

Route Maps

This chapter describes how to configure and customize route-maps, for ASA.

- About route maps, on page 1
- Guidelines for Route Maps, on page 3
- Define a Route Map, on page 3
- Customize a Route Map, on page 3
- Example for Route Maps, on page 6
- History for Route Maps, on page 6

About route maps

Route maps are used when redistributing routes into an OSPF, RIP, EIGRP or BGP routing process. They are also used when generating a default route into an OSPF routing process. A route map defines which of the routes from the specified routing protocol are allowed to be redistributed into the target routing process.

Route map characteristics

Route maps have many features in common with widely known ACLs. These are some of the traits common to both:

- They are an ordered sequence of individual statements, and each has a permit or deny result. Evaluation of an ACL or a route map consists of a list scan, in a predetermined order, and an evaluation of the criteria of each statement that matches. A list scan is aborted once the first statement match is found and an action associated with the statement match is performed.
- They are generic mechanisms. Criteria matches and match interpretation are dictated by the way that they are applied and the feature that uses them. The same route map applied to different features might be interpreted differently.

These are some of the differences between route maps and ACLs:

- Route maps are more flexible than ACLs and can verify routes based on criteria which ACLs can not verify. For example, a route map can verify if the type of route is internal.
- Each ACL ends with an implicit deny statement, by design convention. If the end of a route map is reached during matching attempts, the result depends on the specific application of the route map. Route maps that are applied to *redistribution* behave the same way as ACLs: if the route does not match any

clause in a route map then the route redistribution is denied, as if the route map contained a deny statement at the end.

Permit and deny clauses

Route maps can have permit and deny clauses. The deny clause rejects route matches from redistribution. You can use an ACL as the matching criterion in the route map.

Route map and ACL clause interaction rules

Because ACLs also have permit and deny clauses, the following rules apply when a packet matches the ACL:

- ACL permit + route map permit: routes are redistributed.
- ACL permit + route map deny: routes are not redistributed.
- ACL deny + route map permit or deny: the route map clause is not matched, and the next route-map clause is evaluated.

Match and set clause values

Each route map clause has two types of values:

- A match value selects routes to which this clause should be applied.
- A set value modifies information that will be redistributed into the target protocol.

Route map evaluation process

For each route that is being redistributed, the router first evaluates the match criteria of a clause in the route map. If the match criteria succeeds, then the route is redistributed or rejected as dictated by the permit or deny clause, and some of its attributes might be modified by the values set from the set commands. If the match criteria fail, then this clause is not applicable to the route, and the software proceeds to evaluate the route against the next clause in the route map. Scanning of the route map continues until a clause is found that matches the route or until the end of the route map is reached.

A match or set value in each clause can be missed or repeated several times, if one of these conditions exists:

- If several match entries are present in a clause, all must succeed for a given route in order for that route to match the clause (in other words, the logical AND algorithm is applied for multiple match commands).
- If a match entry refers to several objects in one entry, either of them should match (the logical OR algorithm is applied).
- If a match entry is not present, all routes match the clause.
- If a set entry is not present in a route map permit clause, then the route is redistributed without modification of its current attributes.



Note Do not configure a set entry in a route map deny clause because the deny clause prohibits route redistribution—there is no information to modify.

A route map clause without a match or set entry does perform an action. An empty permit clause allows a redistribution of the remaining routes without modification. An empty deny clause does not allow a redistribution of other routes (this is the default action if a route map is completely scanned, but no explicit match is found).

Guidelines for Route Maps

Firewall Mode

Supported only in routed firewall mode. Transparent firewall mode is not supported.

Additional Guidelines

Route maps do not support ACLs that include a user, user group, or fully qualified domain name objects.

Define a Route Map

You must define a route map when specifying which of the routes from the specified routing protocol are allowed to be redistributed into the target routing process.

Procedure

Create the route map entry:

route-map *name* {**permit** | **deny**} [*sequence_number*]

Example:

```
ciscoasa(config)# route-map name {permit} [12]
```

Route map entries are read in order. You can identify the order using the *sequence_number* argument, or the ASA uses the order in which you add route map entries.

Customize a Route Map

This section describes how to customize the route map.

Define a Route to Match a Specific Destination Address

Procedure

Step 1 Create the route map entry:

route-map *name* {**permit** | **deny**} [*sequence_number*]

Example:

```
ciscoasa(config)# route-map name {permit} [12]
```

Route map entries are read in order. You can identify the order using the *sequence_number* option, or the ASA uses the order in which you add route map entries.

Step 2 Match any routes that have a destination network that matches a standard ACL or prefix list:

match ip address {*acl_id* [*acl_id*] [...] | **prefix-list** *prefix_list_id* [*prefix_list_id*] [...]}

Example:

```
ciscoasa(config-route-map)# match ip address acl1 acl2 acl3
```

If you specify more than one ACL or prefix list, then the route can match any of the ACLs or prefix lists.

Step 3 Match any routes that have a specified metric:

match metric *metric_value*

Example:

```
ciscoasa(config-route-map)# match metric 200
```

The *metric_value* can range from 0 to 4294967295.

Step 4 Match any routes that have a next hop router address that matches a standard ACL:

match ip next-hop *acl_id* [*acl_id*] [...]

Example:

```
ciscoasa(config-route-map)# match ip next-hop acl2
```

If you specify more than one ACL, then the route can match any of the ACLs.

Step 5 Match any routes with the specified next hop interface:

match interface *if_name*

Example:

```
ciscoasa(config-route-map)# match interface if_name
```

If you specify more than one interface, then the route can match either interface.

Step 6 Match any routes that have been advertised by routers that match a standard ACL:

match ip route-source *acl_id* [*acl_id*] [...]

Example:

```
ciscoasa(config-route-map)# match ip route-source acl_id [acl_id] [...]
```

If you specify more than one ACL, then the route can match any of the ACLs.

Step 7 Match the route type:

match route-type {**internal** | **external** [**type-1** | **type-2**]}

Configure the Metric Values for a Route Action

If a route matches the **match** commands, then the following **set** commands determine the action to perform on the route before redistributing it.

To configure the metric value for a route action, perform the following steps:

Procedure

Step 1 Create the route map entry:

route-map *name* {**permit** | **deny**} [*sequence_number*]

Example:

```
ciscoasa(config)# route-map name {permit} [12]
```

Route map entries are read in order. You can identify the order using the *sequence_number* argument, or the ASA uses the order in which you add route map entries.

Step 2 Set the metric value for the route map:

set metric *metric_value*

Example:

```
ciscoasa(config-route-map)# set metric 200
```

The *metric_value* argument can range from 0 to 294967295.

Step 3 Set the metric type for the route map:

set metric-type {**type-1** | **type-2**}

Example:

```
ciscoasa(config-route-map)# set metric-type type-2
```

The *metric-type* argument can be type-1 or type-2.

Example for Route Maps

The following example shows how to redistribute routes with a hop count equal to 1 into OSPF.

The ASA redistributes these routes as external LSAs with a metric of 5 and a metric type of Type 1.

```
ciscoasa(config)# route-map 1-to-2 permit
ciscoasa(config-route-map)# match metric 1
ciscoasa(config-route-map)# set metric 5
ciscoasa(config-route-map)# set metric-type type-1
```

The following example shows how to redistribute the 10.1.1.0 static route into eigrp process 1 with the configured metric value:

```
ciscoasa(config)# route outside 10.1.1.0 255.255.255.0 192.168.1.1
ciscoasa(config-route-map)# access-list mymap2 line 1 permit 10.1.1.0 255.255.255.0
ciscoasa(config-route-map)# route-map mymap2 permit 10
ciscoasa(config-route-map)# match ip address mymap2
ciscoasa(config-route-map)# router eigrp 1
ciscoasa(config-router)# redistribute static metric 250 250 1 1 1 route-map mymap2
```

History for Route Maps

Table 1: Feature History for Route Maps

Feature Name	Platform Releases	Feature Information
Route maps	7.0(1)	We introduced this feature. We introduced the following command: **route-map**.
Enhanced support for static and dynamic route maps	8.0(2)	Enhanced support for dynamic and static route maps was added.
Support for Stateful Failover of dynamic routing protocols (EIGRP, OSPF, and RIP) and debugging of general routing-related operations	8.4(1)	We introduced the following commands: **debug route**, **show debug route**. We modified the following command: **show route**.
Dynamic Routing in Multiple Context Mode	9.0(1)	Route maps are supported in multiple context mode.
Support for BGP	9.2(1)	We introduced this feature. We introduced the following commands: **router bgp**
IPv6 support for Prefix Rule	9.3.2	We introduced this feature.