Field-Tested Solutions to Cisco Router Problems

Cisco Cookbook

O'REILLY®

Kevin Dooley & Ian J. Brown

Cisco Cookbook

Cisco Cookbook

Ian Brown and Kevin Dooley

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

CHAPTER 17

Simple Network Management Protocol

17.0 Introduction

Since its introduction in 1988, the Simple Network Management Protocol (SNMP) has become the most popular network management protocol for TCP/IP based networks. The IETF created SNMP to allow remote management of IP based devices using a standardized set of operations. It is now widely supported by servers, printers, hubs, switches, modems, UPS systems, and (of course) Cisco routers.

The SNMP set of standards define much more than a communication protocol used for management traffic. The standards also define how management data should be accessed and stored, as well as the entire distributed framework of SNMP agents and servers. The IETF has officially recognized SNMP as a fully standard part of the IP protocol suite. The original SNMP definition is documented in RFC 1157.

In 1993, SNMP Version 2 (SNMPv2) was created to address a number of functional deficiencies that were apparent in the original protocol. The added and improved features included better error handling, larger data counters (64-bit), improved efficiency (get-bulk transfers), confirmed event notifications (informs), and most notably, security enhancements. Unfortunately, SNMPv2 did not become widely accepted because the IETF was unable to come to a consensus on the SNMP security features.

So, a revised edition of SNMPv2 was released in 1996, which included all of the proposed enhancements except for the security facility. It is discussed in RFCs 1905, 1906, and 1907. The IETF refers to this new version as SNMPv2c and it uses the same insecure security model as SNMPv1. This model relies on passwords called *community strings* that are sent over the network as clear-text. SNMPv2c never enjoyed widespread success throughout the IP community. Consequently, most organizations continue to use SNMPv1 except when they need to access the occasional large counter variable. The IETF recently announced that SNMPv3 would be the new standard, with SNMPv1, SNMPv2, and SNMPv2c being considered purely historical.

Cisco's IOS supported SNMPv2 until Version 11.2(6)F, when Cisco began supporting SNMPv2c. Cisco continues to support SNMPv2c in every IOS version beginning with 11.2(6)F. In addition, every version of IOS has supported SNMPv1 since the earliest releases.

The compromise that became SNMPv2c left the management protocol without satisfactory security features. So, in 1998, the IETF began working on SNMPv3, which is defined in RFCs 2571–2575. Essentially, SNMPv3 is a set of security enhancements to be used in conjunction with SNMPv2c. This means that SNMPv3 is not a standalone management protocol and does not replace SNMPv2c or SNMPv1.

SNMPv3 provides a secure method for accessing devices using authentication, message integrity, and encryption of SNMP packets throughout the network. We have included a recipe describing how to use the SNMPv3 security enhancements (see Recipe 17.21). Table 17-1 lists the three supported versions of SNMP and highlights their security capabilities.

Table 17-1. SNMP versions supported by Cisco

Version	Authentication	Encryption	Description
v1	Community strings	None	Trivial authentication. Packets sent in clear-text.
v2c	Community strings	None	Trivial authentication. Packets sent in clear-text.
v3(noAuthNoPriv)	Username	None	Trivial authentication. Packets sent in clear-text.
v3(authNoPriv)	SHA or MD5 encrypted pass phrase	None	Strong authentication. Packets sent in clear-text.
v3(authPriv)	SHA or MD5 encrypted pass phrase	DES	Strong authentication. Packets are encrypted.

SNMP Management Model

SNMP defines two main types of entities, *managers* and *agents*. A manager is a server that runs network management software that is responsible for a particular network. These servers are commonly referred to as Network Management Stations (NMS). There are several excellent commercial NMS platforms on the market. Throughout this book we will refer to the freely distributed NET-SNMP system as a reference NMS.

An agent is an embedded piece of software that resides on a remote device that you wish to manage. In fact, almost every IP-capable device provides some sort of built-in SNMP agent. The agent has two main functions. First, the agent must listen for incoming SNMP requests from the NMS and respond appropriately. And second, the agent must monitor internal events and create SNMP traps to alert the NMS that something has happened. This chapter will focus mainly on how to configure the router's agent.

The NMS is usually configured to poll all of the key devices in the network periodically using *SNMP Get* requests. These are UDP packets sent to the agent on the well-known SNMP port 161. The SNMP Get request prompts the remote device to respond with one or more pieces of relevant operating information.

However, because there could be hundreds or thousands of remote devices, it is often not practical to poll a particular remote device more often than once every few minutes (and in many networks you are lucky if you can poll each device more than a few times per hour). On a schedule like this, a remote device may suffer a serious problem that goes undetected—it's possible to crash and reboot in between polls from the NMS. So, on the next poll, the NMS will see everything operating normally and never know that it completely missed a catastrophe.

Therefore, an SNMP agent also has the ability to send information using an *SNMP trap* without having to wait for a poll. A trap is an unsolicited piece of information, usually representing a problem situation (although some traps are more informational in nature). Traps are UDP packets sent from the agent to the NMS on the other well-known SNMP port number, 162. There are many different types of traps that an agent can send, depending on what type of equipment it manages. Some traps represent non-critical issues. It is often up to the network administrator to decide which types of traps will be useful.

The NMS does not acknowledge traps, and since traps are often sent to report network problems, it is not uncommon for trap reports to get lost and never make it to the NMS. In many cases, this is acceptable because the trap represents a transient transmission problem that the NMS will discover by other means if this trap is not delivered. However, critical information can sometimes be lost when a trap is not delivered.

To address this shortcoming, SNMPv2c and SNMPv3 include another type of packet called an *SNMP inform*. This is nearly identical to a standard trap, except that the SNMP agent will wait for an acknowledgement. If the agent does not receive an acknowledgement within a certain amount of time, it will attempt to retransmit the inform.

SNMP informs are not common today because SNMPv2c was never widely adopted. However, SNMPv3 also includes informs. Since SNMPv3 promises to become the mainstream SNMP protocol, it seems inevitable that enhancements such as SNMP informs will start to be more common.

MIBs and OIDs

SNMP uses a special tree structure called a Management Information Base (MIB) to organize the management data. People will often talk about different MIBs, such as the T1 MIB, or an ATM MIB. In fact, these are all just branches or extensions of the

same global MIB tree structure. However, the relative independence of these different branches makes it convenient to talk about them this way.

A particular SNMP agent will care only about those few MIB branches that are relevant to the particular remote device this agent runs on. If the device doesn't have any T1 interfaces, then the agent doesn't need to know anything about the T1 branch of the global MIB tree. Similarly, the NMS for a network containing no ATM doesn't need to be able to resolve any of the variables in the ATM branches of the MIB tree.

The MIB tree structure is defined by a long sequence of numbers separated by dots, such as `.1.3.6.1.2.1.1.4.0`. This number is called an Object Identifier (OID). Since we will be working with OID strings throughout this chapter, it is worthwhile to briefly review how they work and what they mean.

The OID is a numerical representation of the MIB tree structure. Each digit represents a node in this tree structure. The trunk of the tree is on the left; the leaves are on the right. In the example string, `.1.3.6.1.2.1.1.4.0`, the first digit, `.1`, signifies that this variable is part of the MIB that is administered by the International Standards Organization (ISO). There are other nodes at this top level of the tree. The International Telephone and Telegraph Consultative Committee (CCITT) administers the `.0` tree structure. The ISO and CCITT jointly administer `.2`.

The first node under the ISO MIB tree of this example is `.3`. The ISO has allocated this node for all other organizations. The U.S. Department of Defense (DOD) is designated by the branch number `.6`. The DOD, in turn has allocated branch number `.1` for the Internet Activities Board (IAB). So, just about every SNMP MIB variable you will ever see will begin with `.1.3.6.1`.

There are four commonly used subbranches under the IAB (also called simply "Internet") node. These are designated *directory* (1), *mgmt* (2), *experimental* (3) and private (4). The *directory* node is seldom used in practice. The *mgmt* node is used for all IETF-standard MIB extensions, which are documented in RFCs. This would include, for example, the T1 and ATM examples mentioned earlier. However, it would not include any vendor-specific variables such as the CPU utilization on a Cisco router. SNMP protocol and application developers use the *experimental* subtree to hold data that is not yet standard. This allows you to use experimental MIBs in a production network without fear of causing conflicts. Finally, the *private* subtree contains vendor specific MIB variables.

Before returning to the example, we want to take a brief detour down the *private* tree, because many of the examples in this book include Cisco-specific MIB variables. A good example of a Cisco MIB variable is `.1.3.6.1.4.1.9.2.1.8.0`, which gives the amount of free memory in a Cisco router. There is only one subtree under the *private* node, and it is called enterprises, `.1.3.6.1.4.1`. Of the hundreds of registered owners of private MIB trees, Cisco is number 9, so all Cisco-specific MIB extensions begin with `.1.3.6.1.4.1.9`.

Referring again to the previous example string (`.1.3.6.1.2.1.1.4.0`), you can see this represents a variable in the *mgmt* subtree, `.1.3.6.1.2`. The next digit is `.1` here, which represents an SNMP MIB variable.

The following digit, `.1`, refers to a specific group of variables, which, in the case of *mgmt* variables, would be defined by an RFC. In this particular case, the value `.1` refers to the *system* MIB, which is detailed in RFC 1450.

From this level down, a special naming convention is adopted to help you to remember which MIB you are looking at. The names of every variable under the *system* node begin with "sys". They are *sysDescr* (1), *sysObjectID* (2), *sysUpTime* (3), *sysContact* (4), *sysName* (5), *sysLocation* (6), *sysServices* (7), *sysORLastChange* (8), and *sysORTable* (9). You can find detailed descriptions of what all of these mean in RFC 1450.

In fact, reading through MIB descriptions is not only an excellent way to understand the hierarchical structure of the MIB, but it's also extremely useful when you are trying to decide what information you can and should be extracting from your equipment.

In the example string, `.1.3.6.1.2.1.1.4.0`, the value is `.4`, for `sysContact`. The following `.0` tells the agent to send the contents of this node, rather than treating it as the root of further subtrees. So the OID string uniquely identifies a single piece of information. In this case, that information is the contact information for the device.

17.1 Configuring SNMP

Problem

You want to set up basic SNMP services on a router.

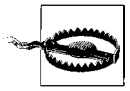
Solution

To enable read-only SNMP services, use the following configuration command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server community ORARO ro
Router(config)#end
Router#
```

To enable read-write SNMP services, use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server community ORARW rw
Router(config)#end
Router#
```



It is extremely risky to enable read-write SNMP services without the appropriate security controls. Please read the following discussion section before implementing this recipe.

Starting with IOS Version 12.0(3)T, Cisco introduced a new system for configuring SNMP services using the *snmp-server group* and *snmp-server user* configuration commands. Use the following commands to enable read-only SNMP services with this new method:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server group COOKRO v1
Router(config)#snmp-server user TESTRO1 COOKRO v1
Router(config)#snmp-server group BOOKRO v2c
Router(config)#snmp-server user TESTRO2 BOOKRO v2c
Router(config)#end
```

Discussion

SNMP services are disabled by default on all Cisco routers. The examples highlighted in the solutions section show only how to configure the router to allow inbound SNMP services so that it will respond to SNMP Get and Set requests. These configuration examples do not enable SNMP traps or informs, which we discuss in Recipe 17.13.

When inbound SNMP services are enabled, the router starts to listen for incoming SNMP requests on UDP port 161. It is important to note that these methods enable SNMPv1 and SNMPv2c only (SNMPv3 is covered in Recipe 17.22).

The first example shows the older method of enabling SNMP services. It uses the *snmp community* command to simultaneously enable both SNMPv1 and SNMPv2c:

```
Router(config)#snmp-server community ORARO ro
```

Cisco documentation often refers to this as *bilingual* SNMP support because it allows the router to speak both SNMP languages (or versions).

The *show snmp group* command gives details on exactly what SNMP versions are enabled, as well as the security models they use. Running this command after implementing the first two configuration examples gives the following output:

```
Router#show snmp group
groupname: ORARO                          security model:v1
readview :v1default                       writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active

groupname: ORARO                          security model:v2c
readview :v1default                       writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active
```

```
groupname: ORARW                          security model:v1
readview :v1default                       writeview: v1default
notifyview: <no notifyview specified>
row status: active

groupname: ORARW                          security model:v2c
readview :v1default                       writeview: v1default
notifyview: <no notifyview specified>
row status: active
```

This shows that the groups we have configured each created two entries: one for SNMPv1 support, and the other for SNMPv2c. This is possible because SNMPv1 and SNMPv2c use the same community string authorization model. Therefore, the router is capable of responding to either version of SNMP.

When using the *snmp-server group* command for enabling SNMP services, you can create an SNMP group entry that belongs to a single SNMP security model (SNMPv1 or SNMPv2c). Cisco added this command to support SNMPv3 services, which may eventually replace the legacy method.

Both methods for enabling SNMP services assign a community string that acts as a password of sorts to protect access. SNMP services run on a well-known UDP port, so the router needs this password to help prevent unauthorized access. The router will simply discard any SNMP requests that contain incorrect SNMP community strings. It is important to note that in both Version 1 and 2c, SNMP transmits these community strings through the network in clear-text, making it relatively insecure.

You can configure the router for either read-only or read-write SNMP service. Read-only access means that users can view the router's MIB tree. This is relatively benign, although information about the router's configuration could be useful to someone planning an attack on your network. Read-write access, on the other hand, permits users to change some of the values in the MIB tree. A user with read-write access could potentially wreak havoc by disabling IP routing, disabling interfaces, erasing router flash, downloading router configurations, uploading configuration commands, or making a variety of other dangerous changes.

Be extremely careful when providing SNMP write access. If SNMP write access is not absolutely required, we recommend disabling it. Far too many organizations automatically enable full SNMP write access without regard for the dangers of possible unauthorized changes.

If write access is required, you will first need to define SNMP views as described in Recipe 17.7. But a better solution is to simply use SNMPv3, which we discuss in Recipe 17.22. SNMPv3 offers advanced authentication and encryption services that ensure safe delivery over insecure networks. Unfortunately, SNMPv3 is still a relatively young standard, and many NMS systems don't yet support it.

So, in those cases where SNMP Version 1 or 2c write access is an absolute requirement, we recommend using the security features described in Recipes 17.5, 17.6, and

17.7. These describe how to implement SNMP ACLs, limit SNMP views, and restrict SNMP TFTP access to help to reduce the risk of attack.

See Also

Recipe 17.13; Recipe 17.5; Recipe 17.6; Recipe 17.7; Recipe 17.22

17.2 Extracting Router Information via SNMP Tools

Problem

You wish to extract or change router information via SNMP.

Solution

To extract router information via SNMP, we will use the suite of SNMP tools provided with the NET-SNMP toolkit (see Appendix A for more details).

Use *snmpget* to extract a single MIB object from the router's MIB tree. This example uses *snmpget* to extract the router's system contact information:

```
freebsd% snmpget -v1 -c ORARO Router .1.3.6.1.2.1.1.4.0
system.sysContact.0 = Helpdesk  800-555-2992
```

Use *snmpset* to alter MIB objects within the router's MIB tree. The next example demonstrates how to modify MIB variables, using *snmpset* to change the system contact information:

```
freebsd% snmpset -v1 -c ORARW Router .1.3.6.1.2.1.1.4.0 s "Ian Brown 555-1221"
system.sysContact.0 = Ian Brown 555-1221
freebsd% snmpget -v1 -c ORARO Router sysContact.0
system.sysContact.0 = Ian Brown 555-1221
```

The *snmpwalk* utility extracts a series of MIB objects from the router's MIB tree. This example uses *snmpwalk* to extract all of the router's interface descriptions:

```
freebsd% snmpwalk -v1 -c ORARO Router ifDescr
interfaces.ifTable.ifEntry.ifDescr.1 = "Ethernet0"
interfaces.ifTable.ifEntry.ifDescr.2 = "Serial0"
interfaces.ifTable.ifEntry.ifDescr.3 = "Serial1"
interfaces.ifTable.ifEntry.ifDescr.4 = "Null0"
interfaces.ifTable.ifEntry.ifDescr.5 = "Loopback0"
interfaces.ifTable.ifEntry.ifDescr.6 = "Serial0.1"
freebsd%
```

Discussion

In this recipe, we use the suite of SNMP tools provided by the NET-SNMP project (formerly UCD-SNMP) to demonstrate basic SNMP functionality. NET-SNMP

provides a variety of useful SNMP tools that you can run from the command-line interface of any Unix or Windows workstation. This software is freely distributed and is available on a variety of platforms, which makes it extremely popular for scripts of all shapes and sizes. We consider NET-SNMP to be a sort of Swiss army knife of SNMP that wonderfully illustrates the usefulness of SNMP for working with Cisco routers. Of course, many commercial software vendors also provide SNMP tools that are equally effective and frequently include a graphical user interface. The underlying concepts remain the same, even if the command syntax differs. In some cases it is easier to do the types of SNMP commands shown in this recipe using a graphical user interface, rather than a command-line utility.

NET-SNMP provides a set of SNMP utilities for performing various useful SNMP functions. This recipe used three of the most basic tools.

- *snmpget* gets a single MIB object and displays its contents. To do this, it sends the router an SNMP "get" request for a particular MIB object. The router responds with the value of the MIB object, if present. The command syntax for SNMPv1 and SNMPv2c queries is:

  ```
  snmpget  [options] {-c <community-string>} <hostname> [<MIB Object or OID>]
  ```

- *snmpwalk* asks the router for a group of related MIB objects and displays their contents. It does this by sending the router a series of SNMP "get-next" commands to list all available MIB objects under the specified node in an MIB tree. The router will continue to respond to the server's "get-next" requests until it reaches the end of the MIB subtree. The command syntax is as follows:

  ```
  snmpwalk [options] {-c <community-string>} <hostname>  [<MIB Object or OID>]
  ```

Note that leaving out the MIB object or OID causes *snmpwalk* to walk the entire MIB tree. This can cause CPU overload problems on the router, as well as congestion problems on low-speed links.

- *snmpset* modifies the contents of a MIB object and displays the changed variable, if successful. It works by sending the router an SNMP "set" request for the specified MIB object. If the requested change is legal, the router will change the value of the corresponding MIB variable and respond back.

  Not all MIB entries can be changed by an SNMP set. For example, it doesn't make sense to change the physical media type of an interface. And, of course, the router has to be configured to allow SNMP read-write access. The command syntax is as follows:

  ```
  snmpset [options] {-c <community>} <hostname> [<objectID> <type> <value>]
  ```

Most NMS systems have similar commands that you can access from the command line and use in scripts. See your software documentation for details.

Table 17-2 shows a number of useful MIB entries and their associated OID numbers. Several of these variables are Cisco-specific, and will not make sense if used on equipment from other vendors.

Table 17-2. Common Cisco router SNMP MIB entries

MIB name	Description	OID
sysName	Hostname	.1.3.6.1.2.1.1.5.0
sysUpTime	Uptime	.1.3.6.1.2.1.1.3.0
sysDescr	System Description	.1.3.6.1.2.1.1.1.0
sysContact	System Contact	.1.3.6.1.2.1.1.4.0
sysLocation	System Location	.1.3.6.1.2.1.1.6.0
ciscoImageString.5	IOS Version	.1.3.6.1.4.1.9.9.25.1.1.1.2.5
avgBusy1	1-Minute CPU Util.	.1.3.6.1.4.1.9.2.1.57.0
avgBusy5	5-Minute CPU Util.	.1.3.6.1.4.1.9.2.1.58.0
freeMem	Free memory	.1.3.6.1.4.1.9.2.1.8.0
ciscoImageString.4	IOS feature set	.1.3.6.1.4.1.9.9.25.1.1.1.2.4
whyReload	Reload reason	.1.3.6.1.4.1.9.2.1.2.0

A complete listing of Cisco-supported MIBs are located at *http://www.cisco.com/public/sw-center/netmgmt/cmtk/mibs.shtml*. Note that this includes a huge amount of information. However, with a little time and effort, you should be able to find a way to extract exactly the information you need.

You can extract the same MIB objects using SNMPv2c:

```
freebsd% snmpget -v 2c -c ORARO Router sysContact.0
system.sysContact.0 = Ian Brown 416-555-2943
freebsd%
```

The only difference in this example is that we specified the SNMP version number as part of the *snmpget* command syntax. This is useful because SNMPv2c introduced 64-bit counters. Cisco supports a small number of MIB objects that can only be accessed using SNMPv2c (or SNMPv3). One such MIB object is *ifHCInOctets*:

```
Freebsd% snmpwalk -v 2c -c ORARO Router ifHCInOctets
ifHCInOctets.7 = Counter64: 145362298
ifHCInOctets.8 = Counter64: 85311547
Freebsd%
```

This MIB object counts the number of inbound bytes (octets) received by an interface. The older SNMPv1 *ifInOctets* MIB object counts exactly the same thing, but uses a 32-bit variable to hold the number. The newer object does not roll over to zero as often, making it more useful for high-speed interfaces. If you attempt to get one of these 64-bit counter objects using SNMPv1, the query will fail.

See Also

Recipe 17.1; Recipe 17.21; Appendix A

17.3 Recording Important Router Information for SNMP Access

Problem

You want to record important information such as physical locations, contact names, and serial numbers for later SNMP access.

Solution

To record important physical information regarding the router, use the following commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server contact Ian Brown 416-555-2943
Router(config)#snmp-server location 999 Queen St. W., Toronto, Ont.
Router(config)#snmp-server chassis-id JAX123456789
Router(config)#end
Router#
```

Discussion

It is an extremely good network management practice to add useful information such as contact names, router locations, and serial numbers directly into the router configuration. This information can be extracted later using *snmpget* requests, either directly or invoked from scripts that make output easier to understand. This is true not only for Cisco routers—it's good practice to configure serial numbers and locations on all equipment so that they can be read with SNMP. When a field technician swaps or moves a piece of equipment, they can update this information. This makes it easy to verify the information stored in your central equipment inventory database.

When you save the contact name, router location, and serial number in the router configuration, it is also easy to extract from the command line and configuration files:

```
Router#show snmp
Chassis: JAX123456789
Contact: Ian Brown 416-555-2943
Location: 999 Queen St. W., Toronto, Ontario
417 SNMP packets input
    0 Bad SNMP version errors
    141 Unknown community name
    3 Illegal operation for community name supplied
    0 Encoding errors
    224 Number of requested variables
    49 Number of altered variables
    224 Get-request PDUs
```

```
    0 Get-next PDUs
    52 Set-request PDUs
299 SNMP packets output
    0 Too big errors (Maximum packet size 1500)
    3 No such name errors
    0 Bad values errors
    0 General errors
    276 Response PDUs
    23 Trap PDUs

SNMP logging: enabled
    Logging to 172.25.1.1.162, 0/10, 21 sent, 2 dropped.
Router#
```

You can also extract individual pieces of information using the following commands:

```
Router#show snmp contact
Ian Brown 416-555-2943
Router#show snmp location
999 Queen St. W., Toronto, Ontario
Router#show snmp chassis
JAX123456789
Router#
```

It is also useful to extract this information from a backup of the router's configuration file. It is a good network management practice to keep a backup copy of every router's configuration on a central server such as the NMS. Then you can extract vital information such as the router's serial number. You will often need this information for service and support when a device fails and you can't reach it through SNMP. On a Unix server, you can use the *grep* utility to easily extract the required information:

```
Freebsd% grep snmp-server Router.confg
snmp-server community ORARO RO
snmp-server community ORARW RW
snmp-server location 999 Queen St. W., Toronto, Ontario
snmp-server contact Ian Brown 416-555-2943
snmp-server chassis-id JAX123456789
snmp-server host 172.25.1.1 ORATRAP
Freebsd%
```

If the router is reachable, you can also extract this information via SNMP:

```
Freebsd% snmpget -v1 -c ORARO Router .1.3.6.1.2.1.1.6.0
system.sysLocation.0 = 999 Queen St. W., Toronto, Ontario
Freebsd% snmpget -v1 -c ORARO Router .1.3.6.1.2.1.1.4.0
system.sysContact.0 = Ian Brown 416-555-2943
Freebsd% snmpget -v1 -c ORARO Router .1.3.6.1.4.1.9.3.6.3.0
enterprises.9.3.6.3.0 = "JAX123456789"
Freebsd%
```

Recipe 17.4 uses this information to construct a summary inventory report for all of the routers in a network.

See Also

Recipe 17.4

17.4 Extracting Inventory Information from a List of Routers with SNMP

Problem

You want to build a report of important router information for all of your managed routers.

Solution

The following Perl script extracts important router information such as router name, physical locations, contact names, and serial numbers from a list of routers, and creates a report of this information. The script is intended to be run manually and no arguments are required or expected.

Here's some example output:

```
Freebsd% ./inventory.pl
  Router             Location                    Contact              Serial
Router     999 Queen St. W., Toronto, Ont  Ian Brown 416-555-2943     JAX123456
Boston     1273 Main Street, Boston, MA    Bob Irwin 800-555-1221     JAX231567
Denver     24 Sussex Drive, Denver, CO     Helpdesk  800-555-2992     JAX928362
Frame      999 Queen St. W., Toronto, Ont  Ian Brown 416-555-2943     JAX212321
Toronto    999 Queen St. W., Toronto, Ont  Ian Brown 416-555-2943     JAX283291
Boston2    1273 Main Street, Boston, MA    Bob Irwin 800-555-1221     JAX292228
Denver2    24 Sussex Drive, Denver, CO     Helpdesk  800-555-2992     JAX219115
Freebsd%
```

Example 17-1 contains the Perl code.

Example 17-1. inventory.pl

```
#!/usr/bin/perl
#
#    inventory.pl -- a script to extract valuable information
#                    from a Router.  (Name, Location, Contact, S/N)
#
#
# Set behaviour
$workingdir="/home/nms";
$snmpro="ORARO";
```

Example 17-1. inventory.pl (continued)

```
$rtrlist="$workingdir/RTR_LIST";
#
#
open (RTR, "$rtrlist") || die "Can't open $rtrlist file";
open (LOG, ">$workingdir/RESULT") || die "Can't open $workingdir/RESULT file";
printf "  Router\t\t Location\t\t\tContact\t\t  Serial\n";
printf LOG  "  Router\t\t; Location\t\t\t;Contact\t\t  ;Serial\n";
while (<RTR>) {
  chomp($rtr="$_");
  $snmpget="/usr/local/bin/snmpget -v1 -c $snmpro $rtr ";
  $rtr=`$snmpget .1.3.6.1.4.1.9.2.1.3.0`;
  $loc=`$snmpget .1.3.6.1.2.1.1.6.0`;
  $con=`$snmpget .1.3.6.1.2.1.1.4.0`;
  $sin=`$snmpget .1.3.6.1.4.1.9.3.6.3.0`;
  chomp(($foo, $RTR) = split (/"/, $rtr));
  chomp(($foo, $LOC) = split (/= /, $loc));
  chomp(($foo, $CON) = split (/= /, $con));
  chomp(($foo, $SIN) = split (/"/, $sin));
  printf ("%-12.12s  %-30.30s  %-25.25s  %-12.12s\n", $RTR, $LOC, $CON, $SIN);
  printf LOG ("%-12.12s; %-30.30s; %-25.25s; %-12.12s\n", $RTR, $LOC, $CON, $SIN);
}
```

Discussion

This script extracts important router information from a list of routers and presents that list in a semicolon-delimited file, as well as displaying it on the screen. It is a very simple script that just extracts MIB variables from a list of routers using *snmpget*, illustrating the value and versatility of SNMP.

This Perl script works by cycling through a list of routers. For each router, it uses SNMP get requests to extract the values of four variables: router name, location, contact, and serial number, as we did manually in Recipe 17.3. The core of this script is NET-SNMP's *snmpget* utility. NET-SNMP must be present on the server and the script expects to find *snmpget* in the */usr/local/bin* directory.

This script contains three important variables that you have to set correctly before you can run this script. First, the variable *$workingdir* contains the directory in which the script resides. Next is the read-only SNMP community string, which is stored in the script's *$snmpro* variable. Substitute your organization's SNMP read-only community string for the example value shown (*ORARO*). Note that this script assumes that you use the same SNMP read-only community string on all routers.

The third variable, *$rtrlist*, contains the location of the router list. The example script uses a file called *RTR_LIST*, which is located in the working directory. You will need to change this variable to point to a file containing a list of routers on your network. The script expects this file to have a single router name per line.

The script produces two types of output. It displays the results on your screen while the script is executing. And the script also logs all results to a file called *RESULT* that resides in the working directory. The script automatically creates this file the first time you run it, and it overwrites the contents on each subsequent execution.

See Also

Recipe 17.3

17.5 Using Access Lists to Protect SNMP Access

Problem

You want to provide extra security to SNMP using access lists.

Solution

You can use the following commands to restrict which IP source addresses are allowed to access SNMP functions on the router. This is the legacy method:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 99 permit 172.25.1.0 0.0.0.255
Router(config)#access-list 99 permit host 10.1.1.1
Router(config)#access-list 99 deny any
Router(config)#snmp-server community ORARO ro 99
Router(config)#access-list 98 permit 172.25.1.0 0.0.0.255
Router(config)#snmp-server community ORARW rw 98
Router(config)#end
Router#
```

Here is a newer method to do the same thing using SNMP server groups:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 99 permit 172.25.1.0 0.0.0.255
Router(config)#access-list 99 permit host 10.1.1.1
Router(config)#access-list 99 deny any
Router(config)#snmp-server group COOKRO v1 access 99
Router(config)#snmp-server user TESTRO1 COOKRO v1
Router(config)#end
Router#
```

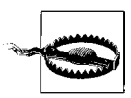
Discussion

By default, when you enable inbound SNMP services, the router will permit all IP addresses to access the SNMP agent on the standard well-known UDP port number (161). We highly recommend using SNMP ACLs to restrict SNMP access to a few trusted hosts or subnets. This will help to protect sensitive data.

You could restrict SNMP access by simply applying an interface ACL to block incoming SNMP packets that don't come from trusted servers. However, this would not be as effective as using the global SNMP commands shown in this recipe. Because you can apply this method once for the whole router, it is much simpler than applying ACLs to block SNMP on all interfaces separately. Also, using interface ACLs would block not only SNMP packets intended for this router, but also would stop SNMP packets that just happened to be passing through on their way to some other destination device.

Despite the different syntax, both of the examples shown in this recipe work the same way. First, you define a standard access list. Then you apply this access list to your SNMP community string. You can assign each SNMP community a separate and unique access list to restrict access. This can be useful when there are several different people or NMS systems that need to access information on different groups of routers. You can also assign an access list to a read-write community string to block SNMP Set commands from unauthorized IP source addresses.

SNMP access lists alone are not an effective way of protecting read-write access to your routers. Because SNMP is a UDP protocol, a rogue device with access to your network can spoof the IP source address so that it matches the IP address of your management server. This means that somebody who knows your community string and the IP address of your management server can submit potentially dangerous SNMP Set commands to your router. SNMP sends packets in clear-text, so this information is relatively easy to discover with a protocol analyzer on a well-chosen network segment.



Applying a nonexistent access list to your SNMP configuration commands implicitly allows all IP source addresses to access your router's SNMP services. Always ensure that the access list exists before applying it to an SNMP community string.

The following command shows the status of a particular access list:

```
Router#show access-list 99
Standard IP access list 99
    permit 10.1.1.1 (1745 matches)
    permit 172.25.1.0, wildcard bits 0.0.0.255 (477 matches)
    deny   any (7 matches)
Router#
```

Note that the router has denied seven requests. This means that the router received and discarded seven SNMP requests because source addresses were not allowed. Although the router automatically appends an implicit *deny all* to the end of every access list, we recommend that you explicitly include a *deny all* statement at the end of each access list to let you keep track of denied packets like this. It is an easy way to tell how often your routers receive SNMP requests from bad source addresses.

The *show snmp group* command shows you which access lists are assigned to which SNMP community strings:

```
Router>show snmp group
groupname: ORARO                        security model:v1
readview :v1default                     writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active      access-list: 99

groupname: ORARW                        security model:v1
readview :v1default                     writeview: v1default
notifyview: <no notifyview specified>
row status: active      access-list: 98

Router>
```

In this example, the SNMP community string *ORARO* is protected by access list 99 and SNMP community string *ORARW* with access list 98. Note that the *show snmp group* command is only available in 12.0(3)T and above.

We discuss how to use the new SNMP syntax for read-write access in Recipes 17.7 and 17.21.

See Also

Recipe 17.1; Recipe 17.6; Recipe 17.7; Recipe 17.21; Chapter 19

17.6 Logging Unauthorized SNMP Attempts

Problem

You want to log unauthorized SNMP attempts.

Solution

Use the following commands to configure your router to log unauthorized SNMP requests:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 99 permit 172.25.1.0 0.0.0.255
Router(config)#access-list 99 permit host 10.1.1.1
Router(config)#access-list 99 deny any log
Router(config)#snmp-server community ORARO ro 99
Router(config)#snmp-server community ORARW rw 99
Router(config)#end
Router#
```

Discussion

If you are concerned about unauthorized access to SNMP services on your router, it can be quite useful to configure the router to maintain detailed records of every failed request. These verbose log messages can provide information on incorrectly configured management servers, as well as malicious (or just plain nosy) users.

Simply adding the keyword *log* to the *deny any* line in your access list instructs the router to log all unauthorized SNMP attempts.

The following command will display the status of your SNMP access list:

```
Router#show access-list 99
Standard IP access list 99
    permit 10.1.1.1  (1293 matches)
    permit 172.25.1.0, wildcard bits 0.0.0.255 (630 matches)
    deny   any log (17 matches)
Router#
```

Unlike the example shown in Recipe 17.5, the *show access-list* output now includes the *log* keyword on the *deny any* line. The router will now send information on every unauthorized SNMP request to the logging facility (see Chapter 18 for more information on logging). Use the *show logging* EXEC command to view the router's internal logging buffer:

```
Router#show logging
Syslog logging: enabled (0 messages dropped, 0 flushes, 0 overruns)
    Console logging: disabled
    Monitor logging: level debugging, 26 messages logged
        Logging to: vty2(0)
    Buffer logging: level debugging, 49 messages logged
    Trap logging: level informational, 53 message lines logged
        Logging to 172.25.1.1, 53 message lines logged
        Logging to 172.25.1.3, 53 message lines logged

Log Buffer (4096 bytes):
Apr 15 22:33:21: %SEC-6-IPACCESSLOGS: list 99 denied 192.168.22.13 1 packet
Apr 15 22:39:18: %SEC-6-IPACCESSLOGS: list 99 denied 10.121.212.11 3 packets
Router#
```

This example shows that access list 99, our SNMP access list, has denied access attempts by two IP source addresses, `192.168.22.13` and `10.121.212.11`. The final logging entry shows that the ACL denied three packets from source address `10.121.212.11`. Note that every packet received doesn't result in a separate log entry. If you are building a custom script to extract failed SNMP attempts, you will need to keep this in mind.

See Also

Recipe 17.5; Chapter 18

17.7 Limiting MIB Access

Problem

You want to limit which MIB variables can be remotely accessed with SNMP.

Solution

You can use the following commands to restrict SNMP access to portions of the MIB tree. This example shows the legacy configuration method:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 99 permit 172.25.1.0 0.0.0.255
Router(config)#access-list 99 deny any log
Router(config)#snmp-server view ORAVIEW mib-2 included
Router(config)#snmp-server view ORAVIEW at excluded
Router(config)#snmp-server view ORAVIEW cisco included
Router(config)#snmp-server community ORARO view ORAVIEW ro 99
Router(config)#snmp-server view RESTRICTED lsystem.55 included
Router(config)#snmp-server community ORARW view RESTRICTED rw 99
Router(config)#end
Router#
```

Cisco also has a new method for restricting MIB access, which uses the *snmp-server group* command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view ORAVIEW mib-2 included
Router(config)#snmp-server view ORAVIEW at excluded
Router(config)#snmp-server view ORAVIEW cisco included
Router(config)#snmp-server group TEST v1 read ORAVIEW
Router(config)#snmp-server user ORARO TEST v1
Router(config)#snmp-server view RESTRICTED lsystem.55 included
Router(config)#snmp-server group TEST2 v1 write RESTRICTED
Router(config)#snmp-server user ORARW TEST2 v1
Router(config)#end
Router#
```

Discussion

By default, enabling SNMP services on your router allows SNMP servers to access the entire SNMP MIB tree. However, you might want to limit which MIB variables can be remotely retrieved or changed, usually for security reasons. We strongly recommend that you limit SNMP write access to only those MIB objects that you absolutely need to change remotely. Remember that it is very easy for a malicious user to cause serious network problems by modifying MIB variables that control the router's configuration.

You can assign an SNMP MIB view to an individual community string or share a view among several community strings (including both read-only and read-write access strings). Assigning a MIB view to a read-only community string restricts which MIB variables can be displayed. Similarly, assigning an SNMP MIB view to a read-write community string restricts which MIB variables you can view or alter.

A MIB view can restrict access to a single MIB object, it can allow access to all but one MIB object, or anything in between. For instance, in both examples we created a view named *RESTRICTED* to the read-write community string *ORARW*. This view restricts access to a single MIB entry, *lsystem.55*, which is the MIB object that triggers the router to send its configuration file to a TFTP server (for nightly configuration backups). The router will prevent any other access to the MIB tree.

We also created an SNMP view named *ORAVIEW* which is less restrictive. In this case, we want to allow access to the MIB-2 variables but prevent access to the ARP table (AT) tree, which we can do by using the *exclude* keyword. At the same time, we allow access to the entire Cisco proprietary MIB tree by including the *cisco* MIB.

To illustrate the functionality of SNMP MIB views, here's an SNMP walk of a router's default MIB tree:

```
Freebsd% snmpwalk -v1 -c ORARO Router
system.sysDescr.0 = Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-JK9O3S-M), Version 12.2(7a), RELEASE SOFTWARE (fc2)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Thu 21-Feb-02 03:48 by pwade
system.sysObjectID.0 = OID: enterprises.9.1.209
system.sysUpTime.0 = Timeticks: (26809590) 3 days, 2:28:15.90
system.sysContact.0 = Ian Brown 416-555-2943
system.sysName.0 = Router.oreilly.com
system.sysLocation.0 = 999 Queen St. W., Toronto, Ont.
system.sysServices.0 = 78
system.sysORLastChange.0 = Timeticks: (0) 0:00:00.00
interfaces.ifNumber.0 = 10
interfaces.ifTable.ifEntry.ifIndex.1 = 1
interfaces.ifTable.ifEntry.ifIndex.2 = 2
interfaces.ifTable.ifEntry.ifIndex.3 = 3
interfaces.ifTable.ifEntry.ifIndex.4 = 4
interfaces.ifTable.ifEntry.ifIndex.5 = 5
interfaces.ifTable.ifEntry.ifIndex.6 = 6
interfaces.ifTable.ifEntry.ifIndex.7 = 7
interfaces.ifTable.ifEntry.ifIndex.8 = 8
interfaces.ifTable.ifEntry.ifIndex.9 = 9
<8000+ lines Removed>
End of MIB
Freebsd%
```

Walking the full MIB tree of a Cisco router can take a great deal of time. This router's MIB tree consisted of more than 8000 entries. However, if we implement a simple SNMP MIB view, the result is quite different:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TEST system.5 included
Router(config)#snmp-server community COOKBOOK view TEST ro
Router(config)#end
Router#
```

In this example, the router restricts access to a single MIB entry, *sysName* (system.5). Now when we attempt to walk the entire MIB tree again, the router sends only this single variable:

```
Freebsd% snmpwalk -v1 -c COOKBOOK Router
system.sysName.0 = Router.oreilly.com
End of MIB
Freebsd%
```

Note that the router displays a single entry, *sysName*, and reports that it has reached the "End of MIB," effectively preventing more than 8000 MIB objects from being accessed via this particular community string.

You can use the *show snmp group* EXEC command to see which views are assigned to which community string:

```
Router>show snmp group
groupname: ORARO                          security model:v1
readview :v1default                       writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active

groupname: COOKBOOK                       security model:v1
readview :TEST                            writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active

Router>
```

In this example, the community string *ORARO* has the default SNMP view, *v1default*. This means the entire MIB tree is accessible.

To see which MIB entries are assigned to which SNMP MIB view, use the following (hidden) command:

```
Router#show snmp view
ORAVIEW mib-2 - included nonvolatile active
ORAVIEW at - excluded nonvolatile active
ORAVIEW cisco - included nonvolatile active
v1default internet - included volatile active
v1default internet.6.3.15 - excluded volatile active
v1default internet.6.3.16 - excluded volatile active
v1default internet.6.3.18 - excluded volatile active
RESTRICTED cisco - included nonvolatile active
RESTRICTED lsystem.55 - included nonvolatile active
Router#
```

Table 17-3 lists a number of valid MIB trees that the router will accept within an SNMP view. Keep in mind that this is not an exhaustive list and that the router will also accept OIDs in their numerical format.

Table 17-3. Valid OID-trees for use with SNMP views

Keyword	Description
internet	Entire MIB tree
mib-2	Entire MIB-II tree
system	System branch of the MIB-II tree
interfaces	Interface branch of the MIB-II tree
at	ARP table branch of the MIB-II tree
ip	IP routing table branch of the MIB-II tree
icmp	ICMP statistics branch of the MIB-II tree
tcp	TCP statistics branch of the MIB-II tree
udp	UDP statistics branch of the MIB-II tree
transmission	Transmission statistics of the MIB-II tree
snmp	SNMP statistics branch of the MIB-II tree
ospf	OSPF MIB
bgp	BGP MIB
rmon	RMON MIB
cisco	Cisco's enterprise MIB tree
x25	X.25 MIB
ifEntry	Interface statistics MIB objects
lsystem	Cisco's system MIB

See Also

Recipe 17.1; Recipe 17.2

17.8 Using SNMP to Modify a Router's Running Configuration

Problem

You want to use SNMP to download or modify a router's configuration.

Solution

To upload or download a current copy of your router's configuration file to a TFTP server via SNMP, you have to first configure the router for read-write SNMP access:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server community ORARW rw
Router(config)#end
```

To download the current configuration file, you need to create an empty file on your TFTP server. In this case we assume a Unix server, but TFTP server software is available for essentially every popular operating system. Send an SNMP command to the router to trigger the TFTP download:

```
Freebsd% touch /tftpboot/router.cfg
Freebsd% chmod 666 /tftpboot/router.cfg
Freebsd% snmpset -v1 -c ORARW Router.1.3.6.1.4.1.9.2.1.55.172.25.1.1 s router.cfg
enterprises.9.2.1.55.172.25.1.1 = "router.cfg"
Freebsd%
```

You can use SNMP to trigger the router to upload a configuration file from your TFTP server via SNMP as follows:

```
Freebsd% echo "no ip source-route" > /tftpboot/new.cfg
Freebsd% echo "end" >> /tftpboot/new.cfg
Freebsd% chmod 666 /tftpboot/new.cfg
Freebsd% snmpset -v1 -c ORARW Router.1.3.6.1.4.1.9.2.1.53.172.25.1.1 s new.cfg
enterprises.9.2.1.53.172.25.1.1 = "new.cfg"
Freebsd% snmpset -v1 -c ORARW Router.1.3.6.1.4.1.9.2.1.54.0 i 1
enterprises.9.2.1.54.0 = 1
Freebsd%
```

Discussion

The ability to extract or modify your router's configuration via SNMP is powerful yet scary. These examples illustrate the power of SNMP read-write access and the main reason we advocate SNMP security features. We highly recommend that you read recipe 17.11 before allowing open SNMP write access on your routers. In that recipe we demonstrates an effective way to mitigate unauthorized tampering with your router's configuration files.

This first example illustrates how to extract your router's running configuration file to a TFTP server using SNMP. Before a typical TFTP server will accept a file transfer, a world-writable file must exist. On a Unix platform, the *touch* command creates this file, while the *chmod* command ensures that it has the proper file attributes.

The *snmpset* command instructs the router to send its running configuration file to a particular file on a particular TFTP server:

```
snmpset -v1 -c ORARW Router .1.3.6.1.4.1.9.2.1.55.172.25.1.1 s router.cfg
```

In this command, *Router* is the name (or IP address) of the router. The read-write SNMP community string is *ORARW*. The MIB OID value is actually in two parts. The first part, `.1.3.6.1.4.1.9.2.1.55`, is the OID value in the Cisco MIB extension that instructs the router to send its configuration file. The second part (`172.25.1.1`, in this case) is the IP address of your TFTP server, and *router.cfg* is the name of the file as it will appear on the TFTP server. The single letter *s* before the file name designates that the argument that follows will be a character string.

Extracting a router's configuration file like this is extremely useful. The Bourne shell script in Example 17-2 uses this method to extract and store the current configuration file from a Cisco router. The script automates the commands listed in the solution section to simplify the extraction of router configuration files. This script takes a single argument, the router name or IP address, and it stores the router configuration file in the */tftpboot* directory. The file will be the name of the router, with *.auto* appended to it (e.g., *router.auto*).

Example 17-2. conf

```
#!/bin/sh
#
#    conf -- A compact script to extract router configs to a
#            tftp server.
#
#
#  set behavior
snmprw="ORARW"
tftp="172.25.1.1"
#
#
router=$1
if [ "$router" = "" ]; then
echo "Usage: `basename $0` <hostname | ip address>" >&2 && exit 1
else
rm /tftpboot/$router-auto
touch /tftpboot/$router-auto
chmod 666 /tftpboot/$router-auto
snmpset="snmpset -v1 -c $snmprw $router "
$snmpset .1.3.6.1.4.1.9.2.1.55.$tftp s $router-auto
if [ -w /tftpboot/$router-auto -a -s /tftpboot/$router-auto ]; then
echo "Completed Successfully"
else
echo "Operation Failed"
fi
fi
```

Run this script as follows:

```
Freebsd% ./conf router
Completed Successfully
Freebsd%
```

This script assumes that NET-SNMP is on the server, and requires two variables to be set, *snmprw* and *tftp*. The *snmprw* variable contains the SNMP read-write community string of your organization and the *tftp* variable contains the IP address of your TFTP server.

The second example in the solution loads new configuration commands into a router. You must have a world-readable file containing these router configuration commands in your TFTP directory before you can upload anything. So in the example we have created a simple configuration file. We used echo commands to create the file, although in practice you should probably use a text editor to help limit the number of typing errors in your router's configuration. The last line in the configuration file should have the *end* command. This prevents the router from complaining about an unexpected end to the configuration file.

Note that when you upload a configuration file like this, the router merges the commands into its existing configuration, just as it does when you type the commands at the router's console.

There are two important differences between *snmpset* commands to upload or download a configuration file. The first is the different OID value. Be very careful that you get the right value here, because you don't want to accidentally upload an old configuration when you're trying to download. The second difference is that after uploading the configuration file, we issued another different *snmpset* command. This second command saves the configuration changes to NVRAM. This is the same as logging into the router and typing *write memory* or *copy running-config startup-config*.

See Also

Recipe 17.2; Recipe 17.5; Recipe 17.7; Appendix A

17.9 Using SNMP to Copy a New IOS Image

Problem

You want use SNMP to remotely upgrade a router's IOS.

Solution

Before you can upload or download the router's IOS image to a TFTP server, you have to set up a valid read-write SNMP community string:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server community ORARW rw
Router(config)#end
```

Then you can download a copy of your router's current IOS file to your TFTP server with the following Unix commands:

```
Freebsd% touch /tftpboot/c2600-jk9o3s-mz.122-7a.bin
Freebsd% chmod 666 /tftpboot/c2600-jk9o3s-mz.122-7a.bin
Freebsd% snmpset -v1 -c ORARW Router .1.3.6.1.4.1.9.2.10.9.172.25.1.1 s c2600-jk9o3s-
mz.122-7a.bin
enterprises.9.2.10.9.172.25.1.1 = "c2600-jk9o3s-mz.122-7a.bin"
Freebsd%
```

Use the following commands to upload an IOS file from your TFTP server to the router's flash memory:

```
Freebsd% chmod 666 /tftpboot/c2600-jk9o3s-mz.122-7a.bin
Freebsd% snmpset -v1 -c ORARW Router .1.3.6.1.4.1.9.2.10.6.0 i 1
enterprises.9.2.10.6.0 = 1
Freebsd% snmpset -v1 -c ORARW Router.1.3.6.1.4.1.9.2.10.12.172.25.1.1 s c2600-jk9o3s-
mz.122-7a.bin
enterprises.9.2.10.12.172.25.1.1 = "c2600-jk9o3s-mz.122-7a.bin"
Freebsd%
```

Discussion

The first example demonstrates how to use SNMP to force a router to download its IOS file to a TFTP server. Most TFTP servers will not accept an incoming transfer unless the destination file is world writable. On Unix computers, the *touch* command creates a file, while the *chmod* command gives it the proper file attributes.

This *snmpset* command instructs the router to use TFTP to copy its IOS file to a particular file on the specified server:

```
Freebsd% snmpset -v1 -c ORARW Router .1.3.6.1.4.1.9.2.10.9.172.25.1.1 s c2600-jk9o3s-
mz.122-7a.bin
```

In this case, *Router* is the router's name or IP address, and we use *ORARW* for the read-write SNMP community string. The OID value of the Cisco MIB variable that instructs the router to transfer its IOS image is `.1.3.6.1.4.1.9.2.10.9`. You concatenate the server's IP address (which is `172.25.1.1` in this example) to the end of the OID value. The last argument, *c2600-jk9o3s-mz.122-7a.bin*, is the name of the file as it will appear on the TFTP server.

This command is useful because it allows you to easily build a central library of all running IOS versions on your network. Then, if you have problems with a router and need to replace it, you can easily make sure that the new device is running the same IOS version as the old one. It is also useful if you discover that a particular IOS version behaves better and want to copy that version into other routers.

The second example shows how to use SNMP to start a TFTP upload of a new IOS version. This is useful because it makes it easy to build a script to automate changing the IOS versions on a large number of similar routers. You should be careful when doing this, however. It is safe to copy a new IOS image into the router's flash

memory on most Cisco routers. The router will continue running the old version until it reboots. A good procedure for doing a large number of upgrades like this is to run a script to copy the new images to the routers, then check each router to ensure that the upload completed successfully before rebooting.

The method shown for uploading a new IOS file to a router is similar to the method for downloading an IOS file. The one important difference is the step that erases the flash before the uploading commences. In our example, the first *snmpset* command erases the flash:

```
Freebsd% snmpset -v1 -c ORARW Router .1.3.6.1.4.1.9.2.10.6.0 i 1
enterprises.9.2.10.6.0 = 1
```

This step is necessary only if there is not enough flash space available to load the new IOS file. The second command actually copies the image into the router's flash memory:

```
Freebsd% snmpset -v1 -c ORARW Router.1.3.6.1.4.1.9.2.10.12.172.25.1.1 s c2600-jk9o3s-
mz.122-7a.bin
```

Note that some types of Cisco routers do not support this method for uploading IOS images. In particular, Cisco 2500 series routers actually run directly from the IOS image in flash instead of copying an image of the IOS into processor memory at boot time. Changing the IOS version in flash would cause serious problems, so the router will not allow you to do it.

See Also

Recipe 17.2; Recipe 17.5; Recipe 17.7; Chapter 1; Appendix A

17.10 Using SNMP to Perform Mass Configuration Changes

Problem

You want to automate the distribution of a set of configuration commands to a large number of routers.

Solution

The Perl script in Example 17-3 will distribute configuration commands to a large number of routers. It works using SNMP to trigger TFTP file transfers into the routers. In effect, this script lets you automatically distribute a series of configuration commands to a list of routers. Automating routine changes like this saves time and effort but more importantly, it virtually eliminates typing mistakes.

Here's some example output:

```
Freebsd% ./snmpcfg.pl
===================================
toronto - Update Successful
toronto - Wr Mem Successful
===================================
boston  - Update Successful
boston  - Wr Mem Successful
===================================
denver  - Update Successful
denver  - Wr Mem Successful
===================================
newyork - Update Successful
newyork - Wr Mem Successful
===================================
detroit - Update Failed
===================================
chicago - Update Successful
chicago - Wr Mem Successful
===================================
sanfran - Update Successful
sanfran - Wr Mem Successful
===================================
seattle - Update Successful
seattle - Wr Mem Successful
===================================
Freebsd%
```

Example 17-3 contains the Perl code.

Example 17-3. snmpcfg.pl

```
#!/usr/bin/perl -w
#
#    snmpcfg.pl -- a script to perform mass configuration changes to
#                  a list of routers using SNMP.
#
#
# Set behaviour
$workingdir="/home/nms";
$snmprw="ORARW";
$tftpsrv="172.25.1.1";
#
#
$rtrlist="$workingdir/RTR_LIST";
open (RTR, "$rtrlist") || die "Can't open $rtrlist file";
open (LOG, ">$workingdir/RESULT") || die "Can't open $workingdir/RESULT file";
#
while (<RTR>) {
  chomp($rtr="$_");
  print LOG "=================================== \n";
  print "=================================== \n";
  $snmpset="/usr/local/bin/snmpset -t 20 -r 2 -v1 -c $snmprw $rtr ";
  chomp($result=`$snmpset .1.3.6.1.4.1.9.2.1.50.$tftpsrv s SNMPCFG`);
```

Example 17-3. snmpcfg.pl (continued)

```
  if ($result=~/.+ = "(.+)"$/ ) {
            if( $1 eq SNMPCFG ) {
                print LOG "$rtr - Update Successful\n";
                print "$rtr - Update Successful\n";
                chomp($result=`$snmpset .1.3.6.1.4.1.9.2.1.54.0 i 1`);
                if ($result=~/.+ = (.+)$/ ) {
                          if( $1 == 1 ) {
                            print LOG "$rtr - Wr Mem Successful\n";
                            print "$rtr - Wr Mem Successful\n";
                          }
                          else {
                            print LOG "$rtr - Wr Mem Failed\n";
                            print "$rtr - Wr Mem Failed\n";
                          }
                }
                else {
                  print LOG "$rtr - Wr Mem Failed\n";
                  print "$rtr - Wr Mem Failed\n";
                }
            }
            else {
              print LOG "$rtr - Update Failed\n";
              print "$rtr - Update Failed\n";
            }
  }
  else {
    print LOG "$rtr - Update Failed\n";
    print "$rtr - Update Failed\n";
  }
}
```

Discussion

This script distributes a set of configuration commands to a list of routers using SNMP to trigger TFTP transfers, as we did manually in Recipe 17.8. The script goes through a list of routers in sequence, performing an *snmpset* command on each one to force the router to upload a predefined configuration file. If the file transfer completes successfully, the script will issue another *snmpset* command that permanently saves the running configuration file to NVRAM. The script displays a status report to the terminal screen and sends the same messages to a flat log file.

This script requires the NET-SNMP toolset. The script looks for the executable *snmpset* in the default location, */usr/local/bin*. If your system has *snmpset* in another location, change the variable *$snmpset*.

Before running the script, change the variable *$workingdir* to point to the directory where the script resides. Also set the variable *$snmprw* to your organization's SNMP read-write community string. This script will not work with a read-only community string. You will need to set the value of *$tftpsrv* to the IP address of the TFTP server where the configuration file resides.

The script expects to find the router list located in the working directory in a file called *RTR_LIST*. This file should have a single router name per line. You can change the default name and location of this file by modifying the variable *$rtrlist*.

By default, the script will copy the configuration file *SNMPCFG* (which is located in the */tftpboot* directory) to every router in the list. The configuration file must be world readable. This file should include a list of Cisco configuration commands as you would type them from a command prompt on the router. As we discussed in Chapter 1, we recommend inserting the keyword *end* at the end of the configuration file to prevent spurious error messages. If you want to change the filename, you will need to change both occurrences of the default filename *SNMPCFG* to the name of the new file.

The script creates a status report in a file called *RESULT* in the working directory. The script will automatically create this file the first time you execute it and will clear its contents each time the script is run. The status file allows you to run the script unattended and check for failures later. The easiest way to check for failures is to use the Unix *grep* utility to search the status report file for the keyword *Fail*.

See Also

Recipe 17.2; Recipe 17.5; Recipe 17.7; Recipe 17.8; Chapter 1; Appendix A

17.11 Preventing Unauthorized Configuration Modifications

Problem

You want to ensure that only authorized devices can use SNMP and TFTP to send or receive configuration information.

Solution

You can use the *snmp-server tftp-server-list* configuration command to restrict which TFTP servers the router can use in response to an SNMP trigger to upload or download configuration information:

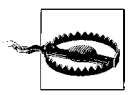
```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 92 permit 172.25.1.1
Router(config)#access-list 92 deny any log
Router(config)#snmp-server tftp-server-list 92
Router(config)#snmp-server community ORARW rw
Router(config)#end
Router#
```

Discussion

By default, the router will send or receive configuration information to any TFTP server. But this can be dangerous because the SNMP request that triggers these transfers cannot be 100% protected. Recipe 17.5 showed how you can restrict SNMP access to a specified list of devices. But, because SNMP uses UDP, it is not difficult for a malicious user to put the IP address of one of these allowed devices in the source of an SNMP packet, which will cause the router to execute the request. This packet could instruct the router to upload or download configuration information to or from any TFTP server. The attacker could then easily compromise the security of the entire network.

Therefore, we strongly recommend that you use the *tftp-server-list* command to restrict the TFTP servers to which your router will forward its configuration file and the TFTP servers from which your router will accept configuration changes.

It is important to note that this command restricts only TFTP sessions that the router initiates via SNMP. You can still use other TFTP servers for file transfers initiated from the router's command prompt.



If the access list assigned to the *tftp-server-list* does not exist, the router implicitly allows access for all TFTP servers.

The example authorizes the router to access only a single TFTP server. Note that the access list is designed to log all unauthorized attempts:

```
Router(config)#access-list 92 permit 172.25.1.1
Router(config)#access-list 92 deny any log
```

We highly recommend doing this because it not only prevents unauthorized access, but it also gives you information about what devices have been involved in the attempts. If there are malicious users with access to your network, this can help you to figure out who they are.

Note that this is a global command that affects all SNMP read-write community strings. There is no way to specify a different *tftp-server-list* for each community string.

See Also

Recipe 17.1; Recipe 17.5

17.12 Making Interface Table Numbers Permanent

Problem

You want to ensure that your router uses the same SNMP interface numbers every time it reboots.

Solution

To ensure that SNMP interface numbers remain permanent after a router power cycle, use the following command. This is a global command that affects all interfaces:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server ifindex persist
Router(config)#end
Router#
```

You can also fix the SNMP interface number of a single interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface Serial0/0
Router(config-if)#snmp ifindex persist
Router(config-if)#end
Router#
```

This command is available in IOS Versions 12.1(5)T and above.

Discussion

It comes as a surprise to most engineers that the internal SNMP interface numbers assigned by the router are not stable. The SNMP interface numbers are prone to change after router reboot, especially if you add or remove logical interfaces (i.e., subinterfaces) or physical modules.

This issue has plagued many administrators and software vendors for years. The problem is that most network performance software packages poll for interface data using the unique interface number assigned by the router. However, if these numbers change after a router reboots, then the performance data becomes meaningless because there is no guarantee that you are still polling the same interface. Most high-end SNMP performance software companies have built fixes to circumvent this exact issue.

Changing interface numbers particularly affects the router's built-in RMON monitoring. With RMON you can configure the router to monitor its own MIB values and assign threshold values for sending notifications. Unfortunately, before the new

functionality shown in this recipe came along, RMON polling was not reliable for interface-specific statistics. RMON services are discussed in detail in recipe 17.22.

There are some minor costs to using this feature. First, each interface number requires 25 bytes of NVRAM to store. Second, some administrators have reported slightly slower boot times on routers that employ this feature. Otherwise, this new functionality is mostly transparent to the network administrator.

To illustrate the *ifindex* stability problem, consider the interface numbers on a typical router:

```
Freebsd% snmpwalk -v1 -c ORARO Router ifDescr
interfaces.ifTable.ifEntry.ifDescr.1 = "BRI0/0"
interfaces.ifTable.ifEntry.ifDescr.2 = "Ethernet0/0"
interfaces.ifTable.ifEntry.ifDescr.3 = "BRI0/0:1"
interfaces.ifTable.ifEntry.ifDescr.4 = "BRI0/0:2"
interfaces.ifTable.ifEntry.ifDescr.5 = "FastEthernet1/0"
interfaces.ifTable.ifEntry.ifDescr.6 = "Null0"
interfaces.ifTable.ifEntry.ifDescr.7 = "Loopback0"
```

Note that the router assigns a unique number to each interface, starting with 1. In this example, the interface `FastEthernet1/0` has an *ifindex* value of 5. This is the number you would use in SNMP polls for various interface level performance statistics. Next, we will power down the router and remove the BRI module before restoring power:

```
Freebsd% snmpwalk -v1 -c ORARO Router ifDescr
interfaces.ifTable.ifEntry.ifDescr.1 = "Ethernet0/0"
interfaces.ifTable.ifEntry.ifDescr.2 = "FastEthernet1/0"
interfaces.ifTable.ifEntry.ifDescr.3 = "Null0"
interfaces.ifTable.ifEntry.ifDescr.4 = "Loopback0"
```

Note that the BRI interface entries are gone and the remaining interface numbers have completely changed. The `FastEthernet1/0` interface now appears as interface number 2. And, worse still, there is no interface number 5 at all. So if you had been doing performance analysis on this port, it would suddenly stop working.

Returning the router to its original state restores the original interface numbers:

```
Freebsd% snmpwalk -v1 -c ORARO Router ifDescr
interfaces.ifTable.ifEntry.ifDescr.1 = "BRI0/0"
interfaces.ifTable.ifEntry.ifDescr.2 = "Ethernet0/0"
interfaces.ifTable.ifEntry.ifDescr.3 = "BRI0/0:1"
interfaces.ifTable.ifEntry.ifDescr.4 = "BRI0/0:2"
interfaces.ifTable.ifEntry.ifDescr.5 = "FastEthernet1/0"
interfaces.ifTable.ifEntry.ifDescr.6 = "Null0"
interfaces.ifTable.ifEntry.ifDescr.7 = "Loopback0"
```

However, if we enable the *snmp ifindex persist* command before powering down the router and removing the BRI module, the only difference is that the three entries associated with the BRI interface are removed:

```
Freebsd% snmpwalk -v1 -c ORARO 172.25.1.8 ifDescr
interfaces.ifTable.ifEntry.ifDescr.2 = "Ethernet0/0"
```

```
interfaces.ifTable.ifEntry.ifDescr.5 = "FastEthernet1/0"
interfaces.ifTable.ifEntry.ifDescr.6 = "Null0"
interfaces.ifTable.ifEntry.ifDescr.7 = "Loopback0"
```

The remaining interfaces have retained their original interface numbers after the router reboot. In particular, the `FastEthernet1/0` interface is once again interface number 5, which means that all polled data will still be useful.

See Also

Recipe 17.22

17.13 Enabling SNMP Traps and Informs

Problem

You want the router to generate SNMP traps or informs in response to various network events.

Solution

The following configuration commands will enable your router to send unsolicited SNMP traps to a network management server:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server enable traps
Router(config)#snmp-server host 172.25.1.1 ORATRAP config entity envmon hsrp
Router(config)#snmp-server host nms.oreilly.com ORATRAP bgp snmp envmon
Router(config)#end
Router#
```

Notice that the *snmp-server host* command will accept either an IP address or a hostname.

Beginning with SNMPv2c, Cisco routers also support SNMP informs. To enable SNMP informs, use the following commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server enable informs
Router(config)#snmp-server host 172.25.1.1 informs version 2c ORATRAP snmp envmon
Router(config)#end
Router#
```

Discussion

SNMP traps originate from the router's agent and are sent via UDP (port 162) to the Network Management Station (NMS). Unlike the information that the router sends

to the NMS in response to an SNMP poll, a trap is unsolicited. The router's agent decides that something important has happened, and that it needs to tell the NMS about it. You must enable global trap support (see Table 17-4) and configure the trap host before the router agent can send traps.

SNMP traps are one of the basic elements of fault management. In fact, RFC 1812 ("Requirements for IP Version 4 Routers") states that all routers must be capable of sending SNMP traps.

Cisco routers can send a large variety of different SNMP traps, including both standard traps described in RFCs, and Cisco-specific traps. The first step in configuring trap support is to enable the particular trap types you wish to use. In our examples, we choose to enable all SNMP trap types using the configuration command *snmp-server enable traps*. The fact that we don't specify individual trap types implicitly enables all trap types. However, you can restrict the router to send only certain types of traps that you are interested in receiving. The various trap type keywords are shown in Table 17-4. Note that this is a global command that affects all SNMP trap receivers.

Table 17-4. Cisco SNMP trap types

Keyword	Description
bgp	Allow BGP state change traps
bstun	Allow bstun event traps
config	Allow SNMP configuration traps
dlsw	Allow DLSw traps
dsp	Allow SNMP DSP traps
dspu	Allow dspu event traps
entity	Allow SNMP entity traps
envmon	Allow environmental monitor traps
frame-relay	Allow SNMP Frame Relay traps
hsrp	Allow SNMP HSRP traps
ipmulticast	Allow SNMP IP multicast traps
isdn	Allow SNMP ISDN traps
msdp	Allow SNMP MSDP traps
rsrb	Allow rsrb event traps
rsvp	Allow RSVP flow change traps
rtr	Allow SNMP Response Time Reporter (RTR) traps
sdlc	Allow SDLC event traps
sdllc	Allow SDLLC event traps
snmp	Allow SNMP-type notifications
stun	Allow stun event traps

Table 17-4. Cisco SNMP trap types (continued)

Keyword	Description
syslog	Allow SNMP syslog traps
tty	Allow TCP connection traps
udp-port	The server host's UDP port number
voice	Allow SNMP voice traps
x25	Allow X25 event traps
xgcp	Allow XGCP protocol traps

For example, you could use the following commands to tell the router to send only BGP and environmental type traps:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server enable traps bgp
Router(config)#snmp-server enable traps envmon
Router(config)#end
Router#
```

You can also disable a particular type of SNMP trap with the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#no snmp-server enable traps envmon
Router(config)#end
Router#
```

The following command displays which SNMP trap types are enabled on a router:

```
Router#show running-config | include snmp-server enable
snmp-server enable traps snmp authentication linkdown linkup coldstart warmstart
snmp-server enable traps hsrp
snmp-server enable traps config
snmp-server enable traps entity
snmp-server enable traps envmon
snmp-server enable traps bgp
snmp-server enable traps ipmulticast
snmp-server enable traps msdp
snmp-server enable traps rsvp
snmp-server enable traps frame-relay
snmp-server enable traps syslog
snmp-server enable traps rtr
snmp-server enable traps dlsw
snmp-server enable traps dial
snmp-server enable traps dsp card-status
snmp-server enable traps voice poor-qov
Router#
```

The second step in configuring SNMP traps is to define the trap recipient using the *snmp-server host* command. This command has the following attributes:

snmp-server host ***host-addr*** **[traps | informs] [version {1 | 2c}]** ***community-string*** **[udp-port** *port] [trap-type*]

The *host-addr* argument is the name or IP address of the NMS server that will receive the traps. You can define whether the router will send SNMP traps or informs to this host by specifying either the *traps* or *informs* keyword. If neither is specified, the default is to send traps. Also, you can specify which version of SNMP traps the router will send by including either *version 1* or *version 2c*. If neither version is specified, the router will default to Version 1. Note that informs don't exist in SNMP Version 1, so you must specify Version 2c (or Version 3) if you want to enable this feature.

The *community-string* argument specifies the community string that the router will send within the SNMP trap or inform. This doesn't need to match either the read-only or read-write community strings on the router.

You can change the default SNMP trap port from 162 (the default) to another value with the optional *udp-port* keyword. The alternative UDP port number that you want to use must follow this keyword.

Finally, if the *trap-type* keyword is present, it allows you to configure the types of traps that the router will send to this server. The command can accept one or more types. However, if no trap types are included, the router will default to sending every enabled trap type.

There are two important things to note about this command. First, you must enable trap types via the global command before you can specify them for a particular host. Second, this command will allow you to send different sets of traps to different servers. This can sometimes be useful if you have multiple NMS servers that handle different management functions.

The configuration for SNMP informs is almost the same as that for SNMP traps. The main difference is that you can't enable individual inform types using the global *snmp-server enable informs* command. The global inform command lacks the granularity of the same trap-based command. However, you can still enable specific inform types on the host-level command. This can mean more typing if there are several inform recipients, but no functionality is lost.

See Also

Recipe 17.21; RFC 1812

17.14 Sending syslog Messages as SNMP Traps and Informs

Problem

You want to send syslog messages as SNMP traps or informs.

Solution

You can configure the router to forward syslog messages to your network management server as SNMP traps instead of syslog packets with the following configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging history informational
Router(config)#snmp-server enable traps syslog
Router(config)#snmp-server host 172.25.1.1 ORATRAP  syslog
Router(config)#end
Router#
```

To forward syslog messages as SNMP informs, use the *snmp-server* configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging history informational
Router(config)#snmp-server enable informs
Router(config)#snmp-server host 172.25.1.1 informs version 2c ORATRAP syslog
Router(config)#end
Router#
```

Discussion

Cisco routers normally forward syslog messages via the syslog facility using UDP port 514. However, in networks that support SNMP traffic only, Cisco routers can encapsulate their syslog messages into SNMP traps before sending them.

This feature is most useful if your network management software doesn't support the syslog protocol. However, since routers can produce many more syslog messages than SNMP traps, we recommend using syslog where possible. Further, the fact that all of the syslog messages sent as SNMP traps use the same OID number can make parsing for particular log messages quite difficult.

Here is an example log message as it appears in the router's log:

```
Router#clear counters
Clear "show interface" counters on all interfaces [confirm]
Router#
May 28 10:07:04: %CLEAR-5-COUNTERS: Clear counter on all interfaces by ijbrown on
vty0 (172.25.1.1)
```

The router sends this message as a trap to the network management server, which records it in its trap log:

```
Freebsd% tail snmptrapd.log
May 28 10:07:04 freebsd snmptrapd[77759]: 172.25.25.1: Enterprise Specific Trap (1)
Uptime: 18 days, 22:35:26.99, enterprises.9.9.41.1.2.3.1.2.118 = "CLEAR",
enterprises.9.9.41.1.2.3.1.3.118 = 6, enterprises.9.9.41.1.2.3.1.4.118 = "COUNTERS",
```

```
enterprises.9.9.41.1.2.3.1.5.118 = "Clear counter on all interfaces by ijbrown on
vty0 (172.25.1.1)", enterprises.9.9.41.1.2.3.1.6.118 = Timeticks: (163652698) 18
days, 22:35:26.98
Freebsd%
```

In this example, we forced the router to create a log message by clearing the interface counters. The router displayed the raw syslog message to the VTY session. The same information appears in the server's *snmptrapd.log* file. This is a flat file that NET-SNMP uses to store all SNMP traps forwarded to the server. Other network management systems store trap information in different formats and with different filenames.

You can also configure the router to forward syslog messages as SNMP informs. The result is the same as for traps. For more information on syslog (and logging in general), refer to Chapter 18.

See Also

Recipe 17.13; Chapter 18

17.15 Setting SNMP Packet Size

Problem

You want to change the default SNMP packet size.

Solution

The following configuration command adjusts the default packet size for all SNMP packets leaving the router:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server packetsize 1480
Router(config)#end
Router#
```

Discussion

By default, Cisco routers limit their SNMP packet size to 1500 bytes. It is usually not necessary to change this parameter. However, it may be useful to reduce it if your network has an MTU of less than 1500 bytes, to prevent unnecessary fragmentation. The conventional wisdom on UDP packets is that smaller is usually better than larger because of fragmentation.

Conversely, if your network media can accepts a larger MTU than 1500 bytes, increasing your SNMP packet size can improve performance, particularly when transferring large MIB tables.

Note that adjusting the maximum SNMP packet size will affect all types of SNMP packets, including responses to SNMP get or set requests, as well as SNMP traps and SNMP informs. You can set the SNMP packet size to any integer between 484 and 17,940 bytes.

17.16 Setting SNMP Queue Size

Problem

You want to increase the size of a router's SNMP trap queue.

Solution

The following command increases the size of a router's SNMP trap queue:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server queue-length 25
Router(config)#end
Router#
```

To increase the size of the router's SNMP inform queue, use the following configuration command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server inform pending 40
Router(config)#end
Router#
```

Discussion

By default, the router can hold 10 trap messages in its queue. The queue holds traps until the router can forward them to the NMS. The queue fills when the router generates traps faster than it can forward them. If it generates additional traps when the queue is already full, these new trap messages are dropped. The router has only one SNMP message queue for all trap recipients.

Regardless of the network's capacity, the router will never send SNMP messages faster than four traps per second. This rate is hardcoded into the router and is not configurable. If you have several NMS systems, or if your router creates a particularly large number of traps, you might want to increase the size of this queue to help prevent inadvertent discarding of traps.

The *snmp-server queue-length* command will accept any integer between 1 and 1000, representing the maximum number of packets that can be held at a time.

To show the current number of SNMP messages in the queue, and the maximum queue size, use the *show snmp* EXEC command:

```
Router#show snmp
Chassis: JAX123456789
Contact: Ian Brown 416-555-2943
Location: 999 Queen St. W., Toronto, Ontario
270 SNMP packets input
    0 Bad SNMP version errors
    12 Unknown community name
    0 Illegal operation for community name supplied
    0 Encoding errors
    231 Number of requested variables
    25 Number of altered variables
    11 Get-request PDUs
    222 Get-next PDUs
    25 Set-request PDUs
584 SNMP packets output
    0 Too big errors (Maximum packet size 1480)
    2 No such name errors
    0 Bad values errors
    0 General errors
    258 Response PDUs
    326 Trap PDUs

SNMP logging: enabled
    Logging to 172.25.1.1.162, 0/25, 309 sent, 17 dropped.
Router#
```

In this example, the `0/25` in the last line means that no SNMP traps are currently queued for transmission, and the queue can accept up to 25 messages at once. If you see the number of dropped SNMP traps growing rapidly over time, you should consider increasing the queue size. In this case, the "Trap PDUs" line tells you that the router has tried to send 326 traps. Of these, the last line tells you that it has successfully sent 309 and dropped 17. This is a 5% drop rate, which suggests that the queue depth should probably be increased.

SNMP informs maintain a queue for messages pending acknowledgement. Each inform is held in the pending queue until an acknowledgement is received. Consequently, the inform queue must be considerably larger than the corresponding trap queue.

If you choose to implement SNMP informs, we highly recommend that you increase the size of the pending queue from the default value of 25:

```
Router(config)#snmp-server inform pending 40
```

The router will accept any integer between 1 and 4,294,967,295 representing the number of unacknowledged informs to hold. There is very little benefit to using informs if the size of the pending queue is too small, because the router will not be able to hold new messages.

17.17 Setting SNMP Timeout Values

Problem

You want to adjust the SNMP trap timeout value.

Solution

You can use the following configuration command to adjust a router's SNMP trap timeout value:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server trap-timeout 60
Router(config)#end
Router#
```

To adjust a router's SNMP inform timeout value, use the following configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server inform timeout 120
Router(config)#end
Router#
```

Discussion

Before a router can send an SNMP trap, it must have a route to the destination address of the trap. If a route to the SNMP server does not exist, the router will store the trap in the retransmission queue. By default, the router will hold a trap in the retransmission queue for 30 seconds before attempting to deliver it again. Sometimes it is useful to modify the default wait time to improve the chances of successful delivery.

For instance, if the router has to send the trap over a low-speed dial backup interface, 30 seconds may not be enough time for it to trigger a call, establish connectivity, and stabilize a routing table. In a situation like this, you should consider increasing the trap timeout. The value is specified as an integer number of seconds between 1 and 1000.

SNMP informs use timeouts differently. The inform timeout is the number of seconds that the router will wait for an acknowledgement before resending. The default value is 30 seconds but the router will accept any value between 0 and 4,294,967,295 seconds.

Note that increasing the timeout values for traps and informs means that the router will tend to hold these messages for longer. This, in turn, generally means that you will have to increase the queue size in order to hold them.

17.18 Disabling Link Up/Down Traps per Interface

Problem

You want to disable link up/down traps for specific interfaces.

Solution

To disable SNMP link status-change traps for a particular interface, use the following configuration command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface Serial0/0
Router(config-if)#no snmp trap link-status
Router(config-if)#end
Router#
```

Discussion

By default, the router forwards SNMP link up or down traps whenever an interface changes states. Normally, you want to receive traps when an interface changes states, because that could indicate a serious problem. But there are times when it is useful to disable these types of traps. For instance, dial interfaces may cycle up and down throughout the day without cause for concern. In these cases, you will probably want to suppress these types of traps to prevent network management staff from needlessly chasing meaningless failure reports.

It is also useful to disable SNMP traps for link up and down messages when you are troubleshooting, testing, or enabling interfaces to prevent extraneous failure reports.

This command will work only on physical interfaces and loopback interfaces. It will not allow you to disable link status traps on subinterfaces.

See Also

Recipe 17.13

17.19 Setting the IP Source Address for SNMP Traps

Problem

You want to set the source IP address for all SNMP traps leaving a router.

Solution

To set the default IP source address for all traps leaving a router, use the following configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server host 172.25.1.1 ORATRAP
Router(config)#snmp-server trap-source loopback0
Router(config)#end
Router#
```

Discussion

Normally, when you enable SNMP traps to a remote server, that server will see the source IP address of the router's closest interface. However, this is not always meaningful. For instance, it is relatively common practice to populate your DNS with only the router's loopback interfaces. In this case, the server will not be able to resolve the originator of the trap.

Further, it can be difficult to correlate traps from the same router delivered through different interfaces. For example, this could happen as a result of a network failure. It can be confusing to see a link down message coming from one IP address and the corresponding link up message from a different one.

By enabling the *snmp-server trap-source* command, you can force the router to always use the same IP source address for all of the SNMP traps it sends. Industry best practices dictate that a loopback interface is usually the best choice for this because it is a virtual interface that is always available. Physical interfaces such as Ethernet or serial interfaces can become unavailable and limit the usefulness of this command. If you set the source interface to an unreachable interface, the router will resort to using the closest interface as the source address.

Note that Cisco's IOS will even allow you to assign a trap source interface without having an IP assigned address to it. However, the router will forward a syslog message highlighting the issue and resort to the default method of using the closest interface address for sending traps. Here is an example of the log message that appears in this case:

```
 Jun 12 00:22:00 EDT: %IP_SNMP-4-NOTRAPIP: SNMP trap source Loopback1 has no ip
address
```

There is not yet an equivalent command for SNMP informs.

17.20 Using RMON to Send Traps

Problem

You want the router to send a trap when the CPU rises above a threshold, or when other important events occur.

Solution

You can configure a router to monitor its own CPU utilization and trigger an SNMP trap when the value exceeds a defined threshold with the following set of configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#rmon event 1 log trap ORATRAP description "CPU on Router has exceeded
threshold" owner ijbrown
Router(config)#rmon event 2 log description "CPU on Router has normalized" owner
ijbrown
Router(config)#rmon alarm 1 lsystem.57.0 60 absolute rising-threshold 70 1 falling-
threshold 40 2 owner ijbrown
Router(config)#end
Router#
```

The following commands will configure the router to monitor its own buffer failures and send an SNMP trap when the number of failures exceeds a threshold:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#rmon event 3 log trap ORATRAP description "Excessive buffer failures
on Router" owner ijbrown
Router(config)#rmon alarm 2 lsystem.46.0 300 delta rising-threshold 5 3 falling-
threshold -1 3 owner ijbrown
Router(config)#end
Router#
```

To configure a router to monitor its own memory utilization and trigger an SNMP trap when it exceeds its threshold, use the following set of configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#rmon event 4 log trap ORATRAP description "Low memory condition on
Router" owner ijbrown
Router(config)#rmon event 5 log trap ORATRAP description "Low Memory condition
cleared on Router" owner ijbrown
Router(config)#rmon alarm 3 lsystem.8.0 60 absolute rising-threshold 1500000 5
falling-threshold 1000000 4 owner ijbrown
Router(config)#end
Router#
```

In this example, the router is configured to monitor the link utilization of a single interface and trigger an SNMP trap when that interface exceeds its threshold:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#rmon event 6 log trap ORATRAP description "Bandwidth utilization has
exceeded threshold on Router interface Serial 0/0" owner ijbrown
Router(config)#rmon event 7 log trap ORATRAP description "Bandwidth utilization has
normalized on Router interface Serial 0/0" owner ijbrown
Router(config)#! Configure inbound alarm on Serial0/0 (ifNumber 3)
Router(config)#rmon alarm 4 lifEntry.6.3 300 absolute rising-threshold 1000000 6
falling-threshold 800000 7 owner ijbrown
Router(config)#! Configure outbound alarm on Serial0/0 (ifNumber 3)
Router(config)#rmon alarm 5 lifEntry.8.3 300 absolute rising-threshold 1000000 6
falling-threshold 800000 7 owner ijbrown
Router(config)#end
Router#
```

To configure a router to monitor the number of interface errors on a particular interface, use the following set of commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#rmon event 7 log trap ORATRAP description "ifErrors have exceeded
threshold" owner ijbrown
Router(config)#rmon alarm 6 ifEntry.14.3 300 delta rising-threshold 5 7 falling-
threshold -1 7 owner ijbrown
Router(config)#rmon alarm 7 ifEntry.20.3 300 delta rising-threshold 5 7 falling-
threshold -1 7 owner ijbrown
Router(config)#end
Router#
```

If you choose to monitor interface-specific MIBS, such as utilization or errors, then we strongly suggest that you implement Cisco's interface *persist* command, which we discuss in Recipe 17.12.

Discussion

IOS includes some extremely helpful but seldom used remote monitoring (RMON) functionality. The main advantages to configuring the router to monitor its own performance are resource savings and flexibility.

The more conventional alternative is to have a centralized performance server that uses SNMP to periodically poll routers for health statistics. On a large network, this can consume a lot of bandwidth. However, if you move the polling functionality into the router itself, you can get the same benefits without the same bandwidth requirements.

The second major advantage of having the router monitor its own health is flexibility. You can configure each router to monitor itself using the thresholds and parameters that are most meaningful to that router. Since most performance monitoring

packages on the market today limit the amount of flexibility in assigning thresholds, this built-in monitoring technique can often give more reliable results.

Of course, the biggest advantage of using RMON is that it is readily available in Cisco's IOS and requires no extra software. If your management software is capable of accepting SNMP traps, no further tools are required. For the budget-conscious administrator, RMON provides detailed performance threshold management without the added cost of performance software.

Cisco's RMON module comes standard in Cisco IOS and is included in the base IP Only feature set. The purpose of RMON is to monitor a certain MIB object on the router and notify the system administrator if the object's value leaves a defined range. It does this by polling itself internally for the values of these MIB objects. You can configure the internal polling interval.

To prevent unnecessary trouble notifications, Cisco RMON alarms use a concept of rising and falling thresholds. An RMON event fires when a value of an SNMP object exceeds the value assigned to the rising threshold. However, subsequent polls that exceed the rising threshold will not trigger an event firing until a polled value drops below the falling threshold value. This concept of rising and falling thresholds prevents the agent from flooding the server with redundant events.

Our examples show how to configure RMON to monitor CPU utilization, low memory situations, buffer failures, link utilization, and interface error counts. This is far from an exhaustive list of the RMON capabilities. Since the router's RMON capabilities rely on the polling of SNMP MIB objects, you can configure it to monitor anything that has an MIB variable. The potential number of statistics you can configure the router to watch is enormous.

The first step in configuring RMON on a router is to define the RMON event(s) that you wish to monitor. In our first RMON example, we configured the router to monitor CPU utilization. The follow two commands tell the router what to do when the CPU load rises above the threshold, and what to do when it falls back into the normal range:

```
Router(config)#rmon event 1 log trap ORATRAP description "CPU on Router has exceeded
threshold" owner ijbrown
Router(config)#rmon event 2 log description "CPU on Router has normalized" owner
ijbrown
```

The following IOS command defines an RMON event:

```
rmon event number [log] [trap community] [description string] [owner string]
```

This *number* keyword assigns a unique identification number for the RMON event. RMON event numbers range between 1 and 65,535. The optional *log* keyword tells the agent to create a log entry, as well as an SNMP trap, when event is triggered.

A community string accompanies the optional trap keyword. This configures the agent to send an SNMP trap when the event is fired, and assigns the SNMP community string for the trap.

You can specify an optional description for an RMON event using the *description* keyword. The text that follows this keyword is sent with the trap.

This *owner* keyword specifies the owner of an event. In our examples the owner is *ijbrown*. If you do not have AAA usernames configured on your router, you should use the default *admin* user ID.

In this example, RMON Event 1 creates a log entry and an SNMP trap when the CPU utilization on the router exceeds a certain threshold. RMON Event 2 creates a log entry, but doesn't send a trap when the CPU utilization returns to normal.

To view the configured RMON events, use the following command:

```
Router>show rmon events
Event 1 is active, owned by ijbrown
 Description is CPU on Router has exceeded threshold
 Event firing causes log and trap to community ORATRAP, last fired 00:00:00
Event 2 is active, owned by ijbrown
 Description is CPU on Router has normalized
 Event firing causes log, last fired 2w2d
 Current log entries:
      index       time   description
          1       2w2d   CPU on Router has normalized
Router>
```

To make the RMON events meaningful, you also must create a corresponding RMON alarm that defines the conditions for each of the event conditions that we just discussed:

```
Router(config)#rmon alarm 1 lsystem.57.0 60 absolute rising-threshold 70 1 falling-
threshold 40 2 owner ijbrown
```

The syntax of this command is as follows:

```
rmon alarm number variable interval {absolute | delta}
rising-threshold value [event-number]
falling-threshold value [event-number] [owner string]
```

The *number* in this command is simply a value between 1 and 65,535 that uniquely identifies this alarm.

You can specify a particular MIB object that you want the router to monitor with the *variable* field. In this example, we are polling the Cisco CPU utilization MIB entry, *lsystem.57.0*.

The next value, *interval*, tells the router how often you want it to poll the MIB object. The value can be between 1 and 4,294,967,295 seconds. This example tells the router to poll on a 60-second cycle. Bear in mind that polling too frequently will

affect your router's CPU utilization, while polling too slowly might cause you to miss important events.

Once the data has been polled, you have the option of treating the value as an *absolute* or *delta*. In this example, we are interested in the actual value of the MIB variable, so we specified the *absolute* keyword. The keyword *delta* tells the router to take the numerical difference between the current sample and the previous one. This feature allows you to determine when a variable's rate of change is too fast or too slow.

You can set a rising threshold value using the *rising-threshold* keyword, which takes an argument between -2,147,483,648 and 2,147,483,647. In this example, the rising threshold is 70%, meaning that the router will trigger an event when the CPU utilization rises through 70%. When this happens, you want the router to trigger a particular event. This association with a particular event is made with the *event-number* argument. In this example, if the rising threshold is exceeded, RMON Event 1 will be triggered.

You configure a falling threshold similarly using the *falling-threshold* keyword. In this example, the falling threshold is 40%, meaning that if the CPU utilization dips below 40%, an event is triggered. As with rising thresholds, you can associate this alarm with a particular event number. In this example, the falling threshold triggers Event 2.

Finally, you can specify an owner for an alarm by using the *owner* keyword.

The example configures RMON Alarm 1 to poll the MIB entry for CPU utilization every minute. If the utilization exceeds 70%, the router will send a trap and create a log message. When the utilization dips back below 40%, it will create a log message.

To view the configured RMON alarms, use the following command:

```
Router>show rmon alarms
Alarm 1 is active, owned by ijbrown
 Monitors lsystem.57.0 every 60 second(s)
 Taking absolute samples, last value was 0
 Rising threshold is 70, assigned to event 1
 Falling threshold is 40, assigned to event 2
 On startup enable rising or falling alarm
Router>
```

Cisco's RMON functionality currently supports SNMP traps, but not SNMP informs.

See Also

Recipe 17.13

17.21 Enabling SNMPv3

Problem

You want to enable SNMPv3 on your router for security purposes.

Solution

SNMPv3 supports three modes of operation, each with different security features. These modes were summarized in Table 17-1 at the beginning of this chapter. The following configuration commands enable SNMPv3 with no authentication and no encryption services (*noAuthNoPriv*):

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TESTV3 mib-2 include
Router(config)#snmp-server group NOTSAFE v3 noauth read TESTV3
Router(config)#snmp-server user WEAK NOTSAFE v3
Router(config)#end
Router#
```

Use the following configuration commands to enable SNMPv3 with MD5 authentication and no encryption services (*authNoPriv*):

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TESTV3 mib-2 include
Router(config)#snmp-server group ORAROV3 v3 auth read TESTV3
Router(config)#snmp-server user cking ORAROV3 v3 auth md5 daytona19y
Router(config)#end
Router#
```

And you can enable SNMPv3 with MD5 authentication and DES encryption services (*authPriv*) as follows:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TESTV3 mib-2 include
Router(config)#snmp-server group ORAROV3 v3 auth read TESTV3
Router(config)#snmp-server user bpugsley ORAROV3 v3 auth md5 hockeyrules priv des56
shortguy
Router(config)#end
Router#
```

Discussion

The IETF has recently approved SNMP Version 3 (SNMPv3) as a full standard and moved SNMPv1 and SNMPv2 to historic status. Essentially, SNMPv3 just acts like a set of security extensions to SNMPv2c, without providing much new core management functionality. All MIB objects and their associated OIDs remain the same from

Versions 1 to 3 (with the small exception of the 64-bit counters that were introduced in Version 2). We will focus our attention on the new security features in Version 3.

Security has traditionally been the Achilles heel of SNMP. The security model for Versions 1 and 2c was little more than a simple password sent through the network as clear-text. SNMP required a security facelift to continue to be useful in the future.

SNMPv3 is standards-based network management protocol that is interoperable between vendors. It provides a secure access to devices by providing authentication and encryption of SNMP packets throughout the network. To do this, SNMPv3 needed to include the following security features: authentication, message integrity, and encryption

- *Authentication* ensures that the messages have originated from a valid source. It proves the authenticity of the packet's source.
- *Message integrity* ensures that a packet has not been tampered with during transmission.
- *Encryption* encodes the contents of the packet to prevent unauthorized people from viewing them.

SNMPv3 provides three security levels: *noAuthNoPriv*, *authNoPriv*, and *authPriv*.

- *noAuthNoPriv* uses a username for authentication and most closely emulates the SNMPv1 and SNMPv2c authentication schemes of transmitting credentials in clear-text. We do not recommend this level of SNMPv3, because it provides no significant advantage over SNMPv2c. If the advanced security features of SNMPv3 are not required for your implementation, it is probably easier to use SNMPv1 or SNMPv2c.
- *authNoPriv* provides authentication based on the MD5 or SHA algorithms. This security model provides packet authentication and message integrity, but no encryption services. Since SNMP packets are authenticated and cannot be altered in transit, this level of security is sufficient for most organizations.
- *authPriv* provides the same MD5 or SHA authentication as *authNoPriv*. In addition, *authPriv* allows you to encrypt SNMP packets using 56-bit DES encryption, so packet contents cannot be viewed without authorization. This provides the maximum security available by combining authentication, message integrity, and encryption. The *authPriv* level of security is suitable for implementations that need to send SNMP packets through the public Internet.

All three SNMPv3 security models require the same three-step process to configure them. First, you must define an SNMP view. Second, you need to create an SNMP group. Third, you create an SNMP user profile and assign it to a group.

Defining an SNMP view for SNMPv3 is no different that creating a view for SNMPv1 or SNMPv2c. In fact, if there are existing SNMP views on the router that were created for SNMPv1 or SNMPv2c, you can use them with SNMPv3 as well. For more information on creating SNMP views, please see Recipe 17.7.

For example, here is a simple SNMP view that allows full access to the MIBII tree:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TESTV3 mib-2 include
Router(config)#end
Router#
```

To define an SNMPv3 group, use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server group ORAROV3 v3 auth read TESTV3
Router(config)#end
Router#
```

In this example, we have created a group named *ORAROV3*, which we have configured as an SNMPv3 group (hence the *v3*). We have configured this group to require authentication and assigned it to SNMP view *TESTV3*. Note that we have not assigned a write view to this group, which means that all users assigned to this group will be limited to read-only access. However, the *snmp-server group* command will also allow you to define a read and a write view at the same time. For example:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TESTRO mib-2 include
Router(config)#snmp-server view TESTRW system include
Router(config)#snmp-server group TESTGRP v3 auth read TESTRO write TESTRW
Router(config)#end
Router#
```

In this example, we defined two separate SNMP views (*TESTRO* and *TESTRW*) and assigned them to our group. Note, however, that you can assign the same SNMP view to both the read-only access and read-write groups.

To define an SNMPv3 user, use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server user bpugsley ORAROV3 v3 auth md5 hockeyrules priv des56
shortguy
Router(config)#end
Router#
```

In this example, we have created a user named *bpugsley* and assigned that user to our group named *ORAROV3*. This user will inherit the qualities that we have configured for that group. We have also defined that our user will use the MD5 algorithm for authentication purposes and assigned an authentication password of *hockeyrules*. We have also configured our user to use the optional DES56 packet encryption with the password *shortguy* to provide maximum security. Note that this command, once entered, will not be viewable using the *show running-config* command. We suspect that this is for security purposes.

To view existing SNMP groups, use the *show snmp group* command:

```
Router#show snmp group

groupname: ORAROV3                      security model:v3 auth
readview :TESTV3                        writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active

Router#
```

In this example, the group *ORAROV3* is assigned to the security model *v3 auth*, the read-only view is *TESTV3*, and no read-write view exists.

To view the configured SNMPv3 users, use the following command:

```
Router#show snmp user
User name: bpugsley
Engine ID: 800000090300000196708770
storage-type: nonvolatile        active

Router#
```

Unfortunately, this command provides very little useful information. Apart from confirming if a user exists or not, the output does not display which group the user belongs to, or if the user is configured to use authentication or encryption. When you consider that Cisco's IOS also hides the user SNMP commands from the running configuration, it becomes clear that managing SNMPv3 users is a difficult task. We hope that Cisco will change the output of this command in upcoming releases as SNMPv3 becomes more popular.

Using the SNMPv3 Security Levels

You can extract SNMP information from the router using each of the three SNMPv3 security levels. We will use NET-SNMP's *snmpget* command, which has full SNMPv3 support.

In our first example (*noAuthNoPriv*), we will poll the router for its system name using a standard MIB-II object, *sysName*.

```
Freebsd% snmpget -v3 -u WEAK -l noAuthNoPriv  Router  sysName.0
system.sysName.0 = Router.oreilly.com
Freebsd%
```

Note that no user password was supplied, so the router simply accepted the user ID *WEAK* for authentication purposes. This user ID was sent through the network in clear-text. This command has also introduced two new attributes for the *snmpget* command: *-u* and *-l*. The *-u* attribute allows you to specify the security name, while *-l* defines the security level.

The next example uses the *authNoPriv* security model. We will poll the exact same MIB object using MD5 authentication:

```
Freebsd% snmpget -v3 -u cking -l authNoPriv -a MD5 -A daytona19y Router sysName.0
system.sysName.0 = Router.oreilly.com
Freebsd%
```

In this example, we specify a user password *daytona19y* using the *-A* option and an the authentication protocol *MD5* with the *-a* option. SNMPv3 uses the authentication protocol to authenticate users without sending the password in clear-text. It is important to notice that the result of this SNMP Get is the same as our first example. However, we gathered the information in a much more secure manner. In fact, the same MIB object (*sysName*) could be retrieved using SNMPv1 if the router was configured to accept the request. However, this method would be considerably less secure.

The final example illustrates how to poll an MIB object using the authentication and encryption services of the *authPriv* security model:

```
Freebsd% snmpget -v3 -u bpugsley -l authPriv -a MD5 -A hockeyrules -x DES -X shortguy
Router sysName.0
system.sysName.0 = Router.oreilly.com
Freebsd%
```

In this example, we added two new variables, privacy protocol type DES using *-x DES* and a privacy protocol pass phrase with *-X shortguy*. These variables enable SNMPv3 packet encryption and specify the pass phrase to use. This ensures that prying eyes cannot view the packet contents in transit. To illustrate the effectiveness of SNMPv3's encryption service, we provide a captured SNMPv3 packet. The packet was captured using the Ethereal protocol analyzer (for more information on Ethereal please see Appendix A):

```
Simple Network Management Protocol
    Version: 3
    Message Global Header
        Message Global Header Length: 16
        Message ID: 1608369049
        Message Max Size: 1480
        Flags: 0x03
            .... .0.. = Reportable: Not set
            .... ..1. = Encrypted: Set
            .... ...1 = Authenticated: Set
        Message Security Model: USM
    Message Security Parameters
        Message Security Parameters Length: 58
        Authoritative Engine ID: 800000090300000196708780
        Engine Boots: 2
        Engine Time: 1469970
        User Name: bpugsley
        Authentication Parameter: B53EFA21230735541B207A39
        Privacy Parameter: 00000002C483B016
    Encrypted PDU (74 bytes)
```

The packet response from the router contains some useful SNMP information, such as current version, encryption enabled, authentication enabled, and username (*bpugsley*), but Ethereal is unable to decipher the payload (*Encrypted PDU*). This is significant because the other versions of SNMP—including the other security models within SNMPv3—transport payload information in clear-text. At last, SNMP has evolved into a secure protocol.

Of course, SNMPv3 also provides full support for traps and informs including authentication, message integrity, and encryption. SNMPv3 traps and informs support the same three models of security as inbound services do. However, the *noAuthNoPriv* model provides no tangible advantage over SNMPv1 or SNMPv2c, and the *authPriv* model tends to be overkill since few networks will require encrypted traps.

To enable SNMPv3 trap support using authentication and message integrity, use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server host 172.25.1.1 version 3 auth ijbrown snmp envmon
Router(config)#end
Router#
```

The process of enabling SNMPv3 traps or informs is similar to the process for SNMPv2c, but with a few minor twists. First, you must define a SNMPv3 group and user, as in the previous examples. Second, you must include the keyword *auth*, which enables authentication. Third, you must include a valid SNMPv3 user (*ijbrown*, in this case). The router is then capable of forwarding SNMPv3 traps with full SNMPv3 authentication and message integrity enabled. For more information on enabling SNMP traps in general, see Recipe 17.13.

See Also

Recipe 17.1; Recipe 17.13; Recipe 17.7

17.22 Using SAA

Problem

You want to configure the routers to automatically poll one another to collect performance statistics.

Solution

Cisco supplies a feature called the Service Assurance Agent (SAA) in IOS Version 12.0(5)T and higher, which allows the routers to automatically poll one another to collect end-to-end performance statistics:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#rtr responder
Router1(config)#rtr 10
Router1(config-rtr)#type echo protocol ipIcmpEcho 10.1.2.3
Router1(config-rtr)#tag ECHO_TEST
Router1(config-rtr)#threshold 1000
Router1(config-rtr)#frequency 300
Router1(config-rtr)#exit
Router1(config)#rtr schedule 10 life 2147483647 start-time now
Router1(config)#rtr 20
Router1(config-rtr)#type jitter dest-ipaddr 10.1.2.3 dest-port 99 num-packets 100
Router1(config-rtr)#tag JITTER_TEST
Router1(config-rtr)#frequency 300
Router1(config-rtr)#exit
Router1(config)#rtr schedule 20 life 100000 start-time now ageout 3600
Router1(config)#exit
Router1#
```

The target router (`10.1.2.3`), which is specified as the destination in both of these tests, must be configured to respond to SAA tests:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#rtr responder
Router2(config)#exit
Router2#
```

Discussion

The SAA feature includes replace the earlier Round Trip Reporter (RTR) and Route Trip Time Monitor (RTTMON) facilities, which were available in IOS Version 11.3, and use the same basic syntax. However, where RTR includes only some simple round-trip ping and SNA tests, SAA includes several more interesting and useful features.

The first line in the example is the *rtr responder* command. This is required on all routers that will be taking part in SAA, including the targets of these tests. Both of these examples use a target IP address of `10.1.2.3`. This destination must be another Cisco router that is also configured with the *rtr responder* command.

In this example, we have configured two tests. The first test is given the arbitrary number 10 and the name *ECHO_TEST*. The second test is number 20, and is called *JITTER_TEST*. Note that you don't actually need to give your SAA tests names, but it is a good idea if you have several of them. This name (or tag) is included in the

SAA SNMP MIB table for this test. So, if you intend to download the test data via SNMP for performance management purposes, it can be extremely useful to name your tests so you know what they are.

Let's look at both of these example tests in more detail.

The first test does an ICMP echo (*ping*) to the destination device, `10.1.2.3`:

```
Router1(config)#rtr 10
Router1(config-rtr)#type echo protocol ipIcmpEcho 10.1.2.3
Router1(config-rtr)#tag ECHO_TEST
Router1(config-rtr)#threshold 1000
Router1(config-rtr)#frequency 300
Router1(config-rtr)#exit
```

The *threshold* command defines a minimum interesting threshold, which is set to 1,000 milliseconds here. This allows you to count the number of *ping* tests in which the round trip time was greater than one second, in addition to keeping track of the *ping* times and number of *ping* failures, which we will show in a moment.

Next is the *frequency* command, which defines how often this test will be run in seconds. In this case, we want the test to run every 5 minutes (300 seconds).

Then, once you have defined the test in the *rtr* configuration block, you have to tell the router when to run it. This is done with the *rtr schedule* command:

```
Router1(config)#rtr schedule 10 life 2147483647 start-time now
```

This command defines the schedule for running test number 10. It sets a lifetime for this test of 2,147,483,647 seconds (a very long time), which is the maximum value. This effectively means that this test will continue to run indefinitely. It is scheduled to start immediately.

When we scheduled the second test we used slightly different parameters:

```
Router1(config)#rtr schedule 20 life 100000 start-time now ageout 3600
```

In this case, the test is scheduled to run only for 100,000 seconds, which is about 27 hours. We have also configured an *ageout* value of 3600 seconds for this test. This says that the router will keep this test rule in memory for this length of time after it expires. The *ageout* option allows you to restart the test without having to reconfigure it.

You can view the data for the first test as follows:

```
Router1#show rtr operational-state 10
        Current Operational State
Entry Number: 10
Modification Time: 18:51:53.000 EST Tue Dec 17 2002
Diagnostics Text:
Last Time this Entry was Reset: Never
Number of Octets in use by this Entry: 1910
Connection Loss Occurred: FALSE
Timeout Occurred: FALSE
```

```
Over Thresholds Occurred: FALSE
Number of Operations Attempted: 203
Current Seconds Left in Life: 2147483647
Operational State of Entry: active
Latest Completion Time (milliseconds): 54
Latest Operation Start Time: 11:41:53.000 EST Wed Dec 18 2002
Latest Operation Return Code: ok
Latest 10.1.2.3
```

In this output you can see that it has run this test 203 times, and that the last test took 54 milliseconds and completed successfully. Note that it doesn't give a running average *ping* time. However, one of the nicest features of SAA is that you can configure a network management station to download this data using SNMP, provided you have the SAA MIB loaded on your server.

The second test is considerably more interesting. This test measures jitter between the routers by sending a series of UDP packets and looking at the time differences between them at both ends:

```
Router1(config)#rtr 20
Router1(config-rtr)#type jitter dest-ipaddr 10.1.2.3 dest-port 99 num-packets 100
Router1(config-rtr)#tag JITTER_TEST
Router1(config-rtr)#frequency 300
Router1(config-rtr)#exit
Router1(config)#rtr schedule 20 life 100000 start-time now ageout 3600
```

The *type* command defines a jitter test to the same destination IP address as the previous test. In this case, we have decided to use UDP port 99 for our test, and each test run will consist of 100 packets. The *frequency* command tells the router to run this test every five minutes. Here is some sample output from this test:

```
Router1#show rtr operational-state 20
        Current Operational State
Entry Number: 20
Modification Time: 10:25:36.000 EST Wed Dec 18 2002
Diagnostics Text:
Last Time this Entry was Reset: Never
Number of Octets in use by this Entry: 1742
Number of Operations Attempted: 22
Current Seconds Left in Life: 93400
Operational State of Entry: active
Latest Operation Start Time: 12:10:36.000 EST Wed Dec 18 2002
RTT Values:
NumOfRTT: 98    RTTSum: 6063    RTTSum2: 384317
Packet Loss Values:
PacketLossSD: 0 PacketLossDS: 2
PacketOutOfSequence: 2  PacketMIA: 0    PacketLateArrival: 0
InternalError: 0        Busies: 0
Jitter Values:
MinOfPositivesSD: 4     MaxOfPositivesSD: 14
NumOfPositivesSD: 32    SumOfPositivesSD: 175   Sum2PositivesSD: 1111
MinOfNegativesSD: 1     MaxOfNegativesSD: 5
NumOfNegativesSD: 60    SumOfNegativesSD: 175   Sum2NegativesSD: 547
```

```
MinOfPositivesDS: 1     MaxOfPositivesDS: 45
NumOfPositivesDS: 20    SumOfPositivesDS: 78    Sum2PositivesDS: 2166
MinOfNegativesDS: 1     MaxOfNegativesDS: 16
NumOfNegativesDS: 21    SumOfNegativesDS: 69    Sum2NegativesDS: 693
```

There is a clearly a lot more information in this test output. This is because measuring jitter is not a simple single-variable test. A jitter measurement characterizes the statistical distribution of packet-by-packet variation in forward and backward latency, as well as for the round trip. Note that, as with the SAA *ping* test we discussed earlier, the router records only the results of the most recent test. If you want to keep historical records, you need to poll and download the SAA MIB tables once per poll cycle.

The first set of numbers are the Round Trip Time (RTT) values. You can see that this sample included 98 packets. The total of all of the round trip times of all of these packets was 6063 milliseconds, and the sum of the squares of all of these times was 384,317 milliseconds. These values are not extremely meaningful in themselves, but if you divide the RTTSum value by the number of measurements, you get the average latency for this set of packets: roughly 61 milliseconds.

Applying some simple statistics, you can use the square value to understand how the actual values are spread around this average. The mean of the squares of the round trip times is 3,922 milliseconds^2 (just dividing the sum of the squares by the total number of samples). If you subtract the square of the average from this value and take the square root, you get a statistical estimate of the variation in milliseconds. The higher this value, the greater the spread. In this case, you can calculate that this spread is roughly 14 milliseconds. This means that half of the time, the round trip latency is within the range 61±14ms. Note that the ± symbol is a standard mathematical notation that, in this case, indicates a range from 47ms (61-14) to 75ms (61+14).

The next set of data records dropped packets. Recall that the sample size is 100 packets, but the NumOfRTT value is only 98. So the network must have dropped two of our test packets. SAA separately keeps track of packets lost in both directions, source to destination (PacketLossSD) and destination to source (PacketLossDS). This router is the source; the other router is the destination. So, in this example, both of the lost packets happened on the way back. Also note that the output claims that there were two out-of-sequence packets during this test, which is consistent with the number of dropped packets. The router saw the sequence number jump by 2 because of the dropped packet, and recorded it as an out-of-sequence error.

The next group of numbers includes the actual jitter measurements. There are two groups of numbers here. The variables that end with "SD" are measured from the source to the destination, and those labeled "DS" are for the return path. Within each of these groups there are two subgroups, one for "Positives," and the other for "Negatives." "Positives" are events where the spacing between two packets has increased since the last pair of successive packets. The "Negatives" counters record

all of the times that the interpacket spacing decreased. Let's look a little bit more closely at one set of values:

```
MinOfPositivesSD: 4     MaxOfPositivesSD: 14
NumOfPositivesSD: 32    SumOfPositivesSD: 175   Sum2PositivesSD: 1111
```

Of the 100 packets the router sent in this polling interval, there were 32 cases in which the jitter in the forward direction had a positive value. Of these, the largest value was 14 milliseconds and the smallest was 4 milliseconds. We can use the sum and the sum of the squares to calculate the average and spread of values in precisely the same way as we did to calculate the average latency. The result we get is that half of the time, positive jitter in this direction was within the range 5.5±2.2ms.

Applying this same technique to the other jitter measurements gives other useful statistics. The negative jitter from source to destination was 2.9±0.8ms with a maximum of 5ms and a minimum value of 1ms. In the other direction, the positive jitter was 3.9±9.6ms and the negative jitter was 3.3±4.7ms. These last two values might look a little bit funny because the spread is larger than the mean. However, this is not actually bad because the output also shows that the maximum positive jitter in this direction was 45ms, and the maximum negative jitter was 16ms. The spread is very large, but the mean jitter values are relatively small. This is a fairly typical result.