

## Directing RADIUS Requests

---

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

You can use the policy engine to determine the AAA services for processing a request packet based on the User-Name suffix, User-Name prefix, Calling-Station-ID, Called-Station-ID and Nas-IP-Address. You configure the policy Engine through policies and rules.

A policy is a group of rules. Each rule consists of a set of conditions and corresponding services. A rule succeeds if all the conditions specified in the rule are satisfied. If a rule succeeds, the services indicated by its service attributes are used to process the packet. However, Cisco Access Registrar (CAR) defers packet processing until the policy succeeds.

This chapter contains the following sections:

- [Configuring Policies and Rules](#)
- [Routing Requests](#)
- [Standard Scripts Used with Rules](#)

## Configuring Policies and Rules

A policy is a group of rules. Each rule consists of a set of conditions and corresponding services. A rule succeeds if all the conditions specified in the rule are satisfied. If a rule succeeds, the services indicated by its service attributes are used to process the packet. However, CAR defers packet processing until the policy succeeds.

## Configuring Policies

You configure policies under **/Radius/Policies**. To enable the CAR server to use policies, you must first configure policy named SelectPolicy.

```
[ //localhost/Radius/Policies/SelectPolicy ]
  Name = SelectPolicy
  Description =
  Grouping = rule1|rule2
```

The Grouping property of a policy determines which rules are to be evaluated and in which order. Rules are evaluated from left to right. Use the pipe (|) or ampersand (&) character to group rules.

**Note**

Before you can provide rules in the Grouping property, the rules must first be added to the configuration under **/Radius/Rules**.

If rules are grouped with the pipe character (`rule1|rule2`), all rules in the group are evaluated in sequential order until one of the rules succeeds. If any one of the rules in the policy succeeds, the policy succeeds.

If rules are grouped with the ampersand character (`rule1&rule2&rule3`), all the rules listed are evaluated until one of the rules fails. For the policy to succeed, all the rules in the policy must succeed.

## Configuring Rules

You configure rules under **/Radius/Rules**. When you add a rule, provide the script that should be executed for the rule and the attributes to use if the rule succeeds. The script you specify must be defined under **/Radius/Scripts**, as shown in the following:

```
[ //localhost/Radius/Rules/rule1 ]
  Name = rule1
  Description =
  Script~ =
  Attributes/
    Authentication-service = local-users
    Authorization-service = local-users
    Realm = @cisco.com

[ //localhost/Radius/Scripts/ExecRealmRule ]
  Name = ExecRealmRule
  Description =
  Language = Rex
  Filename = librexscript.so
  EntryPoint = ExecRealmRule
  InitEntryPoint =
  InitEntryPointArgs =
```

## Wildcard Support

CAR supports limited wildcard functionality in rules for Realm, DNIS, and CLID attributes, specifically the asterisk (\*) and question mark (?) characters. The asterisk matches any number of characters, including the null character. The question mark matches any single character, not including the null character. CAR also supports both wildcard characters in one pattern, as in `CLID = 180098?87*`.

**Note**

The realms should start with either the @ or # character. For example, `Realm=@cisco.com`.

- For an exact matching of the realm, you should configure the rule with the exact realm. For example, for an exact match to `abc@cisco.com`, you should use `Realm=@cisco.com`.
- If you use `Realm=cisco.com` (without any valid character), values such as `xyz@us.cisco.com`, `xyz@uk.cisco.com`, `abc#cisco.com`, and so on can also match and return a success.

The following is an example using the asterisk wildcard character used in a Rule named *rule1*.

```
[ //localhost/Radius/Rules/rule1 ]
```

```
Name=rule1
Description =
ScriptName = ExecRealmRule
Attributes/
  Authentication-Service = Local-Users
  Authorization-Service = Local-Users
  Realm = ~/@*cisco.com/
```

Rule *rule1* succeeds when the domain of the user name in an access request matches the *@\*cisco.com* pattern. Each of the following is a good match: *@us.cisco.com*, *@eng.cisco.com*, and *@cisco.com*. With a match, the **ExecRealmRule** script sets Authentication-Service and Authorization-Service to Local-Users in the environment dictionary.

The following is an example using the "?" wildcard character in a Rule named *rule2*.

```
[ //localhost/Radius/Rules/rule2 ]
Name = rule2
Description =
ScriptName = ExecDNISRule
Attributes/
  Authentication-Service = Translation-Service
  Authorization-Service = Translation-Service
  DNIS = 1800345987?
```

Rule *rule2* succeeds if the Called-Station-Id attribute (DNIS) in the packet matches 1800345987?. Each of the following is a good match: 18003459876 and 18003459870, while 1800345987 is not. With a match, the **ExecDNISRule** script sets Authentication-Service and Authorization-Service to Translation-Service in the environment dictionary.

## Script and Attribute Requirements

The following script and attribute requirements exist:

- **/Radius/Policies/SelectPolicy** is the first policy CAR applies.
- The characters "|" and "&" are reserved as logical operands in a Grouping definition; they cannot be included in a **/Radius/Rules** name.
- A space is not permitted between the logical operands and the rules in a Grouping definition.
- The scripts included in the rules must be defined under the **/Radius/Scripts** directory.
- The attributes included in the rules must be defined under the **/Radius/Advanced/Attribute Dictionary** directory.

The rules included in the policies must be defined under the **/Radius/Rules** directory.

## Validation

When policies are configured, CAR performs the following validations:

- Ensures the scripts included in the rules are defined under the **/Radius/Scripts** directory.
- Ensures the attributes included in the rules are defined under the **/Radius/Advanced/Attribute Dictionary** directory.
- Ensures the rules included in the policies are defined under the **/Radius/Rule** directory.

## Known Anomalies

The following anomalies currently exist:

- Grouping expressions are not checked for validity.
- The use of parentheses to denote precedence is not supported in a Grouping definition.
- A check is not performed to determine whether a policy that is included within another policy is defined under the /Radius/Policies directory.

## Routing Requests

Using the policy engine, CAR enables you to route requests based on attributes in access request packets. The following sections describe how to route requests based on different attributes.

### Routing Requests Based on Realm

The CAR policy engine can process request packets based on the realm in the User-Name attribute.

In the following example, request packets with the User-Name attribute containing *@abc.com* as the suffix should be processed locally and the request packets with User-Name attribute containing *@xyz.com* should be proxied to a remote AAA server.

```
[ //localhost/Radius/Policies ]
  SelectPolicy/
    Name = SelectPolicy
    Description =
    Grouping = abcrule|xyzrule
```

The SelectPolicy refers to two rules, *abcrule* and *xyzrule*. When a request packet arrives, CAR executes SelectPolicy beginning with *abcrule* to determine if the User-Name attribute contains *@abc.com* as the realm. If so, the *abcrule* is successful as is SelectPolicy, therefore the packet is processed locally. If the User-Name attribute does not contain *@abc.com* as the realm, CAR executes *xyzrule* to determine if the User-Name attribute contains *@xyz.com*. If so, *xyzrule* is successful as is SelectPolicy. Hence the request is proxied to the remote server specified in *xyz-service*.

In this example, the rules are grouped using the | (or) operator. So all the rules specified in the grouping parameter will be executed until one of them succeeds.

```
[ //localhost/Radius/Rules ]
  abcrule/
    Name = abcrule
    Description =
    Script~ = ExecRealmRule
    Attributes/
      Authentication-Service = local-users
      Authorization-Service = local-users
      Realm = @abc.com

  xyzrule/
    Name = xyzrule
    Description =
    Script~ = ExecRealmRule
    Attributes/
      Authentication-Service = xyz-service
      Authorization-Service = xyz-service
      Realm = @xyz.com
```

The ExecRealmRule script matches the realm with the suffix in the User-Name attribute and sets the appropriate service for processing the packet. This is a standard script available with CAR. CAR can also be configured to set a particular kind of service for multiple realms. For example, the following configuration can be used if packets with *@pqr.com* or *@klm.com* should be processed using the same service klm-service.

```
[ //localhost/Radius/Rules ]
  rulex/
    Name = rulex
    Description =
    Script~ = ExecRealmRule
    Attributes/
      Authentication-Service = klm-service
      Authorization-Service = klm-service
      Realm = "@pqr.com" "@klm.com"
```

## Routing Requests Based on DNIS

The CAR policy engine can process request packets differently based on the DNIS (Called-Station-Id) attribute in the request packet.

In the following example, request packets with the Calling-Station-Id attribute that contain 1111111 should be processed by abc-service, while request packets with the Called-Station-Id attribute that contain 2222222 or 3333333 should be processed using xyz-service.

```
[ //localhost/Radius/Policies ]
  SelectPolicy/
    Name = SelectPolicy
    Description =
    Grouping = abcrule|xyzrule
```

The SelectPolicy refers to two rules, *abcrule* and *xyzrule*. When a request packet arrives, CAR executes SelectPolicy beginning with abcrule to determine if the DNIS attribute contains 1111111. If so, the abcrule is successful as is SelectPolicy, and the packet is processed using abc-service. If the Called-Station-Id attribute does not contain 1111111, CAR executes the xyzrule to determine if the Called-Station-Id attribute contains 2222222 or 3333333. If so, xyzrule is successful as is SelectPolicy, and the packet is processed using xyz-service.

```
[ //localhost/Radius/Rules ]
  abcrule/
    Name = abcrule
    Description =
    Script~ = ExecDNISRule
    Attributes/
      Authentication-Service = abc-service
      Authorization-Service = abc-service
      DNIS = 1111111

  xyzrule/
    Name = xyzrule
    Description =
    Script~ = ExecDNISRule
    Attributes/
      Authentication-Service = xyz-service
      Authorization-Service = xyz-service
      DNIS = "2222222" "3333333"
```

The **ExecDNISRule** script matches the DNIS value configured in CAR with the value in the Called-Station-Id attribute of the request packet and sets the appropriate service for processing the packet. **ExecDNISRule** is a standard script available with CAR.

## Routing Requests Based on CLID

The CAR policy engine can process request packets differently based on the CLID value in arriving request packets.

In the following example, the request packets with a Calling-Station-Id (CLID) attribute value of 1111111 should be processed by abc-service and the request packets with the CLID attribute value of 2222222 or 3333333 should be processed using xyz-service.

```
[ //localhost/Radius/Policies ]
  SelectPolicy/
    Name = SelectPolicy
    Description =
    Grouping = abcrule|xyzrule
```

The SelectPolicy refers to two rules, abcrule and xyzrule. When a request packet arrives, CAR executes SelectPolicy beginning with abcrule to determine if the CLID attribute contains 1111111. If so, the abcrule is successful as is SelectPolicy, and the packet is processed using abc-service. If the CLID attribute does not contain 1111111, CAR executes xyzrule to determine if the CLID attribute contains 2222222 or 3333333. If so, xyzrule is successful and hence SelectPolicy becomes successful and the packet is processed using xyz-service.

```
[ //localhost/Radius/Rules ]
  abcrule/
    Name = abcrule
    Description =
    Script~ = ExecCLIDRule
    Attributes/
      Authentication-Service = abc-service
      Authorization-Service = abc-service
      CLID = 1111111

  xyzrule/
    Name = xyzrule
    Description =
    Script~ = ExecCLIDRule
    Attributes/
      Authentication-Service = xyz-service
      Authorization-Service = xyz-service
      CLID = "2222222" "3333333"
```

The **ExecCLIDRule** script matches the CLID value configured in CAR with the value in the CLID attribute of the request packet and sets the appropriate service for processing the packet. **ExecCLIDRule** is a standard script available with CAR.

## Routing Requests Based on NASIP

The CAR policy engine can process request packets differently based on the client IP address value in arriving request packets.

In the following example, arriving request packets with the NAS-IP-Address attribute value 1.1.1.1 should be processed by abc-service and arriving request packets with the NAS-IP-Address attribute value 2.2.2.2 should be processed using xyz-service.

```
[ //localhost/Radius/Policies ]
  SelectPolicy/
    Name = SelectPolicy
    Description =
    Grouping = abcrule|xyzrule
```

The SelectPolicy refers to two rules, *abcrule* and *xyzrule*. When a request packet arrives, CAR executes SelectPolicy beginning with *abcrule* to determine if the NAS-IP-Address attribute contains an IP address from the subnet 1.1.1.0/24. If so, the *abcrule* is successful as is SelectPolicy, and the packet is processed using *abc-service*. If the NAS-IP-Address attribute does not contain an IP address from the subnet 1.1.1.0/24, CAR executes *xyzrule* to determine if the NAS-IP-Address attribute contains 2.2.2.2. If so, *xyzrule* is successful as is SelectPolicy, and the packet is processed using *xyz-service*.

```
[ //localhost/Radius/Rules ]
  abcrule/
    Name = abcrule
    Description =
    Script~ = ExecNASIPRule
    Attributes/
      Authentication-Service = abc-service
      Authorization-Service = abc-service
      Client-IP-Address = 1.1.1.0
      Subnet-mask = 255.255.255.0

  xyzrule/
    Name = xyzrule
    Description =
    Script~ = ExecNASIPRule
    Attributes/
      Authentication-Service = xyz-service
      Authorization-Service = xyz-service
      Client-IP-Address = 2.2.2.2
```

The **ExecNASIPRule** script matches the Client IP address configured in CAR with the value in the NAS-IP-Address attribute of the request packet and sets the appropriate service for processing the packet. **ExecNASIPRule** is a standard script available with CAR.

## Routing Requests Based on User-Name Prefix

You can use the CAR policy engine to select a service based on the prefix in the User-Name attribute.

In the following example, request packets with a User-Name attribute that contains *@abc.com* as the suffix and *cisco* as the prefix should be processed using the service *abc-service*. A request packet with User-Name attribute containing *cisco/bob@abc.com* will be processed using *abc-service*.

```
[ //localhost/Radius/Policies ]
  SelectPolicy/
    Name = SelectPolicy
    Description =
    Grouping = suffixrule & prefixrule
```

The SelectPolicy refers to two rules, *suffixrule* and *prefixrule*. When a request packet arrives, CAR executes SelectPolicy beginning with *suffixrule* to determine if the realm in the User-Name attribute contains *@abc.com*. If so, the *suffixrule* is successful. Since there is an “&” operator between the rules, the *prefixrule* must also succeed for the SelectPolicy to be successful. The *prefixrule* is now processed to determine if the prefix in the User-Name attribute contains *cisco*. If so, the *prefixrule* is successful which makes SelectPolicy successful, and the AA service is set to the service specified in the *prefixrule*.

```
[ //localhost/Radius/Rules ]
  abcrule/
```

```

Name = suffixrule
Description =
Script~ = ExecRealmRule
Attributes/
    Realm = @abc.com

prefixrule/
Name = prefixrule
Description =
Script~ = ExecPrefixRule
Attributes/
    Authentication-Service = abc-service
    Authorization-Service = abc-service
    Delimiters = @##%/
    Prefix = cisco
    StripPrefix = No

```

**ExecPrefixRule** script matches the prefix configured in CAR with the prefix in the User-Name attribute of the request packet and sets the appropriate service for processing the packet. **ExecPrefixRule** is a standard script available with CAR. See [ExecPrefixRule](#) for more information.

## Attribute Translation

The attribute translation feature supports the RADIUS proxy enabling you to customize attribute filters so that RADIUS attribute value (AV) pairs can be inserted, deleted, or substituted.

For example, when a request is proxied from AAA server on ISP A to AAA server on ISP B, some AV pairs might be substituted (such as IP address) because they might not be valid on the ISP B network. Additionally, ISP B might return some vendor-specific attributes (VSAs) that are not applicable to ISP A's network. Therefore, ISP A will substitute ISP B's VSA value pairs for ISP A's VSAs.

Two configuration points under the **/Radius** directory support this feature: **Translations** and **TranslationGroups**. Under the **/Radius/Translations** directory, any translation to insert, substitute, or translate attributes can be added. The following is a sample configuration under the **/Radius/Translations** directory:

```

[ //localhost/Radius/Translations/T1 ]
Name = T1
Description =
DeleteAttrs = Session-Timeout, Called-station-id
Attributes/
    Calling-Station-id = 1232909

```

**DeleteAttrs** is the set of attributes to be deleted from the packet. Each attribute is comma separated and no spaces are allowed between attributes.

Under the **/Radius/Translations/T1/Attributes** directory, the attributes that should be inserted and the attributes that should be substituted are specified. These AV pairs are either added to the packet if not present already or replaced with the configured value.

Under the **/Radius/TranslationGroups** directory, translations can be grouped and applied to certain sets of packets, which are referred to in a rule.

The following is a sample configuration under the **/Radius/TranslationGroups** directory:

```

[ //localhost/Radius/TranslationGroups/CiscoIncoming ]
Name = CiscoIncoming
Description =
Translations/
1. T1

```

The translation group is referenced through the CAR policy engine in the **/Radius/Rules/<RuleName>/Attributes** directory. Incoming-Translation-Groups are set to a translation group (for example CiscoIncoming) and Outgoing-Translation-Groups to another translation group (for example CiscoOutgoing).

The following is an example of setting up a rule for a translation group.

```
[ //localhost/Radius/Rules/ciscotranslationrule ]
  Name = ciscotranslationrule
  Description =
  Script~ = ParseTranslationGroupsByRealm
  Attributes/
    Incoming-Translation-Groups = CiscoIncoming
    Outgoing-Translation-Groups = CiscoOutgoing
    Realm = @cisco.com
```

The ciscoTranslationRule rule must be referred to in the **/Radius/Policies** directory, so the CAR policy engine can invoke this rule. If the pattern of Realm, DNIS, or CLID matches the one defined in the rule, CAR sets the environment variable Incoming-Translation-Groups to CiscoIncoming. By looking up the definition of CiscoIncoming, CAR applies all of the translations to the incoming packet (before it is proxied to the other server).

When the proxied packet comes back to the RADIUS server, CAR sets the environment variable, Outgoing-Translation-Groups to CiscoOutgoing.

DNIS, CLID, and Realm are supported for filtering packets. CAR provides the following scripts to facilitate filtering based on DNIS, CLID and Realm.

## Parsing Translation Groups

CAR provides three scripts that enable you to parse translation groups based on the DNIS, CLID or Realm attribute in an incoming packet. These scripts are:

- ParseTranslationGroupsByDNIS
- ParseTranslationGroupsByCLID
- ParseTranslationGroupsByRealm

In the following example, request packets containing @abc.com as the realm should be proxied to the remote server defined under abc-service. Before redirecting the request packet to the remote server, the Calling-Station-Id of the packet should be changed to 111.

```
[ //localhost/Radius/Policies ]
  SelectPolicy/
    Name = SelectPolicy
    Description =
    Grouping = realmrule & translaterule
```

The SelectPolicy refers to two rules, *realmrule* and *translaterule*. When a request packet arrives, CAR executes SelectPolicy beginning with “realmrule” to determine if the realm in the User-Name attribute contains 1.1.1.1. If so, the realmrule is successful and the AA service is set to abc-service. Next CAR executes the translaterule to change the CLID of the packet to 111.

```
[ //localhost/Radius/Rules/ciscotranslationrule ]
  Name = ciscotranslationrule
  Description =
  Script~ = ParseTranslationGroupsByRealm
  Attributes/
    Incoming-Translation-Groups = CiscoIncoming
    Realm = @cisco.com
```

```
[ //localhost/Radius/Translations ]
  Entries 1 to 1 from 1 total entries
  Current filter: <all>
  T1/
    Name = T1
    Description =
    Attributes/
      calling-station-id = 111

[ //localhost/Radius/TranslationGroups ]
  Entries 1 to 1 from 1 total entries
  Current filter: <all>
  CiscoIncoming/
    Name = CiscoIncoming
    Description =
    Translations/
      1. T1
```

## Time of Day Access Restrictions

You can use the CAR policy engine to implement access restriction on users based on the time of day. The **ExecTimeRule** script implements this functionality. **ExecTimeRule** can be used to check the time at which the request packet arrives and determine if access should be granted based on the authorization parameters for the user. If the rule succeeds, **ExecTimeRule** sets the Acceptedprofiles Environment dictionary variable to a profile or a set of profiles, as in the following:

```
Acceptedprofiles=Regularaccess::Highprivilegeaccess
```



### Note

If more than one profile is to be added to the Acceptedprofiles variable, use two colons to separate them (::).

If the user is authenticated, the Baseprofile of the user is compared with the Acceptedprofiles. All the profiles that are in the Baseprofile and in Acceptedprofiles will be used as profiles while sending the response for the user. For example, consider the following user configuration of user1:

```
[ //localhost/Radius/UserLists/new/user1 ]
  Name = user1
  Description =
  Password = <encrypted>
  AllowNullPassword = FALSE
  Enabled = TRUE
  Group~ = regularusers
  BaseProfile~ =highprivilegeaccess::readonlyaccess::regularaccess
  AuthenticationScript~ =
  AuthorizationScript~ =
  UserDefined1 =
  Attributes/
  CheckItems/
```

The Baseprofile of the user1 has highprivilegeaccess, readonlyaccess and regularaccess. If the Acceptedprofiles of the user has regularaccess and highprivilegeaccess, the profiles regularaccess and highprivilegeaccess will be used while sending the response packet.

## Setting Time Ranges in ExecTimeRule

ExecTimeRule accepts time range in the following format.

**Set timerange “\* \* \* \* \*”**

The first star indicates minutes and can be a value from 0-59. The second star indicates hours and can be a value from 0-23. The third star indicates day of the month and can be a value from 1-31. The fourth star indicates month and can be a value from 1-12. The fifth star indicates day of the week and can be a value from 0-6 where 0 indicates Sunday, 1 indicates Monday, and so on.

For example, to schedule a particular action to occur every Sunday during the month of December, use a command line like this:

**Set timerange “\* \* \* 12 0”**

## ExecTimeRule Example Configuration

This section provides a configuration example where a user, user1, is only authorized for PPP service between 10 AM and 6 PM. If a login occurs at any other time, user1 will be authorized only for telnet service.

### Policies

```
[ //localhost/Radius/Policies ]
  Entries 1 to 1 from 1 total entries
  Current filter: <all>
  SelectPolicy/
    Name = SelectPolicy
    Description =
    Grouping = ppprule|telnetrule
```

### Rules

```
[ //localhost/Radius/Rules ]
  Entries 1 to 2 from 2 total entries
  Current filter: <all>
  ppprule/
    Name = ppprule
    Description =
    Script~ = ExecTimeRule
  Attributes/
    acceptedprofiles = default-ppp-users
    timerange = "* 10-18 * * * *"
  telnetrule/
    Name = telnetrule
    Description =
    Script~ = ExecTimeRule
  Attributes/
    acceptedprofiles = default-telnet-users
    timerange = "* 0-10,18-23 * * * *"
```

### Profiles

```
[ //localhost/Radius/Profiles ]
  Entries 1 to 5 from 5 total entries
```

```

Current filter: <all>
default-PPP-users/
  Name = default-PPP-users
  Description =
  Attributes/
    Ascend-Idle-Limit = 1800
    Framed-Compression = "VJ TCP/IP header compression"
    Framed-MTU = 1500
    Framed-Protocol = PPP
    Framed-Routing = None
    Service-Type = Framed
default-Telnet-users/
  Name = default-Telnet-users
  Description =
  Attributes/
    Login-IP-Host = 204.253.96.3
    Login-Service = Telnet
    Login-TCP-Port = 541

```

## User

```

[ //localhost/Radius/UserLists/new/user1 ]
  Name = user1
  Description =
  Password = <encrypted>
  AllowNullPassword = FALSE
  Enabled = TRUE
  Group~ = regularusers
  BaseProfile~ = default-telnet-users::default-ppp-users
  AuthenticationScript~ =
  AuthorizationScript~ =
  UserDefined1 =
  Attributes/
  CheckItems/

```

## Reducing Overhead Using Policies to Group Rules

When you configure a large number of rules, the processing of request packets can be slow. For example, if you have 50 rules and only the last rule succeeds, the CAR server will have to check the preceding 49 rules before executing the rule that succeeds. You can reduce this overhead by using policies to group rules.

The following sample configuration, CAR must choose the AA service to be used for two domains, abc.com and xyz.com, based on the DNIS. You can do this by configuring different policies for different domains.

## Policies

In the following configuration, SelectPolicy selects the policy to process packets with realm abc.com or xyz.com. Based on the realm that arrives in the request packet, abcrealmrule and xyzrealmrule decide whether to use abc-policy or xyz-policy to process the packets. abc-policy and xyz-policy are configured with rules to check for DNIS numbers in the respective domains and set the AA services appropriately.

```

[ //localhost/Radius/Policies ]
  Entries 1 to 3 from 3 total entries
  Current filter: <all>
  SelectPolicy/
    Name = selectpolicy

```

```

        Description =
        Grouping = abcrealmrule|xyzrealmrule
abc-policy/
    Name = abc-policy
    Description =
    Grouping = abcDNISrule1|abcDNISrule2
xyz-policy/
    Name = xyz-policy
    Description =
    Grouping = xyzDNISrule1|xyzDNISrule2

```

## Rules

```

[ //localhost/Radius/Rules ]
Entries 1 to 6 from 6 total entries
Current filter: <all>

abcrealmrule/
    Name = abcrealmrule
    Description =
    Script~ = ExecRealmRule
    Attributes/
        policy = abc-policy
        realm = @abc.com
xyzrealmrule/
    Name = xyzrealmrule
    Description =
    Script~ = ExecRealmRule
    Attributes/
        policy = xyz-policy
        realm = @xyz.com
abcDNISrule1/
    Name = abcDNISrule1
    Description =
    Script~ = ExecDNISRule
    Attributes/
        Authentication-Service = abc1-service
        Authorization-Service = abc1-service
        DNIS = 1111111
abcDNISrule2/
    Name = abcDNISrule2
    Description =
    Script~ = ExecRealmRule
    Attributes/
        Authentication-Service = abc2-service
        Authorization-Service = abc2-service
        DNIS = 2222222
xyzDNISrule1/
    Name = xyzDNISrule1
    Description =
    Script~ = ExecRealmRule
    Attributes/
        Authentication-Service = xyz1-service
        Authorization-Service = xyz2-service
        DNIS = 6666666
xyzDNISrule2/
    Name = xyzDNISrule2
    Description =
    Script~ = ExecRealmRule
    Attributes/
        Authentication-Service = xyz2-service
        Authorization-Service = xyz2-service

```

```
DNIS = 7777777
```

## Standard Scripts Used with Rules

CAR software includes several scripts you can use with the rules. The following sections describe those scripts.

### ExecRealmRule

Use the **ExecRealmRule** script to determine the Authentication service and Authorization service to be used to process the request packet based on the suffix (Realm) in the User-Name attribute. You configure the Realm for which the packet should be checked and the service to use in the Attributes subdirectory of a rule. The **ExecRealmRule** script supports multi-valued attributes with which you can configure to check for multiple Realms.

For example, the following statement checks the request packet for three realms. If one of these three realms is found in the request packet, the **ExecRealmRule** script sets the attributes to the values listed in the Attributes subdirectory of the rule that references the **ExecRealmRule** script.

```
set Realm "@cisco.com" "@foo.com" "#bar.com"
```



#### Note

---

Only the characters @ and # can be used as delimiters in ExecRealmRule.

---

Prior to CAR 4.1.4, ExecRealmRule was interpreted as a regular expression pattern and was evaluated accordingly. As of CAR 4.1.4, ExecRealmRule now does a simple case insensitive comparison by default of the value specified for the realm attribute for the realm of a user name and optionally performs regular expression matching.

Beginning with the CAR 4.1.4 release, you can also specify a pattern using the following notation:

```
~/pattern/
```

Where pattern is a string of alpha-numeric characters that might include wild card characters, as in “@\*cisco.com” to match patterns (realms) that end in *cisco.com*.



#### Note

---

The question mark (?) should not be used without a character pattern preceding it. Specifying ? as the first character might have undesirable results. (For regexp terminology, the question mark should be preceded by an *atom*.)

---

The **ExecRealmRule** script checks the request packet for the Realm and applies the values set for the following attributes:

- Authentication-Service
- Authorization-Service
- Policy

## ExecDNISRule

Use the **ExecDNISRule** script to determine the Authentication service and Authorization service to be used to process the request packet based on the Called-Station-Id (DNIS) attribute. The DNIS for which the packet should be checked and the services can be configured through the Policy Engine. The **ExecDNISRule** script supports multi-valued attributes, by which you can configure multiple DNIS for checking.

For example, the following statement checks for a Calling-Station-Id of 1111111, 2222222, or 3333333. If one of the DNIS values is true, the script applies the values set for the Authentication-Service, Authorization-Service, and Policy attributes.

```
set DNIS "1111111" "2222222" "3333333"
```

## ExecCLIDRule

Use the **ExecCLIDRule** script with the Policy Engine to determine the Authentication service and Authorization service to be used to process the request packet based on the Calling-Station-Id (CLID) attribute. The CLID for which the packet should be checked and the services can be configured through the Policy Engine. **ExecCLIDRule** supports multi-valued attributes by which you can configure multiple CLID for checking.

For example, the following statement checks for Calling-Station-ID and applies Authentication-Service, Authorization-Service, and Policy.

```
set CLID "1111111" "2222222" "3333333"
```

The **ExecCLIDRule** script checks the request packet for the Calling-Station-Id and applies the values set for the following attributes:

- Authentication-Service
- Authorization-Service
- Policy

## ExecNASIPRule

The Policy Engine references the **ExecNASIPRule** script to determine the AAA Services, Policy and Session Manager based on the Client-IP-Address and Subnet-Mask set in the Policy Engine. The **ExecNASIPRule** script supports multi-value attributes by which multiple you can configure the Client-IP-Address and Subnet-Mask in **aregcmd** for checking.

For example, the following statements check for Client-IP-Address and Subnet-Mask and applies Authentication-Service, Authorization-Service, Accounting-Service, Policy, and Session-Manager.

```
set Client-IP-Address "1.1.1.1" "2.2.2.2" "3.3.3.3"
```

```
set Subnet-Mask "255.255.255.0" "255.255.0.0" "255.0.0.0"
```

The **ExecNASIPRule** script checks the request packet for the Client-IP-Address and Subnet-Mask and applies the values set for the following attributes:

- Authentication-Service
- Authorization-Service

- Accounting-Service
- Policy
- Session Manager

## ExecPrefixRule

The Policy Engine references the **ExecPrefixRule** to determine the authentication and authorization services based on the prefix in the User-Name attribute of the request packet and assigns the appropriate service for processing the packet.

Table 17-1 lists the **ExecPrefixRule** script attributes.

**Table 17-1** ExecPrefixRule Attributes

Attribute	Description
Delimiters	A list of valid delimiters; you can use any character as a delimiter, such as @#-/.
Prefix	List of valid prefixes.
StripPrefix	Option to strip or not to strip the prefix from the User-Name. If you configure this attribute to YES, the ExecPrefixRule strips the prefix from the User-Name. If you configure this attribute to NO, the ExecPrefixRule does not strip the prefix from the User-Name. By default, this attribute is set to YES.

For example, if cisco/bob@abc.com is the User-Name attribute, the **ExecPrefixRule** script sets the Authentication-Service to abc-service and User-Name to:

- bob@abc.com when the StripPrefix attribute is set to YES.
- cisco/bob@abc.com when the StripPrefix attribute is set to NO.

You can configure the Prefix attribute in CAR using the `aregcmd` as follows:

### set Prefix “cisco”

Beginning with the CAR 4.1.4 release, the CAR server does a case-insensitive comparison of the value specified for the prefix attribute of a user name.

Beginning with the CAR 4.1.4 release, you can configure the Prefix by specifying a pattern using the following notation:

```
~/pattern/
```

```
[ //localhost/Radius/Rules/prefix/Attributes ]
```

```
Delimiters = #@-/
```

```
Prefix = ~/cis*/
```

Where a pattern is a string of alpha-numeric characters that can include wild card characters, as in “cis\*” to match patterns (realms) that start with “cis”.

**Note**

If you specify / as the delimiter while configuring ExecPrefix Rule, you must configure the prefix as **Prefix =~/pattern//**.

**Note**

The question mark (?) should not be used without a character pattern preceding it. Specifying ? as the first character might have undesirable results. (For regexp terminology, the question mark should be preceded by an atom.)

## ExecSuffixRule

The Policy Engine references **ExecSuffixRule** to determine the AAA services, policy and session managers based on the suffix (or *realm*) set in the Policy Engine. You can use **aregcmd** to configure **ExecSuffixRule** to support multi-valued attributes, as in the following:

```
set Suffix "cisco.com" "abc.com" "domain.com"
```

In the User-Name *bob@abc.com*, **ExecSuffixRule** first checks for any of the configured delimiters in the User-Name. If there is a match, **ExecSuffixRule** checks for the configured suffix in the User-Name. If the suffix matches, **ExecSuffixRule** checks for the value of the StripSuffix variable. If StripSuffix is set to Yes, the suffix (including the delimiter) is stripped from the User-Name attribute of the Access Request.

Table 17-2 lists the **ExecSuffixRule** script attributes.

**Table 17-2** ExecSuffixRule Attributes

Attribute	Description
Delimiters	A list of valid delimiters; you can use any character as a delimiter such as these: @#/
Suffix	List of valid suffixes to scan
StripSuffix	The default value (No) does not strip the suffix from the User-Name. When set to Yes, <b>ExecSuffixRule</b> does strip the suffix.

Beginning with the CAR 4.1.4 release, the CAR server does a case-insensitive comparison of the value specified for the suffix attribute for the suffix of a user name.

Beginning with the CAR 4.1.4 release, you can also specify a pattern using the following notation:

```
~/pattern/
```

Where pattern is a string of alpha-numeric characters that might include wild card characters, as in "@\*cisco.com" to match patterns (realms) that end in *cisco.com*.

**Note**

The question mark (?) should not be used without a character pattern preceding it. Specifying ? as the first character might have undesirable results. (For regexp terminology, the question mark should be preceded by an atom.)

Consider the following sample configuration.

## Policies

You activate the Policy Engine by configuring SelectPolicy. Because the testsuffixrule is the only rule listed in Grouping, it is the only rule run.

```
[ //localhost/Radius/Policies/SelectPolicy ]
  Name = SelectPolicy
  Description =
  Grouping = testsuffixrule
```

## Rule

The testsuffixrule configuration does the following:

- points to the **ExecSuffixRule** script
- specifies the delimiters for which to scan
- specifies the suffixes for which to scan
- indicates whether to strip the suffix from the User-Name

```
[ //localhost/Radius/Rules/testsuffixrule ]
  Name = testsuffixrule
  Description =
  Script~ = ExecSuffixrule
  Attributes/
  Delimiters = @#/
  stripsuffix = yes
  suffix = domain.com
  suffix = ~/@??.cisco.com/
  suffix = ~/abc.*/
```

In this example, if *bob@abc.com* is the User-Name attribute, **ExecSuffixRule** strips the User-Name bob@abc.com and sets the User-Name environment variable to bob because StripSuffix is configured as *yes*.

## ExecTimeRule

Use the **ExecTimeRule** script to implement access restriction on users based on time. The **ExecTimeRule** script checks the time at which the request packet arrives and based on that the authorization parameters for the user can be decided. Based on the time of the request packet if the rule succeeds then **ExecTimeRule** sets the environment variable, Acceptedprofiles to a profile or a set of profiles.

For example, the following statement checks for Timerange and applies AcceptedProfiles.

```
Acceptedprofiles=Regularaccess::Highprivilegeaccess
```

## ParseTranslationGroupsByRealm

The Policy Engine references the ParseTranslationGroupsByReal script to determine the incoming and outgoing translation groups based on realm set in the Policy Engine. Use the ParseTranslationGroupsByReal script to add or filter attributes in request and response packets. The ParseTranslationGroupsByReal script supports multi-value attributes enabling you to configure to check for multiple Realms.

For instance, the following statement checks for three Realms. If True, the Policy Engine applies the values set for the Incoming-Translation-Group and Outgoing-Translation-Groups attributes.

```
set Realm "@cisco.com" "@foo.com" "@bar.com"
```

## ParseTranslationGroupsByDNIS

This script is referenced from the Policy Engine to determine the incoming and outgoing translation groups based on DNIS set in the Policy Engine. This script can be used to add/filter attributes in request/response packets. This script supports multi-value attributes, by which multiple DNIS can be configured for checking.

For example, the following statement checks for Calling-Station-ID and applies Incoming-Translation-Groups and Outgoing-Translation-Groups.

```
set DNIS "1111111" "2222222" "3333333"
```

## ParseTranslationGroupsByCLID

The Policy Engine references the ParseTranslationGroupsByCLID script to determine the incoming and outgoing translation groups based on CLID set in the Policy Engine. You can use the ParseTranslationGroupsByCLID script to add and filter attributes in request and response packets. The ParseTranslationGroupsByCLID script supports multi-value attributes, by which you can configure multiple CLIDs for checking.

For example, the following statement checks for the Calling-Station-ID and applies Incoming-Translation-Groups and Outgoing-Translation-Groups.

```
set CLID "1111111" "2222222" "3333333"
```

## ParseTranslationGroupsByDNIS

The **ParseTranslationGroupsByDNIS** script is referenced from the policy engine to determine the incoming and outgoing translation groups based on DNIS set in the policy engine. The **ParseTranslationGroupsByDNIS** script can be used to add and/or filter attributes in request and response packets. The **ParseTranslationGroupsByDNIS** script supports multi-value attributes, by which multiple DNIS can be configured for checking.

For example, the following statement checks for the Calling-Station-ID and applies Incoming-Translation-Groups and Outgoing-Translation-Groups.

```
set DNIS "1111111" "2222222" "3333333"
```

