Chapter 7
Routing Information Protocol (RIP)

authentication

Name

`authentication` - sets the authentication used on an interface

Syntax

```
[ secondary ] authentication none ;
[ secondary ] authentication simple password ;
[ secondary ] authentication md5 password ;
[ secondary ] authentication md5 key password id number [ {
    [ start-accept YYYY/MM/DD HH:MM ] ;
    [ stop-accept YYYY/MM/DD HH:MM ] ;
    [ start-generate YYYY/MM/DD HH:MM ] ;
    [ stop-generate YYYY/MM/DD HH:MM ] ;
} ; ]
```

Parameters

`password` - a 4-byte, period-separated decimal number (0.0.0.0 to 255.255.255.255), or a 1- to 8-byte hexadecimal number (0x0 to 0xffffffff), or from one to eight characters in double quotes (for example, “a” or “Whoever#”)

`start-accept` *`YYYY/MM/DD HH:MM`* - a time specification for the start accept time for this key

`stop-accept` *`YYYY/MM/DD HH:MM`* - a time specification for the stop accept time for this key

`start-generate` *`YYYY/MM/DD HH:MM`* - a time specification for the start generate time for this key

`stop-generate` *`YYYY/MM/DD HH:MM`* - a time specification for the stop generate time for this key

`number` - the key ID to be used with this key

Description

The authentication command is used both to verify and to generate the authentication field in the RIP header for all packets sent and received on the specified interface. This applies only to version 2 packets and RIPv1-compatible RIPv2 packets, because RIPv1 does not support authentication. One exception to the above is the case of query packets. In the case of querying a router through an interface that is speaking RIPv1, the query packet will still be authenticated, but no authentication will be used on the outgoing packet.

If a packet is received with authentication that does not match (including the case where a packet is received with authentication when none is specified on the interface), then it is ignored.

On a trusted network, simple authentication can be used to create two logical networks because sets of routers with shared passwords will talk to each other, but not communicate with those using a different password. On an untrusted network, however, this technique should not be used because simple passwords are sent in clear text. MD5 should be used instead. However, even MD5 does not encrypt the entire packet, only the authentication field of the header.

Specifying anything other than authentication **`none`** and not specifying **`version 2`** will cause a parse error. For somewhat more serious authentication, use **`trustedgateways`** or a combination of **`trustedgateways`** and **`authentication`**.

Defaults

```
authentication none ;
```

Context

`rip interface` statement

Examples

Example 1

The following command resets the default behavior.

```
rip on {
    interface all authentication none;
};
```

Example 2

To set up two virtual RIP networks, one running on interfaces eth0 and eth1 and one running on eth2 and eth3, something like the following can be used.

```
rip on {
    interface eth0 eth1 version 2
        authentication simple “foo”;
    interface eth2 eth3 version 2
         authentication simple “bar”;
};
```

Example 3

On a network where users are not trusted, MD5 authentication can be used. The following example sets MD5 keys for a month, each overlapping for an hour.

```
    rip on {
        interface all version 2 authentication md5 {
            key “KeepOut” id 1 {
                start-accept 2000/05/01 00:00;
                stop-accept 2000/05/08 01:00;
            };
            key 192.168.10.1 id 2 {
                start-accept 2000/05/08 00:00;
                stop-accept 2000/05/16 01:00;
            };
            key 0xff id 3 {
                start-accept 2000/05/16 00:00;
                stop-accept 2000/05/23 01:00;
            };
            key “TheEnd” id 4 {
                start-accept 2000/05/23 00:00;
                stop-accept 2000/06/01 01:00;
            };
        };
    };
```

Example 4

There is one point of potential confusion in Example 3, in that a semicolon is not necessary after the braces surrounding the MD5 keys, but it is necessary after `interface`. So, the following gives results identical to Example 3.

```
    rip on {
        interface all authentication md5 {
            key “KeepOut” id 1 {
                start-accept 2000/05/01 00:00;
                stop-accept 2000/05/08 01:00;
            };
            key 192.168.10.1 id 2 {
                start-accept 2000/05/08 00:00;
                stop-accept 2000/05/16 01:00;
            };
            key 0xff id 3 {
                    start-accept 2000/05/16 00:00;
                    stop-accept 2000/05/23 01:00;
                };
                key "TheEnd" id 4 {
                    start-accept 2000/05/23 00:00;
                    stop-accept 2000/06/01 01:00;
                };
            } version 2;
        };
```

See Also

`ripin` on page 70

`ripout` on page 72

`metricin` on page 60

`metricout` on page 62

`version` on page 83

`secondary authentication` on page 74

`sourcegateways` on page 76

`trustedgateways` on page 80

`interface` on page 58

broadcast, nobroadcast

Name

`broadcast`, **`nobroadcast`** - specifies whether RIP is an active or passive participant

Syntax

```
broadcast;
nobroadcast;
```

Parameters

none

Description

RIP is most often run on a multi-homed box to cause the machine to act as a router in small LANs; however, it is also run on single-homed machines, often to learn the default route or routes to other networks in the domain. RIP will try to guess whether it should act as a router (i.e., be an active participant in routing) by examining the number of interfaces available to it. **`[no]broadcast`** can be used to override this default behavior. Specifically, if **`broadcast`** is specified, then RIP updates will be sent even if only one interface is present; conversely, if **`nobroadcast`** is specified, RIP updates will not be sent, regardless of the number of interfaces present.

`[no]broadcast` is most often used in its nobroadcast form to turn off active participation in a RIP cloud. This can be used, for example, on a multi-homed machine running different protocols on different interfaces where it is desired not to send the other attached networks or learned routes into the RIP cloud. The same functionality can be achieved through policy rules. But for a simple network, that degree of configuration complexity is not necessary.

Either version of **`[no]broadcast`** applies only to broadcasting (or multicasting, in the case of RIP version 2) of RIP packets. Therefore, it will lack effect if **`sourcegateways`** is present.

`broadcast` should not be confused with the broadcast argument to **`version`**.

Defaults

RIP will default to broadcasting if configured **on** and if there is more than one attached interface. It will default to only listening if there is just one attached interface.

Context

`rip` statement

Examples

Example 1

The following example configures RIP to broadcast updates, even on a single-homed machine. This might be useful for exporting static routes that are configured only on the single-homed machine; however, in some cases, use of broadcast on a single-homed machine can result in data packets traversing a single network twice.

```
rip on {
    broadcast;
};
```

Example 2

The following example turns broadcasting off, even on a multi-homed machine.

```
rip on {
    nobroadcast;
};
```

See Also

`export` on page 623

`import` on page 605

`sourcegateways` on page 76

defaultmetric

Name

`defaultmetric` - sets the default metric for advertising RIP routes learned from other protocols

Syntax

`defaultmetric` *`defmetnum`***`;`**

Parameters

`defmetnum` - a RIP metric from 1 to 16, inclusive

Description

When RIP learns a route from another protocol, the metrics of the two protocols are almost certainly incompatible. Most protocols have a much larger metric space (for example, 0 to 65535) than RIP does, and, therefore, a mechanism is required for specifying the initial metric that the RIP advertiser will use.

`defaultmetric` is used to set the default metric for advertising into the RIP cloud routes learned from other protocols, including **`static`**. Use the **`export`** statement to set this metric for exporting routes to RIP on a per-protocol basis.

Defaults

`defaultmetric 1;`

Context

`rip` statement

Examples

The following example sets the default metric for routes exported to RIP to be 2.

```
rip on {
    defaultmetric 2;
};
```

See Also

`export` page 623

expire-time

Name

`expire-time` - sets the expiration time for routes received via RIP

Syntax

`expire-time` *`expire_time`***;**

Parameters

`expire_time` - an integer from 5 to 180, inclusive

Description

`expire-time` is initialized when a route is established, and any time an update message is received for the route. If *`expire_time`* seconds elapse from the last time the **`expire-time`** was initialized, the route is considered to have expired, and the deletion process begins for that route. This is a **`group`** option in the **`rip`** clause. For example, it is used in the same context as **`preference`** and **`broadcast`**.

Defaults

`expire-time 180;`

Context

`rip` statement

Examples

```
rip yes {
     expire-time 100;
     interface fxp0;
};
```

See Also

`update-time` on page 82

ignorehostroutes

Name

`ignorehostroutes` - ignores host routes learned in route responses

Syntax

```
ignorehostroutes;
```

Parameters

none

Description

Sometimes it is desirable to ignore host routes (routes with a netmask of 255.255.255.255) advertised by other routes.

Note: This is effective only with RIP version 2. (RIP version 1 does not send netmask information in its updates.)

Context

`rip` statement

Examples

```
    rip on {
         interface eth0 version 2;
         ignorehostroutes;
    };
```

See Also

`version` on page 83

`interface` on page 58

interface

Name

`interface` - specifies interface(s) on which RIP operates

Syntax

`interface` *`interfacelist`* **`[`***`ripIFcmd`* **`[`***`ripIFcmd`* **`... ] ];`**

Parameters

`interfacelist` - the list of interfaces on which RIP will operate with the given *`ripIFcmd`*s

Description

`interface` is used both to set interface-specific parameters in RIP and to determine on which interfaces RIP will be run. By default, RIP will run on all interfaces; however, if any specific interfaces (or subsets of interfaces) are explicitly configured with **`interface`**, then all non-configured interfaces will not run RIP.

If multiple interfaces (either physical or logical) have addresses on the same subnet, then (regardless of configuration) RIP will send updates only on the first one for which RIP is configured to do so.

Although it is possible to specify a loopback interface or loopback address in an interface statement, RIP will not normally send packets to a loopback. To override this behavior, use **`sourcegateways`** with the loopback address included in the *`gatewaylist`*.

Defaults

```
interface all;
```

Context

`rip` statement

Examples

Example 1

The following will configure RIP to run on the eth0 interface only.

```
rip on {
    interface eth0;
};
```

Example 2

Alternatively, the following example will set RIP to run on all interfaces except eth0.

```
rip on {
    interface all;
```

```
          interface eth0 noripin noripout;
      };
```

See Also

ripin on page 70

ripout on page 72

metricin on page 60

metricout on page 62

version on page 83

authentication on page 49

secondary authentication on page 74

sourcegateways on page 76

metricin

Name

`metricin` - sets an additional metric on incoming RIP routes

Syntax

`metricin` *`ripmetric`*

Parameters

`ripmetric` - a number signifying hop count, from 0 to 15

Description

It is often the case that a router should prefer routes received on one set of interfaces over those received on another. For example, given two point-to-point links, one can be more expensive than the other and should, therefore, be less preferred. **`metricin`** is used for exactly this purpose: to make routes learned from certain interfaces less preferable.

`metricin` is the default manner by which RIP increments hop count. That is to say, RIP works by adding a hop every time a route is received and before it is sent back out to other interfaces. This implementation adds the hop when the route is received (for example, before decisions regarding whether the route should be used are made). By default, a metric of 1 plus the kernel interface metric is added as the hop count. Normally, this interface metric is zero, but some operating systems allow it to be specified on interface configuration. If **`metricin`** is explicitly given, it is added as an absolute value (for example, without the interface metric).

Defaults

If left unspecified, a metric of 1 plus the kernel interface metric (if any) is the default.

Context

`rip interface` statement

Examples

Example 1

To override any specified interface metric, the following adds exactly 1 to the metric of all received routes.

```
rip on {
    interface all metricin 1;
};
```

Example 2

In somewhat more normal usage, the following example increases the cost of routes by 2 that are received over a point-to-point link more than those received on other interfaces.

```
rip on {
    interface all metricin 1;
    interface ppp0 metricin 3;
};
```

See Also

`ripin` on page 70

`ripout` on page 72

`metricout` on page 62

`version` on page 83

`authentication` on page 49

`secondary authentication` on page 74

`sourcegateways` on page 76

`interface` on page 58

metricout

Name

metricout - specifies an additional cost to be added to outgoing routes

Syntax

metricout *ripmetric*

Parameters

ripmetric - a number signifying hop count, from 0 to 15

Description

Normally, this RIP implementation adds to the hop count only on incoming routes. There are times, however, when the user wants to cause other routers not to prefer routes from a given origin. For example, if the router is a backup router, it might be desirable for its routes to always be less preferred. **metricout** accomplishes this by adding to the RIP metric on top of any metric specified by **metricin** before RIP updates are sent out the specified interface.

Defaults

metricout 0 ;

Context

rip interface statement

Examples

Example 1

If this router (host 128) is acting as backup on 192.168.10/24 for all other interfaces, the following will add 2 to all metrics advertised out the 192.168.10.128 interface.

```
rip on {
    interface all ripin ripout;
    interface 192.168.10.128 metricout 2;
};
```

Example 2

The following example has no effect. It is contrived to point out that **metricout** does not impact routing based on the receipt of updates, so if updates do not go out an interface on which **metricout** is specified, the behavior of the entire network will be identical to what it would have been without **metricout** being issued.

```
rip on {
    interface all ripin ripout;
    interface eth0 ripin noripout metricout 15;
    };
```

See Also

`ripin` on page 70

`ripout` on page 72

`metricin` on page 60

`version` on page 83

`authentication` on page 49

`secondary authentication` on page 74

`sourcegateways` on page 76

`interface` on page 58

nocheckzero

Name

`nocheckzero` - specifies handling of zero-filled reserved fields in RIP version 1

Syntax

```
nocheckzero ;
```

Parameters

none

Description

The RIP version 1 specification mandates that certain fields in RIP routing updates are reserved and should be filled with zeros. It also mandates that version 1 packets, where the reserved fields are not filled with all zeros, should be discarded. However, some implementations of RIP do not follow the specification and carry non-zero information in these fields. **`nocheckzero`** allows RIP to interoperate with those broken implementations.

The user should be certain that the noncompliant router is intentionally generating malformed packets (rather than malfunctioning) before enabling interoperability. Otherwise, correct routing could be compromised.

Defaults

Zero-filled reserved fields are checked for RIP version 1.

Context

`rip` statement

Examples

The following example enables interoperability with a router that is not filling reserved fields with zeros:

```
rip on {
    nocheckzero;
};
```

preference

Name

preference - sets the preference for RIP routes

Syntax

preference *prefvalue* **;**

Parameters

prefvalue - number from 0 to 255

Description

Routers can learn multiple routes to the same destination. Each routing protocol handles this internally, so, for example, if RIP learns two different routes to the same destination, it will select the route with the lower metric. However, a router can learn two routes to the same destination from different routing protocols, and it must find a way to choose between them. GateD uses a preference number to make this decision, choosing the route with the lower preference. Normally, RIP routes are preferred to routes from external routing protocols (for example, BGP or OSPF ASE) and to those that are generated or aggregated. RIP routes are less preferable than those learned from other intra-domain routing protocols and those that are statically configured. **preference** can be used to change this ordering.

Defaults

```
preference 100;
```

Context

rip statement

Examples

Example 1

The following example makes RIP routes more preferable than IS-IS routes, but less preferable than OSPF routes.

```
rip on {
    preference 13;
};
```

Example 2

The following example makes RIP routes less preferable than BGP routes, but still more preferable than EGP routes.

```
rip on {
    preference 180;
    };
```

See Also

"Preferences and Route Selection" on page 11 of *Configuring GateD*

"Route Importation" on page 137 of *Configuring GateD*

`import` on page 605

query authentication

Name

`query authentication` - sets the authentication used by the **`ripquery`** utility

Syntax

```
query authentication none ;
query authentication simple password ;
query authentication md5 password ;
query authentication md5 key password id number [ {
    [ start-accept YYYY/MM/DD HH:MM ] ;
    [ stop-accept YYYY/MM/DD HH:MM ] ;
    [ start-generate YYYY/MM/DD HH:MM ] ;
    [ stop-generate YYYY/MM/DD HH:MM ] ;
} ; ]
```

Parameters

password - a 4-byte, period-separated decimal number (0.0.0.0 to 255.255.255.255), or a 1- to 8-byte hexadecimal number (0x0 to 0xffffffff), or from one to eight characters in double quotes (for example, “a” or “Whoever#”)

start-accept *YYYY/MM/DD HH:MM* **-** a time specification for the start accept time for this key

stop-accept *YYYY/MM/DD HH:MM* **-** a time specification for the stop accept time for this key

start-generate *YYYY/MM/DD HH:MM* **-** a time specification for the start generate time for this key

stop-generate *YYYY/MM/DD HH:MM* **-** a time specification for the stop generate time for this key

number - the key ID to be used with this key

Description

Certain utilities, such as **`ripquery`**, have been created for debugging RIP routers. These tools send a RIP POLL packet, which is an extension undocumented in the RFC.

`query authentication` is used to check the incoming POLL packets. If the authentication matches, then RIP will reply with its full routing table. (For example, it will not run split-horizon or poison reverse before replying.) If the authentication does not match, then the request will be discarded.

Although **`query authentication`** uses the standard key format for passwords, the generate portions of the key are irrelevant because the key is used only to check incoming requests. Outgoing packets use whatever authentication method is set up on the interface over which the POLL packet was received.

Defaults

`query authentication none;`

Context

`rip` statement

Examples

The following example sets a simple password for authentication of RIP POLL packets:

```
rip on {
    query authentication simple 0xdeadbeef;
};
```

See Also

`authentication` on page 49

`secondary authentication` on page 74

“**`ripquery`**” on page 17 of *Operating GateD*

rip

Name

rip - configures the Routing Information Protocol (RIP)

Syntax

```
rip on ;
rip on { ripargs } ;
rip off ;
```

Parameters

ripargs - RIP configuration parameters, which are described throughout this chapter

Description

One of the most widely used interior gateway protocols is RIP. RIP is an implementation of a distance-vector routing protocol (using the Bellman-Ford algorithm) for local networks. RIP classifies routers as active or passive (silent). Active routers advertise their routes (reachability information) to others; passive routers listen and update their routes based on advertisements, but do not advertise. Typically, routers run RIP in active mode, while hosts use passive mode.

Defaults

The default for RIP is **off** unless the **--enable-ripon** flag is passed to **configure**.

Context

global

Examples

The following example configures RIP to run on all interfaces.

```
rip on;
```

See Also

bgp on page 232

ospf on page 124

isis on page 185

dvmrp on page 364

import on page 605

export on page 623

aggregate on page 673

dampen-flap on page 707

“Routing Information Protocol” on page 39 of *Configuring GateD*

ripin, noripin

Name

`ripin, noripin` - specifies whether RIP will listen to RIP updates

Syntax

`ripin`

`noripin`

Parameters

none

Description

`ripin` specifies that RIP will process RIP updates received on a given interface. Since this is also the default, it is not normally required, except to override a **`noripin`** specified on a wildcard list of interfaces. **`noripin`** does exactly the opposite. Although it would almost certainly be a misconfiguration, it is important to note that RIP can send RIP updates on a superset of those interfaces on which it receives updates. This can be a valid configuration if, for example, the user receives RIP updates from an ISP, redistributing those onto the LAN, and does not want to send the LAN topology back to the ISP. But this would be highly unusual.

Defaults

`ripin`

Context

`rip interface` statement

Examples

This somewhat contrived example processes RIP updates received on all eth devices except eth0.

```
rip on {
    interface eth ripin;
    interface eth0 noripin;
};
```

See Also

`ripout` on page 72

`metricin` on page 60

`metricout` on page 62

`version` on page 83

`authentication` on page 49

`secondary authentication` on page 74

`sourcegateways` on page 76

`interface` on page 58

ripout, noripout

Name

`ripout, noripout` - specifies whether RIP will send RIP updates

Syntax

`ripout`

`noripout`

Parameters

none

Description

`noripout` specifies that RIP updates should not be sent on the specified interfaces. **`ripout`** specifies the converse and is the default on broadcast-enabled interfaces.

Defaults

`ripout` on broadcast interfaces

`noripout` on nonbroadcast interfaces (including point-to-point interfaces)

Context

`rip interface` statement

Examples

Example 1

The following example specifies that RIP updates should not be sent on any eth interfaces except eth0.

```
rip on {
    interface eth noripout;
    interface eth0 ripout;
};
```

Example 2

The following example specifies that RIP updates should be sent on all eth and ppp interfaces.

```
rip on {
    interface eth ripout;
    interface ppp ripout;
};
```

See Also

`ripin` on page 70

`metricin` on page 60

`metricout` on page 62

`version` on page 83

`authentication` on page 49

`secondary authentication` on page 74

`sourcegateways` on page 76

`interface` on page 58

secondary authentication

Name

`secondary authentication` - sets the authentication used on an interface

Syntax

```
[ secondary ] authentication none ;
[ secondary ] authentication simple password ;
[ secondary ] authentication md5 password ;
[ secondary ] authentication md5 key password id number [ {
    [ start-accept YYYY/MM/DD HH:MM ] ;
    [ stop-accept YYYY/MM/DD HH:MM ] ;
    [ start-generate YYYY/MM/DD HH:MM ] ;
    [ stop-generate YYYY/MM/DD HH:MM ] ;
} ; ]
```

Parameters

password - a 4-byte, period-separated decimal number (0.0.0.0 to 255.255.255.255), or a 1- to 8-byte hexadecimal number (0x0 to 0xffffffff), or from one to eight characters in double quotes (for example, “a” or “Whoever#”)

`start-accept` *YYYY/MM/DD HH:MM* **-** a time specification for the start accept time for this key

`stop-accept` *YYYY/MM/DD HH:MM* **-** a time specification for the stop accept time for this key

`start-generate` *YYYY/MM/DD HH:MM* **-** a time specification for the start generate time for this key

`stop-generate` *YYYY/MM/DD HH:MM* **-** a time specification for the stop generate time for this key

number - the key ID to be used with this key

Description

`secondary authentication` is identical in functionality to **`authentication`**, with the exception that it is used only for checking authentication and is used only if the primary authentication method fails.

`secondary authentication` can be used while a network is in transition to make sure that old passwords are still accepted.

Defaults

`secondary authentication none ;`

Context

`rip interface` statement

Examples

In order to transition a network from simple password “foo” to simple password “bar,” something like the following can be used until all routers are updated to the new password.

```
rip on {
    interface all version 2 authentication simple “foo”
        secondary authentication simple “bar”;
};
```

See Also

ripin on page 70

ripout on page 72

metricin on page 60

metricout on page 62

version on page 83

authentication on page 49

sourcegateways on page 76

trustedgateways on page 80

interface on page 58

sourcegateways

Name

`sourcegateways` - sets routers to which RIP will send updates

Syntax

`sourcegateways` *`gatewaylist`*`;`

Parameters

`gatewaylist` - a comma-separated list of IP addresses or host names

Description

`sourcegateways` provides a complete list of IP addresses to which the configured router will unicast RIP updates.

RIP must be configured on an interface that shares a subnet with each source gateway.

If any source gateways are specified, then no RIP packets will be broadcast or multicast. They will only be unicast to the list of source gateways. As a result, the arguments to **`version`** do not exactly match their denotation. If version 1 is specified, v1 packets will be unicast to the list of sourcegateways. If version 2 is specified, and the argument to it is broadcast, v1 compatible, then v2 packets will be unicast. If version 2 is specified and the argument to it is multicast, then v2 packets will be unicast to the list of sourcegateways. If no sourcegateways are specified, GateD will not unicast updates.

Defaults

By default, RIP will either broadcast or multicast its updates.

Context

`rip` statement

Examples

Example 1

This example sets a RIP router to send updates to routers with only host addresses of 2 on the 192.168.10 and .12 networks.

```
rip on {
    interface eth;
    sourcegateways 192.168.10.2, 192.168.12.2;
};
```

Example 2

Similarly, the following example shows how this can be accomplished with host names (but DNS must be relied upon).

```
rip on {
    interface eth;
    trustedgateways rtr1.my.domain, rtr2.my.domain;
};
```

See Also

`version` on page 83

`interface` on page 58

traceoptions

Name

`traceoptions` - sets RIP-specific tracing options

Syntax

`traceoptions` *`trace_options`* **`;`**

Parameters

`trace_options` - In addition to the trace options specified in "Chapter 4 Trace Statements" on page 15, the following trace options are available for the RIP protocol:

`request` - Trace RIP information request packets, which include request, poll, and poll entry packets.

`response` - Trace RIP response packets (for example, those that actually contain routing updates).

`other` - Trace any other type of RIP packet.

`packets` - Trace all of the above.

or one of these packet tracing options, optionally prepended by one of the three modifiers (**`send`**, **`recv`**, **`detail`**):

Description

RIP tracing options behave similarly to all other tracing options. The RIP-specific packet tracing options available include: **`request`**, **`response`**, **`other`**, and **`packets`**.

Each of the above can be modified by **`detail`** (include more details than normal tracing), **`send`** (trace only those packets that this router sends), or **`recv`** (trace only those packets that this router receives).

In addition, the trace option of **`policy`** will trace whenever a new route is announced, when a metric being announced is changed, or when a route enters or exits holddown. The trace option of **`all`** traces all of the packets. **`none`** traces nothing.

Defaults

```
traceoptions none;
```

Context

`rip` statement

Examples

Example 1

This example saves detail on all packets sent or received to the file `/var/tmp/rip_pkts`.

```
rip on {
   traceoptions /var/tmp/rip_pkts detail packets;
};
```

Example 2

This example traces all policy changes.

```
rip on {
    traceoptions policy;
};
```

Example 3

This example restores default behavior.

```
rip on {
    traceoptions none;
};
```

See Also

`traceoptions` page 3

“Trace Statements” on page 15 of *Configuring GateD*

trustedgateways

Name

trustedgateways - sets routers from which RIP will accept routes

Syntax

trustedgateways *gatewaylist***;**

Parameters

gatewaylist - a comma-separated list of IP addresses or host names

Description

trustedgateways allows for additional security beyond that provided by **authentication**. Specifically, it provides a complete list of IP addresses from which the configured router will accept RIP updates. Of course, it is still possible to spoof IP addresses, but, short of that, only those RIP packets originating from the listed hosts will be accepted.

Defaults

All routers are trusted.

Context

rip statement

Examples

Example 1

This example sets a RIP router to heed updates sent from routers with only host addresses of 2 on the 192.168.10 and .12 networks.

```
rip on {
    interface eth;
    trustedgateways 192.168.10.2, 192.168.12.2;
};
```

Example 2

Similarly, this can be accomplished with host names (of course, in this situation, the benefit may not be achieved if DNS is not secure).

```
rip on {
    interface eth;
    trustedgateways rtr1.my.domain, rtr2.my.domain;
};
```

See Also

`interface` on page 58

`authentication` on page 49

update-time

Name

`update-time` - sets the update time for unsolicited route response

Syntax

`update-time` *`update_time`***`;`**

Parameters

`update_time` - an integer from 1 to 30, inclusive

Description

This is a group option in the RIP clause. For example, it is used in the same context as **`preference`** and **`broadcast`**.

Defaults

`update-time 30;`

Context

`rip` statement

Examples

```
rip yes {
    update-time 20;
    interface fxp0;
};
```

See Also

`expire-time` on page 56

version

Name

`version` - specifies the version of RIP to be run

Syntax

`version` *`ripversion`* **`[`***`ripversarg`***`]`**

Parameters

`ripversion` - 1 or 2

`ripversarg` - There are no available arguments for version 1; for version 2, valid arguments are **`broadcast`** and **`multicast`**.

Description

`version` is used to override the default version of RIP that will be run on a given interface. Normally, RIPv1 will be run on all interfaces. If version 2 is specified, then the default behavior depends on the capabilities of the interface. If the interface is multicast capable, then RIP updates will be multicast to RIP2-ROUTERS.MCAST.NET (the reserved, multicast address 224.0.0.9). If the interface is not multicast capable, then RIP version 1 compatible version 2 packets will be broadcast.

The *`ripversarg`* (which, if specified, is exclusively either **`broadcast`** or **`multicast`**, and which can only be specified for version 2) allows the above behavior to be overridden. This is normally used to specify that only version 1-compatible packets should be sent for interoperability purposes, even though a given interface is multicast capable.

Another exception is the case in which version 2 multicast is specified on an interface that is not multicast capable (e.g., a point-to-point link). In this case, if **`sourcegateways`** is also specified, the full RIPv2 packets will be directly unicast to the source gateway on the specified interface.

It is important not to confuse the **`broadcast`** argument with **`version`**, which specifies interoperability between RIPv1 and RIPv2, and the **`broadcast`** command.

Defaults

`version 1`

`version 2` defaults to **`version 2 multicast`** (for multicast capable interfaces; broadcast otherwise).

Context

`rip interface` statement

Examples

Example 1

The following example runs RIPv2 on all interfaces, except for eth0. This would be useful if it were necessary to interoperate with old RIP implementations on a certain link in the network.

```
rip on {
    interface all version 2 multicast;
    interface eth0 version 2 broadcast;
};
```

Example 2

This example runs RIPv1 on all interfaces, which is the default behavior.

```
rip on {
    interface all version 1;
};
```

Example 3

This example runs full RIPv2 on all multicast capable interfaces.

```
rip on {
    interface all version 2;
};
```

Example 4

This example is identical to Example 3 but requires typing an additional word.

```
rip on {
    interface all version 2 multicast;
};
```

Example 5

This example specifies that RIPv2 be run in RIPv1-compatible mode on all interfaces, except across a point-to-point link, where full RIPv2 is run. This might be useful if a serial link connected two networks: one that ran RIPv2, and one that ran RIPv1.

```
rip on {
    interface all version 2 broadcast;
    interface ppp0 version 2 multicast;
    sourcegateways the.remote.host;
};
```

See Also

ripin on page 70

`ripout` on page 72

`metricin` on page 60

`metricout` on page 62

`authentication` on page 49

`secondary authentication` on page 74

`sourcegateways` on page 76

`interface` on page 58

`broadcast` on page 53