-Info (9) mandatory (51 bytes)
```

In this example the **simple_eap_fast_test** command indicates that it did not receive an AccessAccept. This is normal because the provisioning step always results in an AccessReject even when a new PAC has been successfully provisioned. The last **pac show** command displayed some status information from the new PAC and is used to verify that provisioning succeeded and authentication can now be tested. The PAC information displayed will vary and depends on how EAP-FAST is configured.

Authentication

The following example authenticates user bob (continuing from the PAC Provisioning example).

tunnel eap-gtc

```
PEAP tunnel method is eap-gtc
EAP-FAST tunnel method is eap-gtc
```

simple_eap_fast_test bob bob

```
EAP-FAST authentication status :
    [0x0e07] TLS authentication succeeded
SUCCESS : Correctly formatted Session Keys received from the server
p01e
```

In this example, the EAP_FAST authentication using the PAC from the previous provisioning step succeeded. The AccessAccept packet received from Cisco AR can be displayed to confirm that it contains the expected attributes including the MS-MPPE session keys.

Parameters Used for Certificate-Based Authentication

EAP-FAST might optionally use RSA certificates to securely create the tunnel that is used for PAC provisioning. However, the Cisco client does not support the use of certificates and the following parameters will be ignored and should be left at their default values:

PrivateKeyPassword

ServerCertificateFile

ServerRSAKeyFile

CACertificateFile

CACertificatePath

ClientVerificationMode

VerificationDepth

EnableSessionCache

SessionTimeout

The parameters for configuring certificate-based operation are identical to those used for PEAP and EAP-TLS.

Table 8-3 describes the parameters used for certificate-based authentication.

Table 8-3 Certificate-Based Authentication Parameters

Parameter	Description
AuthorityIdentifier	A string that uniquely identifies the credential (PAC) issuer. The client uses this value to select the correct PAC to use with a particular server from the set of PACs it may have stored locally. Care should be taken to ensure that the AuthorityIdentifier is globally unique, that is, is distinct from other PAC issuers
AuthorityInformation	A string that provides some descriptive text for this credential issuer. The value may be displayed to the client for identification purposes. It may contain the enterprise and/or server names.
MaximumMessageSize	Indicates the maximum length in bytes that a EAP-FAST message may have before it is fragmented. If certificates are not used for authentication, fragmentation should not be an issue.
AuthenticationTimeout	Indicates the maximum number of seconds before an authentication operation times out and is rejected.
CredentialLifetime	Specifies the maximum lifetime of a PAC (Protected Access Credential). Clients that successfully authenticate with an expired PAC will be re-provisioned with a new PAC.
AuthenticationService	Specifies the name of the EAP-GTC service that is used for authentication. The named service must have the UseLabels parameter set to True.
ProvisionMode	Specifies the TLS mode that is used for provisioning. As of this writing, clients only support the default Anonymous mode.
ProvisionService	Specifies the name of the EAP-MSChapV2 service that is used for provisioning.
AlwaysAuthenticate	Indicates whether provisioning should always automatically rollover into authentication without relying on a separate session. Most environments, particularly wireless, will perform better when this parameter is set to True (the default value).

radclient Command Reference

This section describes the **radclient** commands you can use to test EAP-FAST.

eap-trace

Use the **eap-trace** command to display additional client protocol trace information for EAP methods. Level is a number from 1 to 5 inclusively. Level 5 shows detailed hex dumps of all messages, level 4 shows a message trace without hex dumps, and levels 3 and below show status and error information. To turn off trace displays, set the level to 0.

Set the trace level for all EAP methods.

eap-trace ***level***

For example, the following command sets the trace level to 4 for all EAP methods.

eap-trace 4

Set the trace level for the specified EAP method.

eap-trace ***method level***

The following example sets the trace level to 5 for EAP-FAST only. The trace level for other EAP methods is not affected.

eap-trace eap-fast 5

Note The **eap-trace** command is for client-side trace information only and is independent of the server trace level that can be set using **aregcmd**.

tunnel

The **tunnel** command is used to specify the inner provisioning and authentication methods for EAP-FAST. The specified EAP method type must agree with the server's configured methods or authentication will fail.

tunnel eap-method

For EAP-FAST provisioning, the only allowable tunnel method is eap-mchavp2. For EAP-FAST authentication, the only allowable tunnel method is eap-gtc.

simple_eap_fast_test

The arguments are passed to the inner authentication method as its authentication parameters. If a PAC is not present, the tunnel method should be eap-mschapv2 and provisioning will occur. If a PAC is present, the tunnel method should be eap-gtc and authentication will occur.

simple_eap_fast_test ***username password***

There are also variants for the **simple** test command for other EAP methods as shown in the following examples:

simple_eap_mschapv2_test ***bob bob***

simple_eap_gtc_test ***bob bob***

pac

The **pac** command is used display, save, and delete PACs that are received from the server during testing. **radclient** maintains a cache of PACs that it knows about and that can be used for authentication testing. The current PAC cache can be displayed with the **pac show** command. PACs created during a test session can be stored to files with the **pac save** command, and reloaded in another session with the **pac load** command. The contents of the PAC cache are completely deleted with **pac delete**. If the optional parameter cache is included, PACs are also erased from disk.

pac ***load | save | show { hex } | delete { cache }***

The **pac show** command displays the currently cached PACs. If the optional parameter *hex* is included, additional detailed information including hex dumps are included in the display output.

pac show { hex }

The **pac load** command loads any previously saved PACS from disk into the active cache.

The **pac save** command saves all PACs from the active cache to disk. Any previously existing PACS for the same user will be over-written.

The **pac delete** command deletes all PACs from the active cache. If the optional cache parameter is included then PACs are also erased from disk.

pac delete { cache }

PAC—Credential Export Utility

You can manually provision EAP-FAST PACs to clients and avoid the use of the protocol provisioning phase. This might be desirable from a security perspective since the default provisioning protocol uses an anonymous (unauthenticated) method to construct the tunnel used to download the PAC to the client.

Manual provisioning involves exporting a PAC from Cisco AR to a file which is then copied to the client machine and used by the import utility. Once a PAC has been manually imported, the client should be able to authenticate via EAP-FAST while bypassing the initial provisioning phase. Care should be taken while storing and transporting PAC files since they contain information that potentially allows a client to authenticate via EAP-FAST.

PACs are exported from AR via the **pac** command which is a new utility for this release. (Note that this pac command is a stand-alone executable which is different from the Radclient pac command.) The **pac** command has two capabilities:

- Exports a PAC to a file
- Displays information about an existing PAC file

PAC Export

Use the **pac export** command to create a new PAC file. In the following example, *eap-fast* is the name of the Cisco AR service configured for EAP-FAST authentication, *bob* is the name of the user this PAC will be used for, and *password* is the password used to derive a key for encrypting the resulting file. (This password is not the same as the administrator's password). The PAC file will be named **bob.pac** by default. You can use the –f option to give the file a different name.

pac –s export ***eap-fast bob password***

If you omit the password parameter, a default password will be used.

Note Using the default password is strongly discouraged for security reasons.

PAC Display

Use the **pac show** command to display information about a PAC file. In the following example, **bob.pac** is the name of the PAC file and *password* is the password used to decrypt the file contents.

pac –s show ***bob.pac password***

Syntax Summary

The complete **pac** command syntax is as follows:

pac { options } export <service-name> <user-name> <file-password>

pac { options } show <file-name> file-<password>

Where:

-C <*cluster*>—Specifies the cluster to be used.

-N <*user*>—Specifies the user.

-P <*user-password*>—Specifies the password to be used.

-s —Logs in using defaults

-v—Enables verbose output

-f—Exports file name (default = {user-name}.pac)

EAP-GTC

EAP-GTC, defined in RFC 2284, is a simple method for transmitting a user's name and password to an authentication server. EAP-GTC should not be used except as an authentication method for PEAP Version 1 because the password is not protected.

Configuring EAP-GTC

To enable EAP-GTC, use **aregcmd** to create and configure a service of type *eap-gtc*.

Step 1 Launch **aregcmd** and create an EAP-GTC service.

cd /Radius/Services

add eap-gtc-service

Step 2 Change directory to the service and set its type to eap-gtc.

cd eap-gtc-service

set type eap-gtc

The follow example shows the default configuration for an EAP-GTC service:

```
[ //localhost/Radius/Services/eap-gtc-service ]
    Name = eap-gtc
    Description =
    Type = eap-gtc
    IncomingScript~ =
    OutgoingScript~ =
    AuthenticationTimeout = 120
    UserService =
    UserPrompt = "Enter password:"
    UseLabels = False
```

Table 8-4 lists and describes the EAP-GTC specific properties for EAP-GTC authentication.

Table 8-4 EAP-GTC Properties

Property	Description
UserService	Required; name of service that can be used to authenticate using cleartext passwords.
UserPrompt	Optional string the client might display to the user; default is Enter password:" Use the **set** command to change the prompt, as in the following: **set UserPrompt "Admin Password:"**
UseLabels	Required; must be set to TRUE for EAP-FAST authentication and set to FALSE for PEAP authentication. Set to FALSE by default.

Step 3 Set the service's UserService to local-users or another local authentication service that is able to authenticate using clear-text passwords.

set UserService local-users

Step 4 If configuring for EAP-FAST, set the UseLabels property to TRUE.

Testing EAP-GTC with radclient

To test the EAP-GTC service, launch **radclient** and use the **simple_eap_gtc_test** command. The **simple_eap_gtc_test** command sends an Access-Request for the designated user with the user's password.

The response packet should indicate an Access-Accept if authentication was successful. View the response packet to ensure the authentication was successful.

simple_eap_gtc_test bob bob

```
Packet: code = Access-Accept, id = 2, length = 104, attributes =
        Service-Type = Framed
        Framed-Protocol = PPP
        Framed-IP-Address = 192.168.0.0
        Framed-IP-Netmask = 255.255.255.0
        Framed-Routing = None
        Framed-MTU = 1500
        Framed-Compression = VJ TCP/IP header compression
        Framed-IPX-Network = 1
        EAP-Message = 03:01:00:04
        Ascend-Idle-Limit = 1800
        Message-Authenticator = d3:4e:b1:7e:2d:0a:ed:8f:5f:72:e0:01:b4:ba:c7:e0
```

EAP-LEAP

Cisco AR 4.0 supports the new AAA Cisco-proprietary protocol called Light Extensible Authentication Protocol (LEAP), a proprietary Cisco authentication protocol designed for use in IEEE 802.11 wireless local area network (WLAN) environments. Important features of LEAP include:

- Mutual authentication between the network infrastructure and the user
- Secure derivation of random, user-specific cryptographic session keys
- Compatibility with existing and widespread network authentication mechanisms (e.g., RADIUS)
- Computational speed

Note Cisco Access Registrar supports a subset of EAP to support LEAP. This is not a general implementation of EAP for Cisco Access Registrar.

The Cisco-Wireless or Lightweight Extensible Authentication Protocol is an EAP authentication mechanism where the user password is hashed based on an MD4 algorithm and verified by a challenge from both client and server.

Configuring EAP-LEAP

To enable EAP-LEAP, use **aregcmd** to create and configure a service of type **eap-leap**. When you create an EAP-LEAP service type, you must also specify a UserService to perform AAA service. The UserService can be any configured authentication service.

Step 1 Launch **aregcmd** and create an EAP-LEAP service.

cd /Radius/Services

add eap-leap-service

Step 2 Set the service type to **eap-leap**.

cd eap-leap-service

set type eap-leap

```
[ //localhost/Radius/Services/eap-leap-service ]
    Name = newone
    Description =
    Type =
    IncomingScript~ =
    OutgoingScript~ =
    AuthenticationTimeout = 120
    UserService =
```

Step 3 Set the UserService property to a configured authentication service.

EAP-MD5

Cisco AR 4.0 supports EAP-MD5, or MD5-Challenge, another EAP authentication exchange. In EAP-MD5 there is a CHAP-like exchange and the password is hashed by a challenge from both client and server to verify the password is correct. Once verified correct, the connection proceeds, although the connection is periodically re-challenged (per RFC 1994).

Configuring EAP-MD5

Specify type **eap-md5** when you create an EAP-MD5 service. When you create an EAP-MD5 service type, you must also specify a UserService to perform AAA service. The UserService can be any configured authentication service.

To enable EAP-MD5, use **aregcmd** to create and configure a service of type **eap-md5**. When you create an EAP-MD5 service type, you must also specify a UserService to perform AAA service. The UserService can be any configured authentication service.

Step 1 Launch **aregcmd** and create an EAP-LEAP service.

cd /Radius/Services

add eap-md5-service

Step 2 Set the service type to **eap-md5**.

cd eap-md5-service

set type eap-md5

```
[ //localhost/Radius/Services/eap-md5-service ]
    Name = newone
    Description =
    Type =
    IncomingScript~ =
    OutgoingScript~ =
    AuthenticationTimeout = 120
    UserService =
```

Step 3 Set the UserService property to a configured authentication service.

EAP-Negotiate

EAP-Negotiate is a special service used to select at run-time the EAP service to be used to authenticate the client. EAP-Negotiate is configured with a list of candidate EAP services that represent the allowable authentication methods in preference order. When an EAP session begins, the EAP-Negotiate service tires the first service in the list. If the client does not support that method, it will respond with an EAP-Nak message which triggers EAP-Negotiate to try the next method on the list until a valid method is found or the list is exhausted in which case authentication fails.

EAP-Negotiate is useful when the client population has deployed a mix of different EAP methods that must be simultaneously supported by Cisco Access Registrar. It can be difficult or impossible to reliably distinguish which clients require which methods simply by examining RADIUS attributes or other packet properties. EAP-Negotiate solves this problem by using the method negotiation feature of the EAP protocol. Negotiation can be used to select the primary EAP method used for authentication and also to select the inner method for PEAP.

Configuring EAP-Negotiate

To enable EAP-Negotiate, first use **aregcmd** to create and configure the EAP services that will be used for authentication, then create and configure a service of type eap-negotiate.

Step 1 Launch **aregcmd** and create an EAP-LEAP service.

cd /Radius/Services

add eap-negotiate-service

Step 2 Set the service type to **eap-negotiate**.

cd eap-negoitate-service

set type eap-negotiate

```
[ //localhost/Radius/Services/negotiate ]
    Name = negotiate
    Description =
    Type = eap-negotiate
    IncomingScript~ =
    OutgoingScript~ =
    AuthenticationTimeout = 120
    ServiceList =
```

Step 3 Set the ServiceList property to a list of pre-configured EAP authentication services.

The ServiceList property lists the names of the EAP services that can be negotiated with this instance of EAP-Negotiate. The ServiceList property is a space-separated list and must consist of valid EAP service name, *not service types*, in preference order from left to right. Each service and type on the list must be unique; duplicates are not allowed.

set ServiceList “eap-leap-service eap-md5-service peap-v1-service”

Negotiating PEAP Tunnel Services

EAP-Negotiate can also be used to negotiate the inner tunnel service used for phase two of PEAP-V0 or PEAP-V1. To do this, create and configure a service of type eap-negotiate. The ServiceList can only contain services that are legal for the version of PEAP that it is used with. Set the PEAP service's TunnelService parameter to the name of the eap-negotiate service.

Note Not all supplicants support negotiation of the PEAP inner method. EAP-Negotiate can only be used with supplicants that can use EAP-Nak to reject an unsupported inner method.

Testing EAP-Negotiate with radclient

You can test EAP-Negotiate using the same **radclient** commands used to test the other EAP services. For example, you can use the commands for testing eap-leap and peap-v1.

EAP-MSChapV2

EAP-MSChapv2 is based on **draft-kamath-pppext-eap-mschapv2-00.txt**, an informational IETF draft document. EAP-MSChapv2 encapsulates the MSChapV2 protocol (specified by RFC 2759) and can be used either as an independent authentication mechanism or as an inner method for PEAP Version 0 (recommended).

Configuring EAP-MSChapV2

To enable EAP-MSChapv2, use **aregcmd** to create and configure a service of type *eap-mschapv2*.

Step 1 Launch **aregcmd** and create an EAP-MSChapV2 service.

cd /Radius/Services

add eap-mschapv2

Note This example named the service eap-mschapv2, but you can use any valid name for your service.

Step 2 Set the service's type to eap-mschapv2.

cd eap-mschapv2

set Type eap-mschapv2

```
[ //localhost/Radius/Services/eap-mschapv2 ]
    Name = eap-mschapv2
    Description =
    Type = eap-mschapv2
    IncomingScript~ =
    OutgoingScript~ =
    AuthenticationTimeout = 120
```

```
    UserService =
    SystemID =
```

Step 3 Set the service's UserService to local-users or another local authentication service that is able to authenticate using MSChapV2.

set UserService local-users

Step 4 You might (optionally) set a string for System ID that identifies the sender of the MSChapV2 challenge message, as in the following:

set SystemID system_ID_string

Testing EAP-MSChapV2 with radclient

To test the EAP-MSChapVersion 2 service using **radclient**, perform the following the steps:

Step 1 Launch **radclient**.

Step 2 Use the **simple_eap_mschapv2_test** command to authenticate using EAP-MSChapV2, as in the following:

simple_eap_mschapv2_test bob bob

```
p006
```

The **simple_eap_mschapv2_test** command above sends an Access-Request for user bob with the user's password. The response packet should indicate an Access-Accept if authentication was successful.

Step 3 View the response packet to ensure the authentication was successful.

p006

```
Packet: code = Access-Accept, id = 4, length = 104, attributes =
        Service-Type = Framed
        Framed-Protocol = PPP
        Framed-IP-Address = 192.168.0.0
        Framed-IP-Netmask = 255.255.255.0
        Framed-Routing = None
        Framed-MTU = 1500
        Framed-Compression = VJ TCP/IP header compression
        Framed-IPX-Network = 1
        EAP-Message = 03:01:00:04
        Ascend-Idle-Limit = 1800
        Message-Authenticator = 27:90:7e:20:78:34:43:2e:9d:cd:a8:75:82:53:03:65
```

EAP-Transport Level Security (TLS)

EAP-Transport Level Security (EAP-TLS), described in RFC 2716, is an authentication method designed to mitigate several weaknesses of EAP. EAP-TLS leverages TLS, described in RFC 2246, to achieve certificate-based authentication of the server and (optionally) the client. EAP-TLS provides many of the same benefits as PEAP but differs from it in the lack of support for legacy authentication methods.

Configuring EAP-TLS

To enable EAP-TLS authentication, use **aregcmd** to create and configure a service of type *eap-tls*.

Step 1 Launch **aregcmd** and create an EAP-TLS service.

cd /Radius/Services

add eap-tls-service

Step 2 Change directory to the service and set its type to eap-tls.

cd eap-tls-service

set Type eap-tls

```
[ //localhost/Radius/Services/eap-tls-service ]
    Name = eap-tls
    Description =
    Type = eap-tls
    IncomingScript~ =
    OutgoingScript~ =
    MaximumMessageSize = 1024
    PrivateKeyPassword =
    ServerCertificateFile =
    ServerRSAKeyFile =
    CACertificateFile =
    CACertificatePath =
    ClientVerificationMode = Optional
    VerificationDepth = 4
    EnableSessionCache = True
    SessionTimeout = "5 Minutes"
    AuthenticationTimeout = 120
```

Table 8-5 describes the EAP-TLS configuration properties:

Table 8-5 EAP-TLS Service Properties

Property	Description
IncomingScript	Optional script Cisco AR server runs when it receives a request from a client for PEAP-v0 service
OutgoingScript	Optional script Cisco AR server runs before it sends a response to a client using PEAP-v0
MaximumMessageSize	Indicates the maximum length in bytes that a PEAP or EAP-TLS message may have before it is fragmented.
PrivateKeyPassword	The password used to protect the server's private key.

Table 8-5 EAP-TLS Service Properties (continued)

Property	Description
ServerCertificateFile	The full pathname of the file containing the server's certificate or certificate chain used during the TLS exchange. The pathname may be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are PEM and DER. If an encoding prefix is not present, the file is assumed to be in PEM format.
ServerRSAKeyFile	The full pathname of the file containing the server's RSA private key. The pathname may be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are "PEM" and "DER". If an encoding prefix is not present, the file is assumed to be in PEM format. The following example assumes that the subdirectory **pki** under **/cisco-ar** contains the server's certificate file. The file **server-key.pem** is assumed to be in PEM format. The file extension ***.pem*** is not significant. **set ServerRSAKeyFile PEM:/cisco-ar/pki/server-key.pem**
CACertificateFile	The full pathname of the file containing trusted CA certificates used for client verification. The file may contain more than one certificate but all certificates must be in PEM format. DER encoding is not allowed.
CACertificatePath	The name of a directory containing trusted CA certificates (in PEM format) used for client verification. This parameter is optional, and if it is used there are some special preparations required for the directory it references. Each certificate file in this directory must contain exactly one certificate in PEM format. The server looks up the certificate files using the MD5 hash value of the certificate's subject name as a key. The directory must therefore also contain a set of symbolic links each of which points to an actual certificate file. The name of each symbolic link is the hash of the subject name of the certificate. For example, if a certificate file named **ca-cert.pem** is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in **ca-cert.path.pem** is 1b96dd93, then a symbolic link named 1b96dd93 must point to **ca-cert.pem**. If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extensions as in 1b96dd93.0 and 1b96dd93.1.
ClientVerificationMode	Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional. • RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one. • None will not request a client certificate. • Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.
VerificationDepth	Specifies the maximum length (in bytes) of the certificate chain used for client verification.
EnableSessionCache	Specifies whether TLS session caching (fast reconnect) is enabled or not. Set to True to enable session caching; otherwise set to False.

Table 8-5 EAP-TLS Service Properties (continued)

Property	Description
SessionTimeout	If TLS session caching (fast reconnect) is enabled, SessionTimeout specifies the maximum lifetime of a TLS session. Expired sessions are removed from the cache and will require a subsequent full authentication. SessionTimeout is specified as a string consisting of pairs of numbers and units, where units may be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks, as in the following: **Set SessionTimeout "1 Hour 45 Minutes"**
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out; defaults to 120.

Testing EAP-TLS with radclient

To test the EAP-TLS service, launch **radclient** and use the **simple_eap_tls_test** command, as in the following:

```
simple_eap_tls_test arg1
```

The argument is arbitrary for the **simple_eap_tls_test** command and can be anything. (In the future, the argument may be used to select a client certificate.)

Testing EAP-TLS with Client Certificates

You can test EAP-TLS using client certificates verified by the server during the TLS exchange. The client certificate file and RSA key file must reside in **/cisco-ar/pki** and be named **client-cert.pem** and **client-key.pem** respectively. Both files must be in PEM format.

EAP-SIM

Cisco AR 4.0supports EAP-SIMv16. In a GSM network a subscriber is issued a *smart card* called the subscriber identity module (SIM) that contains a secret key (Ki) and an International Mobile Subscriber Identity (IMSI). The key (Ki) is also stored in the GSM authentication center located with the Home Location Registry (HLR).

An access point uses the Cisco AR RADIUS server to perform EAP-SIM authentication of mobile clients. Cisco AR must obtain authentication information from the HLR. Cisco AR contacts the MAP gateway that performs the MAP protocol over SS7 to the HLR.

Configuring EAP-SIM

To enable EAP-SIM authentication, use **aregcmd** to create and configure a service of type *eap-sim*.

Step 1 Launch **aregcmd** and create an EAP-TLS service.

```
cd /Radius/Services
```

add eap-sim-service

Step 2 Change directory to the service and set its type to *eap-sim*.

cd eap-sim-service

set Type eap-sim

```
[ //localhost/Radius/Services/eap-sim-service ]
    Name = eap-sim
        Description =
        Type = eap-sim
        IncomingScript~ =
        OutgoingScript~ =
        OutageScript~ =
        MultipleServersPolicy = Failover
        NumberOfTriplets = 2
        UseSimDemoTriplets = False
        AlwaysRequestIdentity = False
        EnableIdentityPrivacy = False
        PseudonymSecret = secret
        PseudonymRenewtime = "24 Hours"
        PseudonymLifetime = Forever
        EnableReauthentication = False
        MaximumReauthentications = 16
        ReauthenticationTimeout = 3600
        ReauthenticationRealm =
        TripletCacheTimeout = 120
        AuthenticationTimeout = 120
        UseProtectedResults = True
        RemoteServers/
```

Table 8-6 EAP-SIM Service Properties

Property	Description
IncomingScript~	Optional script Cisco AR server runs when it receives a request from a client for an EAP-SIM service.
OutgoingScript~	Optional script Cisco AR server runs before it sends a response to a client using an EAP-SIM service.
OutageScript~	Optional. If set to the name of a script, Cisco Access Registrar runs the script when an outage occurs. This property allows you to create a script that notifies you when the RADIUS server detects a failure.
MultipleServersPolicy	Required. Must be set to either Failover or RoundRobin. When set to Failover, Cisco Access Registrar directs requests to the first server in the list until it determines the server is off-line. At which time, Cisco Access Registrar redirects all requests to the next server in the list until it finds a server that is on-line. When set to RoundRobin, Cisco Access Registrar directs each request to the next server in the RemoteServers list in order to share the resource load across all of the servers listed in the RemoteServers list.
NumberOfTriplets	Number of triplets (1, 2, or 3) to use for authentication; default is 2.
UseSimDemoTriplets	Set to TRUE to enable the use of demo triplets. This must be disabled for release builds.

Table 8-6 EAP-SIM Service Properties (continued)

Property	Description
AlwaysRequestIdentity	When True, enables the server to obtain the subscriber's identity via EAP/SIM messages instead of relying on the EAP messages alone. This may be useful in cases where intermediate software layers can modify the identity field of the EAP-Response/Identity message. The default value is False.
EnableIdentityPrivacy	When True, the identity privacy feature is enabled. The default value is False.
PseudonymSecret	The secret string that is used as the basis for protecting identities when identity privacy is enabled. This should be at least 16 characters long and have a value that is impossible for an outsider to guess. The default value is secret. **Note** It is very important to change PseudonymSecret from its default value to a more secure value when identity privacy is enabled for the first time.
PseudonymRenewtime	Specifies the maximum age a pseudonym may have before it is renewed. When the server receives a valid pseudonym that is older than this, it generates a new pseudonym for that subscriber. The value is specified as a string consisting of pairs of numbers and units, where the units may be of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks. The default value is "24 Hours". Examples are: "8 Hours", "10 Hours 30 Minutes", "5 D 6 H 10 M"
PseudonymLifetime	Specifies the maximum age a pseudonym may have before it is rejected by the server, forcing the subscriber to authenticate using it's permanent identity. The value is specified as a string consisting of pairs of numbers and units, where the units may be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks. It may also be Forever, in which case, pseudonyms do not have a maximum age. The default value is "Forever". Examples are: "Forever", "3 Days 12 Hours 15 Minutes", "52 Weeks"
EnableReauthentication	When True, the fast re-authentication option is enabled. The default value is False.
MaximumReauthentications	Specifies the maximum number of times a re- authentication identity may be reused before it must be renewed. The default value is 16.
ReauthenticationTimeout	Specifies the time in seconds that re-authentication identities are cached by the server. Subscribers that attempt to re-authenticate using identities that are older than this value will be forced to use full authentication instead. The default value is 3600 (one hour).
ReauthenticationRealm	This information will be supplied later.
TripletCacheTimeout	Time in seconds an entry remains in the triplet cache. A zero (0) indicates that triplets are not cached. The maximum is 28 days; the default is 0 (no caching).
AuthenticationTimeout	Time in seconds to wait for authentication to complete. The default is 2 minutes; range is 10 seconds to 10 minutes.
UseProtectedResults	Enables or disables the use of protected results messages. Results messages indicate the state of the authentication but are cryptographically protected.
RemoteServers/	List of remote RADIUS servers which are map gateways. The remote server type must be set to map-gateway.

Note The EAP-SIM property OutagePolicy present in earlier versions of Cisco AR is no longer part of the EAP-SIM configuration.

radclient Command Reference

This section provides a summary of the **radclient** commands you can use to test PEAP and EAP-TLS.

eap-trace

Use the **eap-trace** command to display additional client protocol trace information for EAP methods. Set the level to a number from 1 to 5 inclusively. Level 5 shows detailed hexadecimal dumps of all messages. Level 4 shows a message trace without hexadecimal dumps. Levels 3 and below show status and error information. To turn off trace displays, set the level to 0.

Use **eap-trace level** to set the trace level for all EAP methods. The following example command sets the trace level to 4 for all EAP methods:

eap-trace 4

Use **eap-trace method level** to set the trace level for the specified EAP method. The following example command sets the trace level to 5 for PEAP Version0 only. The trace level for other EAP methods is not affected.

eap-trace peap-v0 5

Note The **eap-trace** command is for client-side trace information only and is independent of the server trace level you set using **aregcmd**.

tunnel

Use the **tunnel** command to specify the inner authentication method for PEAP. The specified EAP method type must agree with the server's configured authentication method or authentication will fail.

tunnel eap-method

For PEAP Version 0, the allowable tunnel methods are EAP-MSCHAPV2 and EAP-SIM. For PEAP Version 1, the allowable tunnel methods are EAP-GTC and EAP-SIM.

simple_eap_mschapv2_test username password

simple_eap_gtc_test username password

simple_eap_peapv0_test arg1 arg2

The arguments are passed to the inner authentication method as its authentication parameters. For EAP-MSChapv2 the arguments are username and password; for EAP-SIM they are IMSI and key.

simple_eap_peapv1_test arg1 arg2

The arguments are passed to the inner authentication method as its authentication parameters. For EAP-GTC the arguments are username and password; for EAP-SIM they are IMSI and key.

simple_eap_tls_test arg1

Protected EAP

Protected EAP (PEAP) is an authentication method designed to mitigate several weaknesses of EAP. PEAP leverages TLS (RFC 2246) to achieve certificate-based authentication of the server (and optionally the client) and creation of a secure session that can then be used to authenticate the client. PEAP provides several benefits:

- Industry standard authentication of the server using certificates (TLS)
- Standardized method for session key generation using TLS PRF
- Strong mutual authentication
- Identity privacy
- Fast reconnect using TLS session caching
- EAP message fragmentation
- Secure support for legacy client authentication methods

Cisco Access Registrar 4.0 supports the two major existing variants of PEAP, PEAP Version 0 (Microsoft PEAP) and PEAP Version 1 (Cisco PEAP). PEAP Version 0 is described in IETF drafts **draft-kamath-pppext-peapv0-00.txt** and **draft-josefsson-pppext-eap-tls-eap-02.txt**. This version of PEAP can use either EAP-MSChapV2 or EAP-SIM as an authentication method. PEAP Version 1 is described by IETF draft **draft-zhou-pppext-peapv1-00.txt**. PEAP Version 1 can use either EAP-GTC or EAP-SIM as an authentication method.

PEAP Version 0

This section describes configuring PEAP Version 0 and testing it with **radclient**.

Configuring PEAP Version 0

To enable PEAP Version 0, use **aregcmd** to create and configure a service of type *peap-v0*.

Step 1 Launch **aregcmd** and create a PEAP Version 0 service.

cd /Radius/Services

add peap-v0-service

Step 2 Set the service's type to peap-v0.

cd peap-v0-service

set Type peap-v0

```
[ //localhost/Radius/Services/peap-v0-service ]
    Name = peap-v0
    Description =
    Type = peap-v0
    IncomingScript~ =
    OutgoingScript~ =
    MaximumMessageSize = 1024
    PrivateKeyPassword =
    ServerCertificateFile =
```

```
ServerRSAKeyFile =
CACertificateFile =
CACertificatePath =
ClientVerificationMode = Optional
VerificationDepth = 4
EnableSessionCache = True
SessionTimeout = "5 Minutes"
AuthenticationTimeout = 120
TunnelService =
```

Step 3 Set the service's TunnelService property to the name of an existing EAP-MSCHAPV2 or EAP-SIM service.

set TunnelService name_of_EAP-MSCHAPv2_service

or

set TunnelService name_of_EAP-SIM_service

Table 8-7 describes the PEAP service properties for PEAP Version 0.

Table 8-7 PEAP Version 0 Service Properties

Property	Description
IncomingScript	Optional script Cisco AR server runs when it receives a request from a client for PEAP-v0 service
OutgoingScript	Optional script Cisco AR server runs before it sends a response to a client using PEAP-v0
MaximumMessageSize	Indicates the maximum length in bytes that a PEAP or EAP-TLS message may have before it is fragmented.
PrivateKeyPassword	The password used to protect the server's private key.
ServerCertificateFile	The full pathname of the file containing the server's certificate or certificate chain used during the TLS exchange. The pathname may be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are PEM and DER. If an encoding prefix is not present, the file is assumed to be in PEM format. The following example assumes that the subdirectory **pki** under **/cisco-ar** contains the server's certificate file. The file **server-cert.pem** is assumed to be in PEM format; note that the file extension ***.pem*** is not significant. **set ServerCertificateFile PEM:/cisco-ar/pki/server-cert.pem**
CACertificateFile	The full pathname of the file containing trusted CA certificates used for client verification. The file may contain more than one certificate but all certificates must be in PEM format. DER encoding is not allowed.

Table 8-7 PEAP Version 0 Service Properties (continued)

Property	Description
CACertificatePath	The name of a directory containing trusted CA certificates (in PEM format) used for client verification. This parameter is optional, and if it is used there are some special preparations required for the directory it references. Each certificate file in this directory must contain exactly one certificate in PEM format. The server looks up the certificate files using the MD5 hash value of the certificate's subject name as a key. The directory must therefore also contain a set of symbolic links each of which points to an actual certificate file. The name of each symbolic link is the hash of the subject name of the certificate. For example, if a certificate file name **ca-cert.pem** is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in **ca-cert.path.pem** is 1b96dd93, then a symbolic link named 1b96dd93 must point to the **ca-cert.pem** file. If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extensions as in 1b96dd93.0 and 1b96dd93.1.
ClientVerificationMode	Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional. • RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one. • None will not request a client certificate. • Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.
VerificationDepth	Specifies the maximum length of the certificate chain used for client verification.
EnableSessionCache	Specifies whether TLS session caching (fast reconnect) is enabled or not. Set to True to enable session caching; otherwise set to False.
SessionTimeout	If TLS session caching (fast reconnect) is enabled, SessionTimeout specifies the maximum lifetime of a TLS session. Expired sessions are removed from the cache and will require a subsequent full authentication. SessionTimeout is specified as a string consisting of pairs of numbers and units, where units may be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks, as in the following: **Set SessionTimeout "1 Hour 45 Minutes"**
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out; defaults to 120.
TunnelService	Mandatory; must be the name of an existing EAP-MSCHAPv2 or EAP-SIM service for PEAP Version 0.

Testing PEAP Version 0 with radclient

To test the PEAP Version 0, complete the following steps:

Step 1 Launch **radclient**.

Step 2 Specify the inner authentication method, eap-mschapv2 or eap-sim, as in the following.

tunnel eap-mschapv2

or

tunnel eap-sim

Step 3 Use the **simple_eap_peapv0_test** command to authenticate using PEAP Version 0, as in the following:

simple_eap_peapv0_test arg1 arg2

The **simple_eap_peapv0_test** command passes its arguments to the inner authentication mechanism which treats the arguments as either a username and a password (for eap-mschapv2) or as an IMSI and a key (for eap-sim).

The following example tests PEAP Version 0 with EAP-MSCHAPV2 as the inner authentication mechanism using username bob and password bob:

tunnel eap-mschapv2

simple_eap_peapv0_test bob bob

The following example tests PEAP Version 0 with EAP-SIM as the inner authentication mechanism using IMSI 1234567891 and key 0123456789ABCDEF:

tunnel eap-sim

simple_eap_peapv0_test 1234567891 0123456789ABCDEF

Testing PEAP Version 0 with Client Certificates

You can test PEAP Version 0 using client certificates verified by the server during the TLS exchange. The client certificate file and RSA key file must reside in **/cisco-ar/pki** and be named **client-cert.pem** and **client-key.pem** respectively. Both files must be in PEM format.

PEAP Version 1

This section describes configuring PEAP Version 1 and testing it with **radclient**.

Configuring PEAP Version 1

To enable PEAP Version 1, use **aregcmd** to create and configure a service of type *peap-v1*.

Step 1 Launch **aregcmd** and create a PEAP Version 1 service.

cd /Radius/Services

add peap-v1-service

Step 2 Set the service's type to peap-v1.

cd peap-v1-service

set Type peap-v1

```
[ //localhost/Radius/Services/peap-v1-service ]
    Name = peap-v1-service
    Description =
    Type = peap-v1
    IncomingScript~ =
    OutgoingScript~ =
    MaximumMessageSize = 1024
    PrivateKeyPassword =
    ServerCertificateFile =
    ServerRSAKeyFile =
    CACertificateFile =
    CACertificatePath =
    ClientVerificationMode = Optional
    VerificationDepth = 4
    EnableSessionCache = True
    SessionTimeout = "5 Minutes"
    AuthenticationTimeout = 120
    TunnelService =
```

Step 3 Set the service's TunnelService property to the name of an existing EAP-GTC or EAP-SIM service.

set TunnelService name_of_EAP-GTC_service

or

set TunnelService name_of_EAP-SIM_service

Table 8-8 describes the PEAP service properties for both PEAP Version 1.

Table 8-8 PEAP Version 1 Service Properties

Property	Description
IncomingScript	Optional script Cisco AR server runs when it receives a request from a client for PEAP-v0 service
OutgoingScript	Optional script Cisco AR server runs before it sends a response to a client using PEAP-v0
MaximumMessageSize	Indicates the maximum length in bytes that a PEAP or EAP-TLS message may have before it is fragmented.
PrivateKeyPassword	The password used to protect the server's private key.
ServerCertificateFile	The full pathname of the file containing the server's certificate or certificate chain used during the TLS exchange. The pathname may be optionally prefixed with a special string that indicates the type of encoding used for the certificate. The two valid encoding prefixes are PEM and DER. If an encoding prefix is not present, the file is assumed to be in PEM format.
CACertificateFile	The full pathname of the file containing trusted CA certificates used for client verification. The file may contain more than one certificate but all certificates must be in PEM format. DER encoding is not allowed.

Table 8-8 PEAP Version 1 Service Properties (continued)

Property	Description
CACertificatePath	The name of a directory containing trusted CA certificates (in PEM format) used for client verification. This parameter is optional, and if it is used there are some special preparations required for the directory it references. Each certificate file in this directory must contain exactly one certificate in PEM format. The server looks up the certificate files using the MD5 hash value of the certificate's subject name as a key. The directory must therefore also contain a set of symbolic links each of which points to an actual certificate file. The name of each symbolic link is the hash of the subject name of the certificate. For example, if a certificate file named **ca-cert.pem** is located in the CACertificatePath directory, and the MD5 hash of the subject name contained in **ca-cert.path.pem** is 1b96dd93, then a symbolic link named 1b96dd93 must point to the **ca-cert.pem** file. If there are subject name collisions such as multiple certificates with the same subject name, each link name must be indexed with a numeric extensions as in 1b96dd93.0 and 1b96dd93.1.
ClientVerificationMode	Specifies the type of verification used for client certificates. Must be set to one of RequireCertificate, None, or Optional. • RequireCertificate causes the server to request a client certificate and authentication fails if the client refuses to provide one. • None will not request a client certificate. • Optional causes the server to request a client certificate but the client is allowed to refuse to provide one.
VerificationDepth	Specifies the maximum length of the certificate chain used for client verification.
EnableSessionCache	Specifies whether TLS session caching (fast reconnect) is enabled or not. Set to True to enable session caching; otherwise set to False.
SessionTimeout	If TLS session caching (fast reconnect) is enabled, SessionTimeout specifies the maximum lifetime of a TLS session. Expired sessions are removed from the cache and will require a subsequent full authentication. SessionTimeout is specified as a string consisting of pairs of numbers and units, where units may be one of the following: M, Minute, Minutes, H, Hour, Hours, D, Day, Days, W, Week, Weeks, as in the following: **Set SessionTimeout "1 Hour 45 Minutes"**
AuthenticationTimeout	Mandatory; specifies time (in seconds) to wait before an authentication request times out; defaults to 120.
TunnelService	Mandatory; must be the name of an existing EAP-GTC or EAP-SIM service for PEAP Version 0.

Testing PEAP Version 1 with radclient

To test the PEAP Version 1, complete the following steps:

Step 1 Launch **radclient**.

Step 2 Specify the inner authentication method, EAP-GTC or EAP-SIM, as in the following.

tunnel eap-gtc

or

tunnel eap-sim

Step 3 Use the **simple_eap_peapv1_test** command to authenticate using PEAP Version 1, as in the following:

simple_eap_peapv1_test arg1 arg2

The **simple_eap_peapv1_test** command passes its arguments to the inner authentication mechanism which treats the arguments as either a username and a password (for EAP-GTC) or as an IMSI and a key (for EAP-SIM).

Testing PEAP Version 1 with Client Certificates

You can test PEAP Version 1 using client certificates verified by the server during the TLS exchange. The client certificate file and RSA key file must reside in **/cisco-ar/pki** and be named **client-cert.pem** and **client-key.pem** respectively. Both files must be in PEM format.