Using SNMP

This chapter provides the following information about Cisco Prime Access Registrar (Prime Access Registrar) support for SNMP:

Overview

Prime Access Registrar provides SNMP MIB and trap support for users of network management systems. The supported MIBs enable the network management station to collect state and statistic information from an Prime Access Registrar server. The traps enable Prime Access Registrar to notify interested network management stations of failure or impending failure conditions.

Prime Access Registrar supports the MIBs defined in the following RFCs:

- RADIUS Authentication Client MIB for IPv6, RFC 4668
- RADIUS Authentication Server MIB for IPv6, RFC 4669
- RADIUS Accounting Client MIB for IPv6, RFC 4670
- RADIUS Accounting Server MIB for IPv6, RFC 4671
- CISCO Diameter Base Protocol MIB

Prime Access Registrar MIB support enables a standard SNMP management station to check the current state of the server as well as the statistics on each client or each proxied remote server.

Prime Access Registrar Trap support enables a standard SNMP management station to receive trap messages from an Prime Access Registrar server. These messages contain information indicating that either the server was brought up or down, or that the proxied remote server is down or has come back online.

Supported MIBs

The MIBs supported by Prime Access Registrar enable a standard SNMP management station to check the current state of the server and statistics for each client or proxied remote server.

This section contains the following topics:

RADIUS-AUTH-CLIENT-MIB

The RADIUS-AUTH-CLIENT-MIB describes the client side of the RADIUS authentication protocol. The information contained in this MIB is useful when an Prime Access Registrar server is used as a proxy server.

RADIUS-AUTH-SERVER-MIB

The RADIUS-AUTH-SERVER-MIB describes the server side of the RADIUS authentication protocol. The information contained in this MIB describes managed objects used for managing a RADIUS authentication server.

RADIUS-ACC-CLIENT-MIB

The RADIUS-ACC-CLIENT-MIB describes the client side of the RADIUS accounting protocol. The information contained in this MIB is useful when an Prime Access Registrar server is used for accounting.

RADIUS-ACC-SERVER-MIB

The RADIUS-ACC-CLIENT-MIB describes the server side of the RADIUS accounting protocol. The information contained in this MIB is useful when an Prime Access Registrar server is used for accounting.

CISCO-DIAMETER-BASE-PROTOCOL-MIB

Prime Access Registrar uses the CISCO-DIAMETER-BASE-PROTOCOL-MIB as an interface to query the Diameter statistics, though configuring the Diameter through SNMP is not possible.
Prime Access Registrar supports LocalStatistics and PeerStatiscics only. The LocalStats provides statistical information about the local diameter server and the PeerStats provides statistical information about the peers and the messages to/from the peers.

Diameter SNMP and Statistics Support

Prime Access Registrar also supports Diameter SNMP MIB (CISCO-DIAMETER-BASE-PROTOCOL-MIB) to describe the Diameter Base Protocol statistics.

Prime Access Registrar supports statistic of Diameter messages to include the additional counters. This is supported through the CLI/GUI and SNMP. The diameter statistics includes peer statistics and global summary statistics details.

TACACS+ SNMP and Statistics Support

Prime Access Registrar supports the CISCO-AAA-SERVER-MIB to describe the statistics of TACACS+ protocol.TACACS+ protocol is used to authenticate an user via various services such as login services see TACACS+ Support for AAA for more information. This is supported through the CLI/GUI and SNMP.

SNMP Traps

The traps supported by Prime Access Registrar enable a standard SNMP management station to receive trap messages from an Prime Access Registrar server. These messages contain information indicating whether a server was brought up or down, or that the proxied remote server is down or has come back online.

A trap is a network message of a specific format issued by an SNMP entity on behalf of a network management agent application. A trap is used to provide the management station with an asynchronous notification of an event.

When a trap is generated, a single copy of the trap is transmitted as a trap PDU to each destination contained within a list of trap recipients.

The list of trap recipients is shared by all events and is determined at server initialization time along with other trap configuration information. The list of trap recipients dictates where Prime Access Registrar traps are directed.

The configuration of any other SNMP agent on the host is ignored. By default, all traps are enabled but no trap recipients are defined. By default, no trap is sent until trap recipients are defined.

Traps are configured using the command line interface (CLI). After configuring traps, the configuration information is re initialized when a server reload or restart occurs.

When you configure traps, you must provide the following information:

- List of trap recipients (community string for each)
- Suppressing traps for any type of message
- Frequency of traps for any type of messag

This section contains the following topics:

Supported Traps

The traps supported by Prime Access Registrar enable the Prime Access Registrar server to notify interested management stations of events, failure, or impending failure conditions. Traps are a network message of a specific format issued by an SNMP entity on behalf of a network management agent application. Traps are used to provide the management station with an asynchronous notification of an event.

This section contains the following topics:

carServerStart

carServerStart signifies that the server has started on the host from which this notification was sent. This trap has one object, *carNotifStartType,* which indicates the start type. A *firstStart* indicates this is the server process' first start. *reload* indicates this server process has an internal reload. This typically occurs after rereading some configuration changes, but *reload* indicates this server process did not quit during the reload process.

carServerStop

carServerStop signifies that the server has stopped normally on the host from which this notification was sent.

carInputQueueFull

carInputQueueFull indicates that the percentage of use of the packet input queue has reached its high threshold. This trap has two objects:

- *carNotifInputQueueHighThreshold*—indicates the high limit percentage of input queue usage
- *carNotifInputQueueLowThreshold*—indicates the low limit percentage of input queue usage

By default, *carNotifInputQueueHighThreshold* is set to 90% and *carNotifInputQueueLowThreshold* is set to 60%.

Note The values for these objects cannot be changed at this time. You will be able to modify them in a future release of Prime Access Registrar.

After this notification has been sent, another notification of this type will not be sent again until the percentage usage of the input queue goes below the low threshold.

If the percentage usage reaches 100%, successive requests might be dropped, and the server might stop responding to client requests until the queue drops down again.

carInputQueueNotVeryFull

carInputQueueNotVeryFull indicates that the percentage usage of the packet input queue has dropped below the low threshold defined in *carNotifInputQueueLowThreshold*. This trap has two objects:

- *carNotifInputQueueHighThreshold*—indicates the high limit percentage of input queue usage
- *carNotifInputQueueLowThreshold*—indicates the low limit percentage of input queue usage

After this type of notification has been sent, it will not be sent again until the percentage usage goes back up above the high threshold defined in *carNotifInputQueueHighThreshold*.

carOtherAuthServerNotResponding

carOtherAuthServerNotResponding indicates that an authentication server is not responding to a request sent from this server. This trap has three objects:

- *radiusAuthServerAddress*—indicates the identity of the concerned server
- *radiusAuthClientServerPortNumber*—indicates the port number of the concerned server
- *carAuthServerType*—indicates the type of the concerned server

The index of these three objects identifies the entry in *radiusAuthServerTable* and *carAccServerExtTable* which maintains the characteristics of the concerned server.

Note One should not rely solely on **carOtherAuthServerNotResponding** for server state. Several conditions, including a restart of the Prime Access Registrar server, could result in either multiple *carOtherAuthServerNotResponding* notifications being sent or in a *carOtherAuthServerResponding* notification *not* being sent. NMS can query the *carAuthServerRunningState* in *carAuthServerExtTable* for the current running state of this server.

carOtherAuthServerResponding

carOtherAuthServerResponding signifies that an authentication server which had formerly been in a *down* state is now responding to requests from the Prime Access Registrar server. This trap has three objects:

- *radiusAuthServerAddress*—indicates the identity of the concerned server
- *radiusAuthClientServerPortNumber*—indicates the port number of the concerned server
- *carAuthServerType*—indicates the type of the concerned server

The index of these three objects identifies the entry in *radiusAuthServerTable* and *carAccServerExtTable* which maintains the characteristics of the concerned server.

One should not rely on receiving this notification as an indication that all is well with the network. Several conditions, including a restart of the Prime Access Registrar server, could result in either multiple *carOtherAuthServerNotResponding* notifications being sent or in a *carOtherAuthServerResponding* notification *not* being sent. The NMS can query the *carAuthServerRunningState* in *carAuthServerExtTable* for the current running state of this server.

carOtherAccServerNotResponding

carOtherAuthServerNotResponding signifies that an accounting server is not responding to the requests sent from this server. This trap has three objects:

- *radiusAccServerAddress*—indicates the identity of the concerned server
- *radiusAccClientServerPortNumber*—indicates the port number of the concerned server
- *carAcchServerType*—indicates the type of the concerned server

The index of these three objects identifies the entry in *radiusAuthServerTable* and *arAccServerExtTable* which maintains the characteristics of the concerned server.

One should not solely rely on this for server state. Several conditions, including the restart of the Prime Access Registrar server, could result in either multiple *carOtherAccServerNotResponding* notifications being sent or in a *carOtherAccServerResponding* notification *not* being sent. The NMS can query the *carAccServerRunningState* in *carAccServerExtTable* for current running state of this server.

carOtherAccServerResponding

carOtherAccServerResponding signifies that an accounting server that had previously sent a *not responding* message is now responding to requests from the Prime Access Registrar server. This trap has three objects:

- *radiusAccServerAddress*—indicates the identity of the concerned server
- *radiusAccClientServerPortNumber*—indicates the port number of the concerned server
- *carAccServerType*—indicates the type of the concerned server

The index of these three objects identifies the entry in *radiusAuthServerTable* and *arAccServerExtTable* which maintains the characteristics of the concerned server.

One should not rely on the reception of this notification as an indication that all is well with the network. Several conditions, including the restart of the Prime Access Registrar server, could result in either multiple *carOtherAccServerNotResponding* notifications being sent or in a **carOtherAccServerResponding** notification *not* being sent. The NMS can query the *carAccServerRunningState* in *carAccServerExtTable* for the current running state of this server.

carAccountingLoggingFailure

carAccountingLoggingFailure signifies that this Prime Access Registrar server cannot record accounting packets locally. This trap has two objects:

- carNotifAcctLogErrorReason—indicates the reason packets cannot be recorded locally
- *carNotifAcctLogErrorInterval*—indicates how long to wait until another notification of this type might be sent. A value of 0 (zero) indicates no time interval checking, meaning that no new notification can be sent until the error condition is corrected.

carLicenseUsage

carLicenseUsage signifies the percentage of transaction per second(TPS) or session License Usage.

TPS

The TPS trap is generated when the Prime Access Registrar server reaches license usage slabs namely 80%, 90%, 100%, and 110%. These traps are generated only once for every slab during the increasing steady state. Increasing steady state is a state when Prime Access Registrars' incoming request rate shows 80% of the license usage over a period of 20 minutes. These traps will be regenerated only if a increasing steady state is observed after a decreasing steady state.

Concurrent Session

The concurrent session trap is generated when the Prime Access Registrar server reaches 80%. The incoming traffic slabs defined for trap generation are 80%, 90%, 100%, and 110% of the licensed Concurrent Sessions. These traps are generated once for every slab during the increasing steady state.

carDiameterPeerDown

carDiameterPeerDown signifies that a Diameter peer is down. The identity of the peer is given by cdbpPeerIpAddress.

carDiameterPeerUp

carDiameterPeerUp signifies that a Diameter peer is up. The identity of the peer is given by cdbpPeerIpAddress.

Configuring Traps

The Prime Access Registrar SNMP implementation uses various configuration files to configure its applications.

This section contains the following topics:

Directories Searched

Configuration files can be found and read from numerous places. By default, SNMP applications look for configuration files in the following three directories (in the order listed):

1. /usr/local/share/snmp/snmp.conf

   This directory contains common configuration for the agent and the application. See man page **snmp.conf(5)** for details.

2. /usr/local/share/snmp/snmpd.conf

3. /usr/local/share/snmp/snmp.local.conf

   This directory configures the agent. See man page **snmp.conf(5)** for details.

In each of these directories, an SNMP application looks for files with the extension ***.conf***. The application also looks for configuration files in default locations where a configuration file can exist for any given configuration file type.

These files are optional and are only used to configure the extensible portions of the agent, the values of the community strings, and the optional trap destinations. By default, the first community string ("public" by default) is allowed read-only access and the second ("private" by default) is allowed write access, as well. The third to fifth community strings are also read-only.

Additionally, the above default search path can be over-ridden by setting the environmental variable SNMPCONFPATH to a colon-separated list of directories to search.

Finally, applications that store persistent data will also look for configuration files in the **/var/snmp** directory.

Configuration File Types

Each application can use multiple configuration files, which will configure various different aspects of the application. For instance, the SNMP agent (**snmpd**) knows how to understand configuration directives in both the **snmpd.conf** and the **snmp.conf** files. In fact, most applications understand how to read the contents of the **snmp.conf** files. Note, however, that configuration directives understood in one file might not be understood in another file. For further information, read the associated manual page with each configuration file type. Also, most of the applications support a '-H' switch on the command line that will list the configuration files it will look for and the directives in each one that it understands.

The **snmp.conf** configuration file is intended to be a application suite-wide configuration file that supports directives that are useful for controlling the fundamental nature of all of the SNMP applications, such as how they all manipulate and parse the textual SNMP MIB files.

Switching Configuration Files in Mid-File

It's possible to switch in mid-file the configuration type that the parser is supposed to be reading. Since that output for the agent by default, but you didn't want to do that for the rest of the applications (for example, **snmpget** and **snmpwalk**, you would need to put a line like the following into the **snmp.conf** file.

```
dumpPacket true
```

But, this would turn it on for all of the applications. So, instead, you can put the same line in the snmpd.conf file so that it only applies to the snmpd demon. However, you need to tell the parser to expect this line. You do this by putting a special type specification token inside a square bracket ([ ]) set. In other words, inside your **snmpd.conf** file you could put the above **snmp.conf** directive by adding a line like the following:

```
[snmp] dumpPacket true
```

This tells the parser to parse the above line as if it were inside a **snmp.conf** file instead of an snmpd.conf file. If you want to parse a bunch of lines rather than just one then you can make the context switch apply to the remainder of the file or until the next context switch directive by putting the special token on a line by itself:

```
# make this file handle snmp.conf tokens:
[snmp]
dumpPacket true
```

```
logTimestamp true
# return to our original snmpd.conf tokens:
[snmpd]
rocommunity mypublic
```

Configuring Trap Recipient

The following example shows the default configuration that sets up trap recipients for SNMP versions v1 and v2c.

Note Most sites use a single NMS, not two as shown below.

```
# --------------------------------------------------------------------------
trapcommunity trapcom
trapsink zubat trapcom 162
trap2sink ponyta trapcom 162
##############################################################################
```

Note **trapsink** is used in SNMP version 1; **trap2sink** is used in SNMP version 2.

trapcommunity defines the default community string to be used when sending traps. This command must appear prior to **trapsink** or **trap2sink** which use this community string.

trapsink and **trap2sink** are defined as follows:

trapsink hostname community port

trap2sink hostname community port

Community String

A community string is used to authenticate the trap message sender (SNMP agent) to the trap recipient (SNMP management station). A community string is required in the list of trap receivers.