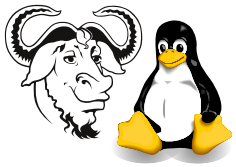


SNMP Agent, the Set Operation, Traps and Notifications

Contents

1 Background

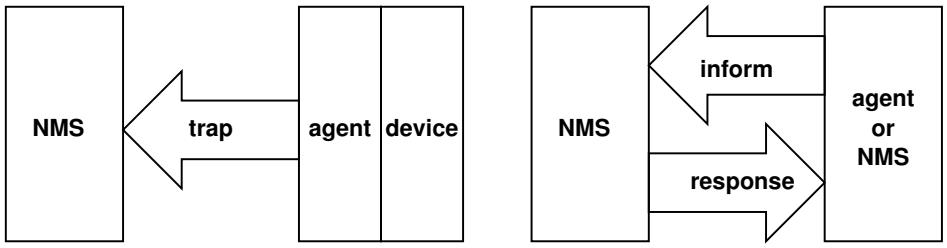
Before you do this exercise, you will need a Linux installation on your hard disk. To see how to do this, refer to the document `http://nicku.org/snm/lab/install-linux/install-linux.pdf`. You will then need to configure `sudo` as described in `http://nicku.org/ossi/lab/sudo/sudo.pdf`.

You then need to configure your SNMP agent to allow `set-request` and `get-request` operations as described below. For this exercise, you will set a simple *community string* for read and another for read-write operations. The agent is called `snmpd`. The configuration for the agent is in the file `/etc/snmp/snmpd.conf`.

1.1 Traps and Inform Requests

All versions of SNMP support a mechanism called a *trap* or *inform request*. A *trap* or *inform request* is like an interrupt from the agent to the manager. There is no response to a trap, as seen in figure 1(a) on the following page, whereas inform requests have responses, as shown in figure 1(b). A trap usually indicates something is wrong, allowing the manager software to take actions such as flag the problem with a red symbol, and perhaps send email and perhaps a short message to the management team.

The Net-SNMP tools provide the `snmptrap` command to send traps and notifications, and the `snmptrapd` program to receive them.



(a) Traps (b) Notification

Figure 1: Traps get no response, while an SNMPv2 inform requires a response. The agent may resend the inform-request after a timeout period during which it gets no response.

Generic trap name and number	Definition
`coldStart(0)`	Shows the agent has been restarted.
`warmStart(1)`	Indicates that the agent has reinitialised itself. None of the management variables have been reset.
`linkDown(2)`	Sent when an interface on the device has gone down. The first variable binding indicates which interface went down.
`linkUp(3)`	Sent when an interface on the device comes back up. The first variable binding indicates which interface came up.
`authenticationFailure(4)`	Indicates someone used the wrong community string to access your agent. Useful to detect if someone is attacking your device using SNMP.
`egpNeighborLoss(5)`	Indicates that an *Exterior Gateway Protocol* (EGP) neighbour has gone down.
`enterpriseSpecific(6)`	Indicates that the trap is enterprise specific. Devices from vendors often come with their own SNMP traps under the vendor's branch under the private-enterprise node of the MIB tree. To process this, the NMS needs to decode the trap number from the SNMP message. You can implement your own traps. See the example below, where a trap MIB module defines a trap.

Table 1: Generic traps in SNMP version 1.

1.2 SNMPv1 traps

Traps in SNMP version 1 come with seven "generic traps" as shown in table 1.

You can define your own MIB for your own specific traps. See the exercises in section 2.4.1 on page 7.

1.3 Using `snmptrap` to send traps

The syntax of the program `snmptrap` for sending SNMP v1 traps is:

```
$ snmptrap -v 1 -c ⟨community-string⟩ ⟨hostname⟩ ⟨enterprise-oid⟩ ⟨agent⟩ \
```

⟨*generic-trap*⟩ ⟨*specific-trap*⟩ ⟨*uptime*⟩ [⟨*OID*⟩ ⟨*type*⟩ ⟨*value*⟩]...

Argument	Meaning
⟨*hostname*⟩	the hostname of the manager to which you are sending the trap.
⟨*enterprise-oid*⟩	the OID under which a number of different enterprise traps may be defined. In the example below, this is `TRAP-TEST-MIB::demotraps`.
⟨*agent*⟩	the hostname or IP address of the agent. If you leave this as `""`, then it will be set to the address of the current machine. This may seem pointless, but it is important when you use a proxy.
⟨*generic-trap*⟩	the number from table 1 on the previous page, which is in the range 0...6.
⟨*specific-trap*⟩	the number of the trap defined under the MIB object ⟨*enterprise-oid*⟩. In the example below, this is 17, corresponding to `TRAP-TEST-MIB::demo-trap`.
⟨*uptime*⟩	normally an empty string; that means use the local value of `sysUpTime` on this agent.
⟨*OID*⟩	there may be one or more variable values sent with the trap to indicate what is happening. You provide them in a group of ⟨*OID*⟩ ⟨*type*⟩ ⟨*value*⟩
⟨*type*⟩	The data type of the value sent. Usually `i` for integer, or `s` for string. `man snmptrap` for all the types.
⟨*value*⟩	the value of the variable that is sent in the trap.

Table 2: The required arguments for an SNMPv1 trap with the format: `snmptrap -v 1 -c` ⟨*community-string*⟩ ⟨*hostname*⟩ ⟨*enterprise-oid*⟩ ⟨*agent*⟩ ⟨*generic-trap*⟩ ⟨*specific-trap*⟩ ⟨*uptime*⟩ [⟨*OID*⟩ ⟨*type*⟩ ⟨*value*⟩]...

1.4 SNMPv2c Notifications

In SNMPv2c and v3, the agent sends *notifications*. Notification is a macro which sends either a trap or an inform request. An inform request is like a trap with an acknowledgment. A trap is simpler; it has no generic numbers, specific numbers or enterprise OIDs. As with SNMPv1 traps, you will use `''` to indicate the current system uptime.

The syntax for generating an SNMPv2 notification is

`$ snmptrap -v 2c -c` ⟨*community-string*⟩ ⟨*hostname*⟩ ⟨*uptime*⟩ ⟨*trapOID*⟩ `\`
[⟨*OID*⟩ ⟨*type*⟩ ⟨*value*⟩]...

The meaning of each argument is the same as for SNMPv1 traps, except for the ⟨*trapOID*⟩, which is the OID of a complete MIB object which must be defined for the notification.

2 Procedure

2.1 Installing and Configuring the Agent

I took the `net-snmp` source RPM software package from Fedora Core 1 and compiled it on Red Hat 9. I did this because it provides better support for Perl SNMP programming. We

will install the resulting binary software packages from an NFS directory on our server.

1. Change to the network directory:

```
$ cd /home/nfs/snmp
```

2. Now install the software packages:

```
$ sudo rpm -Uhv net-snmp-*5.0.9-2.i386.rpm
```

   Note the asterisk '`*`'. I suggest press the Tab key to complete the file name, then add the asterisk afterwards, to reduce typing mistakes.

3. Edit the configuration for the SNMP agent:

```
$ xhost +localhost
$ sudo -v
$ sudo emacs /etc/snmp/snmpd.conf &
```

   The first line allows users other than your own account (such as the user root) to display graphical objects on your local X server.

   The second line starts a new five-minute password free period for `sudo`. If the five-minute period has expired, then the editor cannot start in the background; `sudo` will wait for you to bring it to the foreground by typing `fg`, so that you can enter your password. Typing `sudo -v` simply avoids this inconvenience.

4. Configure two community strings in the `snmpd.conf` file, as in the following patch. Note that we are using VACM commands here; we could just use `rocommunity` and `rwcommunity` instead (see `man snmpd.conf`), but we are learning about VACM.

```
--- snmpd.conf~ 2003-11-28 16:25:40.000000000 +0800
+++ snmpd.conf  2003-12-01 11:47:49.000000000 +0800
@@ -39,13 +39,16 @@

 #       sec.name  source          community
 com2sec notConfigUser  default       public
+com2sec RWUser         default       private

 ####
 # Second, map the security name into a group name:

 #       groupName      securityModel securityName
-group   notConfigGroup v1           notConfigUser
+group   notConfigGroup v1            notConfigUser
 group   notConfigGroup v2c           notConfigUser
+group   RWGroup        v1            RWUser
+group   RWGroup        v2c           RWUser

 ####
 # Third, create a view for us to let the group have rights to:
@@ -54,12 +57,15 @@
 #       name           incl/excl     subtree         mask(optional)
 view    systemview    included   .1.3.6.1.2.1.1
 view    systemview    included   .1.3.6.1.2.1.25.1.1
+view    all           included   .1

 ####
```

```
 # Finally, grant the group read-only access to the systemview view.

 #       group          context sec.model sec.level prefix read   write  notif
-access  notConfigGroup ""      any       noauth    exact  systemview none none
+#access  notConfigGroup ""      any       noauth    exact  systemview none none
+access  notConfigGroup ""      any       noauth    exact  all none none
+access  RWGroup        ""      any       noauth    exact  all all all

 # ----------------------------------------------------------------------
```

You can see all the options that the agent can understand in its configuration file `snmpd.conf` by typing `snmpd -H`, and of course, from the man page: `man snmpd.conf`.

Another good source of information about the Net-SNMP tools is the Net-SNMP FAQ: `http://net-snmp.sourceforge.net/FAQ.html`, and also the Net-SNMP tutorials: `http://net-snmp.sourceforge.net/tutorial/`.

Note that "in real life," these passwords are sensitive, and should be selected as you would any other password.

5. Start the agent:

```
$ sudo /sbin/service snmpd start
```

6. Enable the agent to always start when the computer boots:

```
$ sudo /sbin/chkconfig snmpd on
$ /sbin/chkconfig snmpd --list
```

The first command ensures that the next time a computer boots on your hard disk, it will start the agent `snmpd` whenever it moves into runlevels 3, 4 or 5.

The second command lists which runlevels `snmpd` will start at, to confirm to yourself that the previous command worked.

Note: you may wish to add the directories `/sbin` and `/usr/sbin` to your `PATH` in your login script, `~/.bash_profile`. The line you would want to add is:

```
export PATH=$PATH:/sbin:/usr/sbin
```

7. Note that any time you change the configuration for the agent (by editing `/etc/snmp/snmpd.conf`), you will need to restart the agent to get it to read the new configuration with:

```
$ sudo /sbin/service snmpd restart
```

2.2 The MIB Files

1. Identify the list of MIB files installed on your computer:

```
$ rpm -ql net-snmp | grep mibs
```

The RPM package manager manages software packages on your computer. It maintains a database of all the software packages installed, all the files within each package, and checksums on each file, as well as plenty of other information that you can query. For more about this, read the chapter on RPM in the Red

Hat Reference Guide (download from the Red Hat web site—it's a really well-written manual, worth reading).

The command `rpm -ql net-snmp` is a **q**uery to **l**ist the files in the `net-snmp` package. We are just `grep`ping here for the MIB files.

2. Examine the file `SNMPv2-MIB.txt`.

This is one of the MIB files on your computer. They are text files, written using Abstract Syntax Notation One (ASN.1). The MIB files are like the schema for a database: they determine what type of data each MIB variable can carry, and provide a name and OID for each MIB variable (managed object). They also show the exact location of each MIB variable in the MIB tree.

3. Identify some MIB objects that have a `MAX-ACCESS` of `read-write`. These are variables that you can change with a `set-request` SNMP operation.

Note that the manual page for the agent configuration file, `snmpd.conf`, shows that if `/etc/snmp/snmpd.conf` has a value for `syslocation`, `syscontact` or `sysname`, then the MIB variables `sysLocation`, `sysContact` and `sysName` respectively will become read only, so only try to set these MIB variables if you comment out the corresponding value in `/etc/snmp/snmpd.conf`.

2.3 Setting Managed Objects

1. Identify some managed objects that you can set.

2. Read their original values with `snmpgetnext`.

3. Change their value with `snmpset`.

You can refer to the man page for `snmpset`. There is also a simple tutorial on `snmpset` at `http://net-snmp.sourceforge.net/tutorial-5/commands/snmpset.html`.

4. You can use `snmpset` like this:

`$ snmpset -v 2c -c` ⟨*rw community string*⟩ ⟨*hostname*⟩ `\`
⟨*OID*⟩ ⟨*type*⟩ ⟨*value*⟩ `[`⟨*OID*⟩ ⟨*type*⟩ ⟨*value*⟩`]...`

Just in case you are wondering what the value of ⟨*rw community string*⟩ might be, you set it in your agent back in step 4 on page 4. The value of ⟨*type*⟩ is shown on the `snmpset` manual page. It is a single letter, such as `s` for string, or `i` for integer.

5. Verify that the value has or has not changed. Note that the `SYNTAX` determines that type of data that you may put into that managed object. The syntax values are what we have discussed in the lectures.

2.4 Sending Traps

Now read the Net-SNMP tutorial on traps at `http://net-snmp.sourceforge.net/tutorial-5/commands/snmptrap.html`. Using this as a guide, working with a partner if possible, send your partner a trap using `snmptrap`, and also receive a trap from your partner using `snmptrapd`. If you have no partner available, then you can send and receive the trap on the same machine.

2.4.1 Exercises with Traps and Notifications

We begin by installing two MIB files, one for an enterprise OID for an SNMPv1 trap, and another for a notification MIB object for an SNMPv2c notification. We install the MIB files as described in the Net-SNMP FAQ at `http://net-snmp.sourceforge.net/FAQ.html#How_do_I_add_a_MIB_`.

1. Download the two MIB files from the subject web site, at `http://nicku.org/snm/lab/snmp-set-trap/TRAP-TEST-MIB.txt` and `http://nicku.org/snm/lab/snmp-set-trap/NOTIFICATION-TEST-MIB.txt`. These are from the Net-SNMP Trap tutorial.

   Here is the file `TRAP-TEST-MIB.txt`

   ```
   TRAP-TEST-MIB DEFINITIONS ::= BEGIN
           IMPORTS ucdExperimental FROM Net-SNMP-MIB;

   demotraps OBJECT IDENTIFIER ::= { ucdExperimental 990 }

   demo-trap TRAP-TYPE
           STATUS current
           ENTERPRISE demotraps
           VARIABLES { sysLocation }
           DESCRIPTION "This is just a demo"
           ::= 17
   END
   ```

   And here is `NOTIFICATION-TEST-MIB.txt`

   ```
   NOTIFICATION-TEST-MIB DEFINITIONS ::= BEGIN
           IMPORTS ucdavis FROM Net-SNMP-MIB;

   demonotifs OBJECT IDENTIFIER ::= { ucdavis 991 }

   demo-notif NOTIFICATION-TYPE
           STATUS current
           OBJECTS { sysLocation }
           DESCRIPTION "Just a test notification"
           ::= { demonotifs 17 }

   END
   ```

2. Create a directory in your home directory to put these MIB files':

   ```
   $ mkdir -p ~/.snmp/mibs
   ```

   Notice the dot in the directory name.

3. Copy the two MIB files into this directory:

   ```
   $ cp NOTIFICATION-TEST-MIB.txt TRAP-TEST-MIB ~/.snmp/mibs
   ```

4. Now start the `snmptrapd` service:

   ```
   $ sudo /sbin/service snmptrapd start
   ```

   and optionally set the service to always start on boot:

   ```
   $ sudo chkconfig snmptrapd on
   ```

   . . . then verify that it is on:

   ```
   $ sudo chkconfig snmptrapd --list
   ```

5. Open another terminal window to watch the main log:

   ```
   $ sudo tail -f /var/log/messages
   ```

6. Now send some traps while watching the log window. This first example is an SNMPv1 trap.

   ```
   $ snmptrap -c public localhost TRAP-TEST-MIB::demotraps \
       '' 6 17 '' SNMPv2-MIB::sysLocation.0 s "Just here"
   ```

   See section 1.3 on page 2 for a more detailed explanation of what is happening here.

7. This other example from the tutorial is an SNMPv2c notification.

   ```
   $ snmptrap -v 2c -c public localhost '' \
       NOTIFICATION-TEST-MIB::demo-notif \
       SNMPv2-MIB::sysLocation.0 s "just here"
   ```

   See section 1.4 on page 3 for an explanation of the parameters.

   For the SNMP version 1 example, we are using the enterprise specific trap (see table 1 on page 2). The MIB file `TRAP-TEST-MIB.txt` defines a trap (`demo-trap`) with numeric OID of `.1.3.6.1.4.1.2021.13.990.0.17`. Go ahead: try this after installing the two MIB files:

   ```
   $ snmptranslate -On TRAP-TEST-MIB::demo-trap
   .1.3.6.1.4.1.2021.13.990.0.17
   $ snmptranslate -On NOTIFICATION-TEST-MIB::demo-notif
   .1.3.6.1.4.1.2021.991.17
   ```