

Using Extension Points

Revised: March 20, 2009, OL-17222-03

This chapter describes how to use Cisco Access Registrar (CAR) scripting to customize your RADIUS server. This chapter contains the following sections:

- [Determining the Goal of the Script, page 10-2](#)
- [Writing the Script, page 10-2](#)
- [Adding the Script Definition, page 10-4](#)
- [About the Tcl/Tk 8.3 Engine, page 10-6](#)
- [Cisco Access Registrar Scripts, page 10-6](#)

You can write scripts to affect the way CAR handles and responds to requests and to change the behavior of CAR after a script is run.

All scripts reference the three dictionaries listed below, which are data structures that contain key/value pairs.

- Request dictionary—contains all of the attributes from the access-request or other incoming packets, such as the username, password, and service hints
- Response dictionary—contains all of the attributes in the access-accept or other response packets. As these are the attributes the server sends back to the NAS, you can use this dictionary to add or remove attributes.
- Environment dictionary—contains well-known keys whose values enable scripts to communicate with CAR or to communicate with other scripts.

The process for creating and implementing a script involves:

- Determining the goal of the script
- Writing the script
- Adding the new script definition to CAR
- Choosing a scripting point from within CAR
- Testing the script using the **radclient** command



Note

[“Overview”](#).

Determining the Goal of the Script

The goal of the script and its scripting point are tied together. For example, when you want to create a script that performs some special processing of a username before it is processed by the CAR server, you would reference this script as an *incoming*

outgoing

processes requests and responses in a hierarchical fashion; incoming requests are processed from the most general to the most specific levels, whereas, outgoing responses are processed from the most specific to the most general levels. Extension points are available at each level.

An incoming script can be referenced at each of the following extension points:

- RADIUS server
- Vendor (of the immediate client)
- Client (individual NAS)
- NAS-Vendor-Behind-the-Proxy
- Client-Behind-the-Proxy
- Remote Server (of type RADIUS)
- Service

An authentication or authorization script can be referenced at each of the following extension points:

- Group Authentication
- User Authentication
- Group Authorization
- User Authorization

The outgoing script can be referenced at each of the following extension points:

- Service
- Client-Behind-the-Proxy
- NAS-Vendor-Behind-the-Proxy
- Client (individual NAS)
- NAS Vendor
- RADIUS server

Writing the Script

You can write scripts in either Tcl or as shared libraries using C or C++. In this section, the scripts are shown in Tcl.

To write a script, do the following:

-
- Step 1** Using an editor, create the Tcl source file.
 - Step 2** Give it a name.

Step 3 Define one or more procedures, using the following syntax:

```
proc name {request response environment}
{Body}
```

Step 4 Create the body of the script.

Step 5 Save the file and copy it to the `/opt/CSCOAr/scripts/radius/tcl` directory, or to the location you chose when you installed CAR.

Choosing the Type of Script

When you create a script you can use any one or all of the three dictionaries: Request, Response, or Environment.

- When you use the Request dictionary, you can modify the contents of a NAS request. Scripts that use the Request dictionary are usually employed as incoming scripts.
- When you use the Response dictionary, you can modify what the server sends back to the NAS. These scripts are consequently employed as outgoing scripts.
- When you use the Environment dictionary, you can do the following:
 - Affect the behavior of the server after the script is run. For example, you can use the Environment dictionary to decide which of the multiple services to use for authorization, authentication, and accounting.
 - Communicate among scripts, as the scripts all share these three dictionaries. For example, when a script changes a value in the Environment dictionary, the updated value can be used in a subsequent script.

The following examples show scripts using all three dictionaries.

Request Dictionary Script

The Request Dictionary script is referenced from the server's IncomingScript scripting point. It checks to see whether the request contains a **NAS-Identifier** or a **NAS-IP-Address**. When it does not, it sets the **NAS-IP-Address** from the request's source IP address.

```
proc MapSourceIPAddress {request response environment}
{
    if { ! ( [ $request containsKey NAS-Identifier ] ||
            [ $request containsKey NAS-IP-Address ] ) } {
        $request put NAS-IP-Address [ $environment get Source-IP-Address ]
    }
}
```

Tcl scripts interpret **\$request** arguments as active commands that can interpret strings from the Request dictionary, which contains keys and values.

The **containsKey** method has the syntax: `<$dict> containsKey <attribute>`. In this example, `<$dict>` refers to the Request dictionary and the attributes **NAS-identifier** and **NAS-IP-Address**. The **containsKey** method returns **1** when the dictionary contains the attribute, and **0** when it does not. Using the **containsKey** method prevents you from overwriting an existing value.

The **put** method has the syntax: `<$dict> put <attribute value>[<index>]`. In this example, `<$request>` refers to the Request dictionary and the attribute is **NAS-IP-Address**. The **put** method sets the NAS's IP address attribute.

The **get** method has the syntax: `<$dict> get <attribute>`. In this example, `<$dict>` refers to the Environment dictionary and `<attribute>` is the **Source-IP-Address**. The **get** method returns the value of the attribute from the environment dictionary.

For a list of the methods you can use with scripts, see [Appendix A, “Cisco Access Registrar Tcl and REX Dictionaries.”](#) They include **get**, **put**, and others.

Response Dictionary Script

This script is referenced from either the user record for users whose sessions are always PPP, or from the script, **AuthorizeService**, which checks the request to determine which service is desired. The script merges the Profile named **default-PPP-users** into the Response dictionary.

```
proc AuthorizePPP {request response environment}
{
    $response addProfile default-PPP-users
```

The **addProfile** method has the syntax: `<$dict> addProfile <profile>[<mode>]`. In this example, `<$dict>` refers to the Response dictionary and the profile is **default-PPP-users**. The script copies all of the attributes of the Profile `<profile>` into the dictionary.

Environment Dictionary Script

This script is referenced from the NAS Incoming Script scripting point. It looks for a realm name on the username attribute to determine which AAA services should be used for the request. When it finds `@radius`, it selects a set of AAA services that will proxy the request to a remote RADIUS server. When it finds `@tacacs`, it selects the Authentication Service that will proxy the request to a TACACS server for authentication. For all of the remaining usernames, it uses the default Service (as specified in the configuration by the administrator).

Note the function, **regsub**, is a Tcl function.

```
proc ParseProxyHints {request response environment} {
    set userName [ $request get User-Name ]
    if { [ regsub "@radius" $userName "" newUser ] } {
        $request put User-Name $newUser
        $radius put Authentication-Service "radius-proxy"
        $radius put Authorization-Service "radius-proxy"
        $radius put Accounting-Service "radius-proxy"
    } else {
        if { [ regsub "@tacacs" $userName "" newUser ] } {
            $request put User-Name
            $radius put Authentication-Service "tacacs-client"
```

Adding the Script Definition

After you have written the script, you must add the script definition to the CAR's script **Configuration** directory so it can be referenced. Adding the script definition involves:

- Specifying the script definition; it must include the following:
 - **Name**—used in other places in the configuration to refer to the script. It must be unique among all other scripts.
 - **Language**—can be either Tcl or REX (shared libraries)
 - **Filename**—the name you used when you created the file

- **EntryPoint**—the function name.

The **Name** and the **EntryPoint** can be the same name, however they do not have to be.

- Choosing a scripting point; nine exist for incoming and outgoing scripts. These include:
 - the server
 - the vendor of the immediate client
 - the immediate client
 - the vendor of the specific NAS
 - the specific NAS
 - the service (rex or tcl)
 - the group (only AA scripts)
 - the user record (only AA scripts)
 - remote server (only type RADIUS)

The rule of thumb to use in determining where to add the script is the most general scripts should be on the outermost points, whereas the most specific scripts should be on the innermost points.


Note

The client and the NAS are the same entity, unless the immediate client is acting as a proxy for the actual NAS.

Adding the Example Script Definition

In the server configuration a **Scripts** directory exists. You must add the script you created to this directory. To add the **ParseProxyHints** script to the CAR server, enter the following command and supply the following information:

```
Name=ParseProxyHints
Description=ParseProxyHints
Language=tcl
Filename=ParseProxyHints
Entrypoint=ParseProxyHints
```

```
aregcmd add /Radius/Scripts/ParseProxyHints ParseProxyHints tcl ParseProxyHints
ParseProxyHints
```

Choosing the Scripting Point

As the example script, **ParseProxyHints**, applies to a specific NAS (NAS1), the entry point should be that NAS. To specify the script at this scripting point, enter:

```
aregcmd set /Radius/Clients/NAS1/IncomingScript ParseProxyHints
```

Testing the Script

To test the script, you can use the **radclient** command, which lets you create and send packets. For more information about the **radclient** command, see [Chapter 2, “Using the aregcmd Commands.”](#)

About the Tcl/Tk 8.3 Engine

CAR1.6 and above uses Tcl engine is version Tcl/Tk8.3. Since the Tcl/Tk8.3 engine supports a multi-threading application environment, it can achieve much better performance than Tcl/Tk7.6.

**Note**

In this release, scripts that use Tcl global variables will not work across CAR extension points. A future release will address script compatibility issues.

Tcl/Tk8.3 also performs *byte-compile*, so no run-time interpretation is required.

Cisco Access Registrar Scripts

The CAR scripts are stored in `/localhost/Radius/Scripts`. Most of the scripts are written in the RADIUS Extension language (REX). Some scripts are provided in both REX and Tcl. The scripts written in Tcl all begin with the letter **t** followed by their functional name. The Tcl scripts are listed below:

- tACMEOutgoingScript
- tAuthorizePPP
- tAuthorizeService
- tAuthorizeTelnet
- tMapSourceIPAddress
- tParseARealm
- tParseAASRealm
- tParseProxyHints
- tParseServiceAndAARealmHints
- tParseServiceAndAAASRealmHints
- tParseServiceAndARealmHints
- tParseServiceAndAASRealmHints
- tParstServiceAndProxyHints
- tParseServiceHints

ACMEOutgoingScript

ACMEOutgoingScript is referenced from Vendor ACME for the outgoing script. If the CAR server accepts this Access-Request and the response does not yet contain a Session-Timeout, set it to 3600 seconds.

AltigaIncomingScript

AltigaIncomingScript maps Altiga-proprietary attributes to CAR's global attribute space.

AltigaOutgoingScript

AltigaOutgoingScript maps Altiga attributes from CAR's global attribute space to the appropriate Altiga-proprietary attributes.

ANAAAOutgoing

ANAAAOutgoing can be referenced from either the client or vendor outgoing scripting point to be used in HRPD/EV-DO networks where CAR is the Access Network (AN) AAA server. ANAAAOutgoing checks to see if the response contains the Callback-Id attribute. If the response contains the Callback-Id attribute and the value is less than 253 characters, ANAAAOutgoing prefixes a zero (0) to the value. For example, it changes "123" into "0123." The ANAAAOutgoing script always returns REX_OK.

AscendIncomingScript

AscendIncomingScript maps Ascend-proprietary attributes to CAR's global attribute space.

AscendOutgoingScript

AscendOutgoingScript maps Ascend attributes from CAR's global attribute space to the appropriate Ascend-proprietary attributes.

AuthorizePPP

AuthorizePPP is referenced from either the use record for users who's sessions are always PPP or from the from the script AuthorizeService, which checks the request to determine which service is desired. This script merges in the Profile named "default-PPP-users" into the response dictionary.

AuthorizeService

AuthorizeService is referenced from user record for users who's sessions might be PPP, SLIP or Telnet depending on how they are connecting to the NAS. This script checks the request to determine which service is desired. If it is telnet, it calls the script AuthorizeTelnet. If it is PPP, it calls the script AuthorizePPP. If it is SLIP, it calls the script AuthorizeSLIP. If it is none of these, it rejects the request.

AuthorizeSLIP

AuthorizeSLIP is referenced from either the user record for users who's sessions are always SLIP or from the from the script AuthorizeService, which checks the request to determine which service is desired. This script merges in the Profile named "default-SLIP-users" into the response dictionary.

AuthorizeTelnet

AuthorizeTelnet is referenced from either the user record for users who's sessions are always telnet or from the from the script AuthorizeService, which checks the request to determine which service is desired. This script merges in the Profile named "default-Telnet-users" into the response dictionary.

CabletronIncoming

CabletronIncoming maps Cabletron-proprietary attributes to CAR's global attribute space.

CabletronOutgoing

Use CabletronOutgoing to map Cisco-proprietary attributes from CAR's global attribute space to the appropriate Cabletron-proprietary attributes.

CiscoIncoming

Use CiscoIncoming to map Cisco-proprietary attributes to CAR's global attribute space.

CiscoOutgoing

Use CiscoOutgoing to map Cisco-proprietary attributes from CAR's global attribute space to the appropriate Cabletron-proprietary attributes.

CiscoWithODAPIncomingScript

Use CiscoWithODAPIncomingScript to map Cisco-proprietary attributes to CAR's global attribute space and to map ODAP requests to the appropriate services and session managers.

CiscoWithODAPIncomingScript checks the incoming NAS-Identifier sent by the client. If the NAS-Identifier does not equal odap-dhcp, the request is not an ODAP request. If the request is not an ODAP request, the script does no more ODAP-specific processing, and calls CiscoIncomingScript to allow it to process the request.

If the request is an ODAP request, CiscoWithODAPIncomingScript removes the NAS-Identifier attribute because it is no longer required. The script then sets the Authentication-Service and the Authorization-Service to odap-users and sets the Accounting-Service to odap-accounting.

ExecCLIDRule

ExecCLIDRule is referenced from the policy engine to determine the authentication and authorization service and policy based on the CLID set in the policy engine.

ExecDNISRule

ExecDNISRule is referenced from the policy engine to determine the authentication and authorization service and policy based on the DNIS set in the policy engine.

ExecFilterRule

ExecFilterRule is referenced from the policy engine to determine whether a user packet should be rejected or not based on whether a special character like "*", "/", "\" or "?" shows up in the packet.

ExecNASIPRule

ExecNASIPRule is referenced from the policy engine to enable configuration of policies based on the incoming NAS-IP-Address. You can configure two attributes, *client-ip-address* and *subnetmask*, to match the incoming NAS-IP-Address and its subnet mask. If the attributes match, ExecNASIPRule sets the environment variables (if they are configured in that rule).

ExecRealmRule

ExecRealmRule is referenced from the policy engine to determine the authentication and authorization service and policy based on the realm set in the policy engine.

ExecTimeRule

ExecTimeRule either rejects or accepts Access Request packets based on the time range specified in a user's login profile. You can configure the TimeRange and AcceptedProfile attributes.

The format for the TimeRange is to set the allowable days followed by the allowable times, as in:

TimeRange = dateRange, timeRange

The dateRange can be in the form of a date, a range of allowable dates, a day, or a range of allowable days. The timeRange should be in the form of hh:mm-hh:mm.

Here are a few examples:

mon-fri,09:00-17:00

Allows access Monday through Friday from 9 AM until 5 PM.

mon,09:00-17:00;tue-sat,12:00-13:00

Allows access on Monday from 9 AM until 5 PM and from 12 noon until 1 PM on Tuesday through Saturday

mon,09:00-24:00;tue,00:00-06:00

Allows access on Monday from 9 AM until Tuesday at 6 AM

1-13,10-17:00; 15,00:00-24:00

Allows access from the first of the month until the thirteenth of the month from 10 AM until 5 PM and all day on the fifteenth of the month

LDAPOutage

LDAPOutage is referenced from LDAP Services as OutageScript. LDAPOutage logs when the LDAP binding is lost.

MapSourceIPAddress

MapSourceIPAddress is referenced from the CAR server's IncomingScript scripting point. MapSourceIPAddress checks to see if the request contains either a NAS-Identifier or a NAS-IP-Address. If not, this script sets the NAS-IP-Address from the request's source IP address.

The Tcl version of this script is tMapSourceIPAddress.

ParseAARealm

ParseAARealm is referenced from the NAS IncomingScript scripting point. It looks for a realm name on the user name attribute as a hint of which AAA service should be used for this request. If @<realm> is found, the AAA service is selected which has the same name as the realm.

ParseAAASRealm

ParseAAASRealm is referenced from the NAS incoming script extension point. ParseAAASRealm looks for a realm name on the user name attribute as a hint of which AAA service and which SessionManager should be used for this request. If @<realm> is found, the AAA service and SessionManager which have the same name as the realm are selected.

ParseAARealm

ParseAARealm is referenced from the NAS IncomingScript scripting point. It looks for a realm name on the user name attribute as a hint of which authentication and authorization service should be used for this request. If @<realm> is found, it selects the AA service that has the same name as the realm and the DefaultAccountingService (as specified in the configuration by the administrator).

The Tcl version of this script is named tParseAARealm.

ParseAASRealm

ParseAASRealm is referenced from the NAS IncomingScript scripting point. It looks for a realm name on the user name attribute as a hint of which AA service and which SessionManager should be used for this request. If @<realm> is found, the AA service and the SessionManager which have the same name as the realm are selected. The Accounting service will be the DefaultAccountingService (as specified in the configuration by the administrator).

The Tcl version of this script is named tParseAASRealm.

ParseProxyHints

ParseProxyHints is referenced from the NAS IncomingScript scripting point. It looks for a realm name on the user name attribute as a hint of which AAA services should be used for this request. If @radius is found, a set of AAA services is selected which will proxy the request to a remote radius server. If @tacacs is found, the AuthenticationService is selected that will proxy the request to a tacacs server for authentication. For any services not selected, the default service (as specified in the configuration by the administrator) will be used.

The Tcl version of this script is named tParseProxyHints.

ParseServiceAndAAALealmHints

ParseServiceAndAAALealmHints is referenced from the NAS IncomingScript scripting point. It calls both ParseServiceHints and ParseAAALealm.

The Tcl version of this script is named tParseServiceAndAAALealmHints.

ParseServiceAndAAASLealmHints

ParseServiceAndAAASLealmHints is referenced from the NAS IncomingScript scripting point. It calls both ParseServiceHints and ParseAAASLealm.

The Tcl version of this script is named tParseServiceAndAAASLealmHints.

ParseServiceAndAALealmHints

ParseServiceAndAALealmHints is referenced from the NAS IncomingScript scripting point. It calls both ParseServiceHints and ParseAALealm.

The Tcl version of this script is named tParseServiceAndAALealmHints.

ParseServiceAndAASLealmHints

ParseServiceAndAASLealmHints is referenced from the NAS IncomingScript scripting point. It calls both ParseServiceHints and ParseAASLealm.

The Tcl version of this script is named tParseServiceAndAASLealmHints.

ParseServiceAndProxyHints

ParseServiceAndProxyHints is referenced from the NAS IncomingScript scripting point. It calls both ParseServiceHints and ParseProxyHints.

The Tcl version of this script is named tParseServiceAndProxyHints.

ParseServiceHints

ParseServiceHints is referenced from the NAS IncomingScript scripting point. Check to see if we are given a hint of the service type or the realm. If so, set the appropriate attributes in the request or radius dictionary to record the hint and rewrite the user name to remove the hint.

The Tcl version of this script is named tParseServiceHints.

ParseTranslationGroupsByCLID

ParseTranslationGroupsByCLID is referenced from the policy engine to determine the incoming and outgoing translation groups based on CLID set in the policy engine so that the attributes can be added and/or filtered out by the configuration data set in MCD.

ParseTranslationGroupsByDNIS

ParseTranslationGroupsByDNIS is referenced from the policy engine to determine the incoming and outgoing translation groups based on realm set in the policy engine so that the attributes can be added/filtered out by the configuration data set in MCD.

ParseTranslationGroupsByRealm

ParseTranslationGroupsByRealm is referenced from the policy engine to determine the incoming and outgoing translation groups based on the realm set in the policy engine. ParseTranslationGroupsByRealm allows the attributes to be added or filtered out by the configuration data set in MCD.

UseCLIDAsSessionKey

UseCLIDAsSessionKey is used to specify that the Calling-Station-Id attribute should be used as the session key to correlate requests for the same session. This is a typical case for 3G mobile user session correlation.

USRIncomingScript

USRIncomingScript maps USR-proprietary attributes to CAR's global attribute space.

USRIncomingScript-IgnoreAccountingSignature

USRIncomingScript-IgnoreAccountingSignature maps USR-proprietary attributes to CAR's global attribute space and sets a flag to ignore the signature on Accounting-Request packets. Earlier versions of the USR RADIUS client did not correctly sign Accounting-Request packets.

USROutgoingScript

USROutgoingScript maps USR attributes from CAR's global attribute space to the appropriate USR-proprietary attributes.