[Optional] Simple Network Management Protocol (SNMP)

COSC301 Laboratory Manual

Old Lab

This lab hasn't been run recently, so no guarantees are made regarding the commands and their output.

In this lab, we shall use our Server1 and Client1 machines to explore the use of the Simple Network Management Protocol (SNMP) for monitoring network devices such as routers and hosts. We shall explore concepts such as how SNMP works, its conceptual model, and how the data is given meaning using Management Information Bases (MIBs). We shall briefly consider some different tools for processing the data.

We shall also enable SNMP monitoring of a server using the Net-SNMP package. This allows us to monitor a host, rather than just network devices. Finally, we shall extend an extensible SNMP *agent* so we can publish information of our choosing using SNMP.

Please do the first section of the assessment before coming to the lab. This is to ensure you review the conceptual model and architecture that you learned during the corresponding lecture. Once you have done this, please proceed with the rest of the lab.

1. Reference only: Configuring SNMP Agent on Cisco and Vyatta

You don't need to do this for this lab, as you don't have a Vyatta router in your network with Server1 and Client1. Therefore, it is for reference only.

Here are the basic commands you would use to configure an SNMP agent on Cisco IOS and Vyatta. These are for reference only; compare them with the following section, but you are not expected to perform anything for this section in the lab. First, here are some basic commands for Cisco devices, just to get you started, which would be done in configuration mode. For production use, you would also want to consider restricting access using ACLs (where queries can come from) and views (what parts of the tree can be seen, depending on the submitted community string).

```
snmp-server community COMMUNITY ro|rw
snmp-server location LOCATION
snmp-server contact CONTACT
```

And here are the commands for Vyatta, also to be done in configuration mode.

```
set service snmp community COMMUNITY authorization ro|rw
set service snmp location LOCATION
set service snmp contact CONTACT
```

2. Install the Net-SNMP Agent

We install and configure the Net-SNMP agent software on our Linux-based gateway.

SNMP agents are commonly already available on router products and managed switches, but you can also get them for hosts as well a number of other embedded devices on the network, such as for monitoring the health of battery backup systems, air-conditioning plants etc. One commonly used SNMP agent is Net-SNMP, which is quite comprehensive and extensible. In this section, we install and configure **snmpd** for basic use, including some of the host-resources MIBs. We shall practice querying it in the subsequent sections.

Install the Net-SNMP agent (server) software on Server1, which is available in the package **snmpd** on Debian-based machines. Also, install the **snmp** package, which contains the manager software (clients) on both Server1 and Client1.

Because SNMP can be used to monitor all sorts of devices, we need to map from the OIDs (we'll see these later on) into human readable names. Install the `snmp-mibs-downloader` package, and comment out the `mibs` line in `/etc/snmp/snmp.conf`.

You should now find two other configuration files in `/etc/snmp/`: `snmpd.conf` and `snmptrapd.conf`. Ignore the last one, we won't be making use of any traps for this lab. The first file, `snmpd.conf` is the one we want to be looking at further as it controls the agent (server). On some systems you might also find `snmp.conf`, which lists defaults for the clients; it does not exist by default.

Net-SNMP actually comes with a configuration program called **snmpconf** to get you started, but it doesn't add much value unless you're doing it yourself; the configuration file is fairly simple. As root, open the `snmpd.conf` file in an editor and configure with the following (I have ommited comments to save space):

```
agentaddress udp:161,udp6:161,tcp:161,tcp6:161
rocommunity public
rocommunity6 public
proc  named          1 1
proc  squid
proc  sshd          5 1
disk  /      10%
load  5 3 2
sysLocation  "Rm1.23, FOO Building"
sysContact  "Your e-mail <admin@example.com>"
```

So what is happening in this configuration? First we have configured all the various ways I wish to be able to access this agent, over both IPv4 and IPv6 on all supported addresses, and both UDP (standard) and TCP (Net-SNMP extension). I like having TCP available because it means I can query it easily using SSH tunnelling remotely.

I have configured a read-only string called "public", for both IPv4 and IPv6. I could have configured additional community strings, optionally limiting what each community string can see, but I have opted not to. Note that no read-write access has been configured, and no SNMPv3 access has been configured.

If we just skip down to 'syslocation' and 'syscontact', then we complete the basic SNMP configuration. This leaves us with the 'proc', 'disk' and 'load' lines, which configures support in Net-SNMP for Process Monitoring, Disk Usage Monitoring and Load Monitoring.

'proc' entries are for monitoring various configured processes. Here we see there must be exactly one process of **named** running, at least one **squid** process should be running, and between one and five **sshd** processes should be running. If these limits are exceeded, then the 'prTable', which is where this information will be reported, will show an error state.

'disk' lines are for monitoring file-system utilisation. Here we only have one, but you could have one for every filesystem, particularly for filesystems that are often written to, such as `/var/` and `/tmp/`. We have specified a minimum percentage of the filesystem that must be available, if not an error flag will be seen in the 'diskTable'.

Finally, the 'load' entry provides a way of monitoring system load, which is one of the more useful things to monitor via SNMP for a host.

Change `/etc/defaults/snmpd` to be as the following. The only change should be to the SNMPDOPTS line, which reduces the amount of logging that happens, which is useful when graphing, and also removing the default behaviour of only being accessible from localhost[1]. I have removed comments to save space. Note that the documentation for **snmpd** disagreed with what it actually accepts, which was most annoying.

```
export MIBDIRS=/usr/share/snmp/mibs
SNMPDRUN=yes
SNMPDOPTS='-LS 4d -Lf /dev/null -u snmp -g snmp -I -smux -p /var/run/snmpd.pid'
  DELETE "127.0.0.1" at the end of SNMPDOPTS
TRAPDRUN=no
TRAPDOPTS='-Lsd -p /var/run/snmptrapd.pid'
SNMPDCOMPAT=yes
```

Now you can restart the software.

```
# /etc/init.d/snmpd restart
```

Ensure that it is running and check the logs. I found a couple of messages about getaddrinfo, but otherwise it seems to be okay.

```
$ ps -eo command | grep snmpd
/usr/sbin/snmpd -LS 4d -Lf /dev/null -u snmp -g snmp -I -smux -p /var/run/snmpd.pid
# tail /var/log/syslog
```

Finally, ensure that **snmpd** is listening on the expected interfaces.

```
# lsof -Pni | grep snmpd
snmpd  1821  snmp  7u   IPv4  3641153  UDP  *:161
snmpd  1821  snmp  9u   IPv6  3641158  UDP  *:161
snmpd  1821  snmp  10u  IPv4  3641159  TCP  *:161  (LISTEN)
snmpd  1821  snmp  11u  IPv6  3641161  TCP  *:161  (LISTEN)
```

Great, we should be ready to practice querying it. Before we do, let's just run a very simple query to ensure that it's working.

```
$ snmpget -v2c -c public localhost sysContact.0
Timeout: No Response from localhost.
```

It failed. Looking at the logs, we see that it has reported a "Connection Refused" event, which indicates that TCP Wrappers is likely being used. Edit `/etc/hosts.allow` and create a suitable entry (based on the other entries in that file) for the `snmpd` service. Restart the SNMP service and try the command again:

```
$ snmpget -v2c -c public localhost sysContact.0
SNMPv2-MIB::sysContact.0 = STRING: "Your e-mail <admin@example.com>"
```

[1] Configurations on the command-line (which this file is for) override configurations in `snmpd.conf`.

If you got that, then you are ready to proceed. Otherwise, check your logs. Timeouts can indicate either that nothing is running on that interface or port, the port is blocked (firewall or other access-control-mechanism on the agent), or the community string is wrong.

3. Command-Line Clients from Net-SNMP

In this section, we shall exercise the use of various Net-SNMP command-line tools and briefly investigate how we can configure some defaults to make them easier to use. This section should build your confidence in the use of these tools and make you aware of the sort of programs which are available and when you might want to use them.

All of the Net-SNMP command-line managers share many common arguments, which are documented in snmpcmd(1). Two of the most common options are to specify the SNMP version and community-string, which we briefly saw at the end of the previous section.

`-v 2c`
: Specifies SNMP version 2c ("SNMPv2")

`-c public`
: Specifies the community string to use.

These are very common, so we can put them in `/etc/snmp/snmp.conf` or `~/.snmp/snmp.conf`. Here is what I have configured in mine, make your version the same.

```
defVersion 2c
defCommunity "public"
```

Now, let's try the same command we used previously, but this time we don't need to specify quite as much on the command-line.

```
$ snmpget localhost sysContact.0
SNMPv2-MIB::sysContact.0 = STRING: "Your e-mail <admin@example.com>"
```

3.1. snmptranslate

There are numerous ways of specifying the same OID, some more convenient as others. We often wish to input or output these in a particular format, and the **snmptranslate** command allows us to do just that. We shall just cover the basics with some common examples.

All of the options pertaining to the (user) input of OIDs begin with an 'I', but the only really useful one is `-IR`: this enables "random access" to the SMI tree, so you don't have to specify it relative to any fixed point. This is very useful for casual use as it saves a lot of typing. You'll probably want to use it very often. It's not on by default because there is potential to be ambiguous.

```
$ snmptranslate ifInOctets
ifInOctets: Unknown Object Identifier (Sub-id not found: (top) -> ifInOctets)
$ snmptranslate .iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifInOctets
IF-MIB::ifInOctets
$ snmptranslate -IR ifInOctets
IF-MIB::ifInOctets
```

Here are some alternative ways of specifying the same OID in an unambiguous manner:

IF-MIB::ifInOctets
Relative to a known MIB module name. This is the most convenient way of specifying on OID while generally not being ambiguous.

.1.3.6.1.2.1.2.2.1.10
Fully qualified numeric.

.iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifInOctets
Fully qualified symbolic (or mixed).

Of slightly less importance is output format, of which the arguments begin with 'O' (Capital 'o' for 'output', not zero). Compare the output from these commands, beginning with the default output:

```
$ snmptranslate -IR ifInOctets
IF-MIB::ifInOctets
$ snmptranslate -IR -Of ifInOctets    output fully qualified format
.iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifInOctets
$ snmptranslate -IR -Os ifInOctets    output short, unqualified format
ifInOctets
$ snmptranslate -IR -OS ifInOctets    output short format with containing MIB
IF-MIB::ifInOctets
$ snmptranslate -IR -On ifInOctets    output numeric, fully-qualified format
.1.3.6.1.2.1.2.2.1.10
```

snmptranslate can be a useful front-end for scripts and tools that require, for example, a numeric OID, but you want to enable the user to provide the OID in a more friendly manner.

There are many more options available, see the snmpcmd(1) for further details.

Finally, there are some arguments that are specific to **snmptranslate**. One argument is useful for printing part of the MIB tree; another is useful for viewing documentation about a particular OID; there are others, but these are the only two demonstrated below.

```
$ snmptranslate -Tp -IR prTable
+--prTable(2)
   |
   +--prEntry(1)
      |  Index: prIndex
      |
      +-- -R-- Integer32 prIndex(1)
      |        Range: 0..65535
      +-- -R-- String    prNames(2)
      |        Textual Convention: DisplayString
      |        Size: 0..255
      +-- -R-- Integer32 prMin(3)
      +-- -R-- Integer32 prMax(4)
      +-- -R-- Integer32 prCount(5)
      +-- -R-- EnumVal   prErrorFlag(100)
      |        Textual Convention: UCDErrorFlag
      |        Values: noError(0), error(1)
      +-- -R-- String    prErrMessage(101)
      |        Textual Convention: DisplayString
      |        Size: 0..255
      +-- -RW- EnumVal   prErrFix(102)
      |        Textual Convention: UCDErrorFix
      |        Values: noError(0), runFix(1)
      +-- -R-- String    prErrFixCmd(103)
               Textual Convention: DisplayString
               Size: 0..255
$ snmptranslate -Td -IR prTable
UCD-SNMP-MIB::prTable
prTable OBJECT-TYPE
```

```
  -- FROM       UCD-SNMP-MIB
  MAX-ACCESS    not-accessible
  STATUS        current
  DESCRIPTION   "A table containing information on running
         programs/daemons configured for monitoring in the
         snmpd.conf file of the agent.  Processes violating the
         number of running processes required by the agent's
         configuration file are flagged with numerical and
         textual errors."
::= { iso(1) org(3) dod(6) internet(1) private(4)
      enterprises(1) ucdavis(2021) 2 }
```

3.2. snmpget and snmpgetnext

One of the essential skills is knowing how to get data using SNMP. For this, you need to know how to specify the agent. Net-SNMP allows you to connect to the agent using a variety of different mechanisms, such as UDP/IPv4, UDP/IPv6, TCP/IPv4, TCP/IPv6, ATM, IPX or Unix sockets; don't worry if some of these are unfamiliar to you, the important one is UDP/IPv4 and UDP/IPv6, although the TCP ones can also be useful.

The full details can be found in snmpcmd(1), but here are some examples from that manual page to help you quickly get the idea of what you can say. Remember that SNMP is typically UDP port 161, so when using IP (any version), all you need is a hostname, and it will talk to that host on UDP port 161.

`hostname`:161
: Perform query using UDP/IPv4 datagrams to `hostname` on port 161. The `:161` is redundant here since that is the default SNMP port.

udp:`hostname`
: Identical to the previous specification. The `udp:` is redundant here since UDP/IPv4 is the default transport.

TCP:`hostname`:1161
: Connect to `hostname` on port 1161 using TCP/IPv4 and perform the query over that connection.

udp6:`hostname`:10161
: Perform the query using UDP/IPv6 datagrams to port 10161 on `hostname` (which will be looked up as an AAAA record).

UDP6:[fe80::2d0:b7ff:fe21:c6c0]
: Perform the query using UDP/IPv6 datagrams to port 161 at address fe80::2d0:b7ff:fe21:c6c0.

tcpipv6:[::1]:1611
: Connect to port 1611 on the local host (::1 in IPv6 parlance) using TCP/IPv6 and perform the query over that connection.

  Note that `tcpipv6` is the same as `tcp6`.

Screenshot

Practice querying the agent over each of UDP/IPv4, UDP/IPv6, TCP/IPv4 and TCP/IPv6. Take a screenshot to show you have done each step. The command will look something like the following:

```
$ snmpgetnext agent sysContact
```

1. Install the Net-SNMP manager software (the ‘snmp’ package) onto Client1.

2. Practice querying the agent remotely using at least UDP/IPv6 and UDP/IPv6.

3. Query the agent remotely over an SSH ‘local’ tunnel using TCP port forwarding.

Later during the assessment, you need to be able to explain the difference between **snmpget** and **snmpgetnext**, and why an OID specified as ‘sysContact’ won’t work for snmpget.

3.3. snmpwalk and snmpbulkwalk

Walking a tree in SNMP version 1 is done using a series of GetNext and GetResponse packets, which can take a long to complete for a large tree, even on a fast network. The problem is that we have too many network round-trips to complete. It would be better to ask the agent (server) for the tree under a particular OID, and have it return a bulk set of answers, vastly cutting down the number of round-trips to the server. Version 2c introduced such bulk operations, which has a major impact on even very low-delay communications.

We can measure the time taken to complete a command using the **time** command. Because the cost of writing output to the terminal is very expensive, I have redirected output to `/dev/null` so the cost of writing to the terminal is minimised.

```
$ time snmpbulkwalk localhost . >/dev/null
real    0m1.134s
…
$ time snmpwalk localhost . >/dev/null
real    0m2.428s
…
```

This was testing just on localhost, which is typically a *very* low-delay way of talking to yourself. Imagine how much more time you would save if there was a real network between yourselves, with much greater delay.

snmpwalk and **snmpbulkwalk** are basically equivalent in the arguments you give them:

```
$ snmpwalk localhost ifDesc
IF-MIB::ifDescr.1 = STRING: lo
IF-MIB::ifDescr.2 = STRING: outside
IF-MIB::ifDescr.3 = STRING: inside
$ snmpbulkwalk localhost ifDesc
… you get the same result
```

3.4. snmptable

Many of the interesting items available via SNMP are logically represented as a table. Some examples are the interface table ‘ifTable’; the IP address table ‘ipAddrTable’, routing table ‘ipRouteTable’, and ARP[2] table ‘ipNetToMediaTable’; the TCP connection table ‘tcpConnTable’ and UDP table ‘udpTable’.

The UCD-SNMP MIB (which is where the Net-SNMP-specific additions are found) has other tables which we have previously configured, have a look at the ‘dskTable’, ‘laTable’, and ‘prTable’.

Other devices may support further tables, such as NAT translations for routers, CAM tables on Ethernet switches (to see which MAC addresses lie on which switch ports).

[2]Well, network-to-datalink mapping.

A number of these tables can get very wide and wrap awkwardly in a terminal, so I suggest you pipe the output to **less -S**, which allows you to scroll side-to-side.

```
$ snmptable localhost ifTable | less -S
… remember to press ‘q’ to quit …
```

> **Screenshot**
> Take a screenshot showing one of the tables.

3.5. Other commands…

There are numerous other commands available. We can group them into several groups: those that deal with setting values via SNMP; those that deal with SNMPv3; those that deal with sending asynchronous traps and informs; those that are SNMP versions of well-known commands; and others. You can get a descriptive list of all the snmp* commands using the following command:

```
$ for prog in $(cd /usr/bin; echo snmp*); do
>   whatis $prog
> done
```

4. Management Information Bases (MIBs)

In this section, we shall look at how numeric data are translated to and from names, and how the available data are documented. We shall look at how MIBs can be added to the Net-SNMP manager software. This knowledge is important when making use of data under the ‘Enterprises’ tree.

MIBs are specified in SMI, which is a subset of the ASN.1 standard. Network Management Stations (NMSs) make use of the data in MIBs (once they have been compiled in some way for fast access) to do things such as document the SMI tree and map human-readable OIDs, such as ifInOctets.1 to their numeric equivalent; .1.3.6.1.2.1.2.2.1.10.1 in this case.

Each NMS will often come with a collection of MIBs, which is stored in a software-specific location.

Often, you need to add some vendor-specific MIBs to monitor some of your devices; some agents will support vendor and/or model-specific information that is not in a well-known MIB, so you may need to search the vendor’s website for any MIB information. For example, the Apple Extreme range of WiFi access points have an MIB available. How and where you install these MIBs will depend on the particular NMS software you are using, but often they will be contained in a directory called `mibs`, in files with either a `*.txt` or `*.mib` extension.

Under Debian-based distributions, the MIBs should be found in `/usr/share/snmp/mibs/`. Go there and have a look at some of the files. Depending on what software is installed on the system, you may find other MIBs there as well: for example, if you have the Squid proxy-cache installed, you might find a MIB for Squid, which is useful for monitoring your Squid proxy-cache.

Have a look at `UDP-MIB.txt` — UDP is reasonably simple, though still has plenty of things we could track using SNMP. Here is an extract from that file:

```
-- the UDP group

udp     OBJECT IDENTIFIER ::= { mib-2 7 }

udpInDatagrams OBJECT-TYPE
   SYNTAX     Counter32
   MAX-ACCESS read-only
   STATUS     current
   DESCRIPTION
         "The total number of UDP datagrams delivered to UDP
          users.

          Discontinuities in the value of this counter can occur
          at re-initialization of the management system, and at
          other times as indicated by discontinuities in the
          value of sysUpTime."
   ::= { udp 1 }
```

Note that the line beginning with ‘udp’ is defined as (‘::=’) child number 7 under the OID known as ‘mib-2’. The OID ‘mib-2’ is imported earlier in the MIB file and is defined as ‘.1.3.6.1.2.1’ in the MIB file `SNMPv2-SMI.mib`.

The bulk of the lines in the file are for the description, which can be displayed to the user of the NMS.

‘udpInDatagrams’ is being defined as a 32-bit counter that is read-only and is not deprecated; it’s OID will be ‘udp.1’, equivalently ‘mib-2.udp.1’ or ‘.1.3.6.1.2.1.7.1’.

However, if we were to simply ‘GET’ that OID, we would find nothing there, because each object can have a number of instances; for scalar values such as ‘udpInDatagrams’ we need to use the 0th index, so we could look up ‘udpInDatagrams.0’.

Screenshot

Do this: use **snmpget** to get ‘udpInDatagrams’; what do you see? Now try getting ‘udpInDatagrams.0’; what do you see now? What if you use **snmpwalk** to get ‘udpInDatagrams’; what do you see? Take a screenshot showing what you have seen.

A number of the more interesting things available in the SMI tree are tables, these are defined as ‘SEQUENCE OF’ other things; the instance here becomes the row we want to address. Using again the example of UDP, can you find an example of a table? If the values are all zeros, use a different table — the agent doesn’t support all of the MIB.

Do this: use **snmptable** to look at the table you found. When using **snmptable** the output is generally very wide, I suggest piping it to **less -S**. You can compare the results with viewing the table of ‘ifTable’. Repeat using **snmpwalk** to see how the table relates the tree representation.

Now, on Client1, start up the program **tkmib** (if not installed, install it via the package of the same name). **tkmib** is from the same people as Net-SNMP, though can be quite awkward to use (and on smaller screens pratically useless, so make sure you have a nice, large screen).

It is worth noting that most larger NMSs are commercial and cost quite a lot; these are relatively quite simple. One nice free MIB browser for Windows is from iReasoning.

Screenshot

Practice how to perform the following, taking a screenshot to show your results for each:

- Getting the value of 'sysName.0'. You will need to add the .0 to the OID.
- Walking the 'system' sub-tree.
- Viewing the description of ipInDiscards. Why might you be interested in graphing this value?

  You're also shown that the status is Deprecated. Reading the text you find that it has been replaced by the IP version-neutral ipSystemStatsInDiscards (which you will find inside ipTrafficStats). However, you will probably find that vendors support may be slow in arriving, so it really pays to test.

  Note that graphing support in **tkmib** is not very usable in this version, so we suggest you don't try graphing with **tkmib**.

Finally, you may recall that we have set up a Squid proxy-cache, which allows querying statistics over SNMP. First, here is what you would need to edit in `/etc/squid/squid.conf` to have Squid support statistics over SNMP.

```
…  Search in the file for where to put these
snmp_port 3401
…
snmp_access allow localhost
…
```

Here is an example of walking Squid's embedded SNMP agent.

```
$ snmpwalk -m +SQUID-MIB localhost:3401 SQUID-MIB::squid
SQUID-MIB::cacheSysVMsize.0 = INTEGER: 104
SQUID-MIB::cacheSysStorage.0 = INTEGER: 11096
SQUID-MIB::cacheUptime.0 = Timeticks: (19894) 0:03:18.94
SQUID-MIB::cacheAdmin.0 = STRING: webmaster@localdomain
SQUID-MIB::cacheSoftware.0 = STRING: squid
SQUID-MIB::cacheVersionId.0 = STRING: "2.7.STABLE7"
…
SQUID-MIB::cacheRequestHitRatio.1 = INTEGER: 0     1 minute average
SQUID-MIB::cacheRequestHitRatio.5 = INTEGER: 0     5 minute average
SQUID-MIB::cacheRequestHitRatio.60 = INTEGER: 0    60 minute average
SQUID-MIB::cacheRequestByteRatio.1 = INTEGER: 0
SQUID-MIB::cacheRequestByteRatio.5 = INTEGER: 0
SQUID-MIB::cacheRequestByteRatio.60 = INTEGER: 0
…
End of MIB
```

Looking at the MIB (`/usr/share/snmp/mibs/SQUID.txt`), perhaps the most immediately useful things to look at is the cacheRequestHitRatio and cacheRequestByteRatio), as these give you a good indication of how effective the cache is at saving you traffic.

5. Long-Term Graph Trending

In this section, you will intrepret some long-term graph data captured from real-life situations. The practice in this section is in intrepreting the graphs, writing down everything you can observe from it.

Much of the value of long-term graphing is in seeing behaviour over periods such as a day, week, month and year. Daily patterns are often fairly easy to spot this way. The yearly graphs often show some useful trends. The website MRTGHelp.com [http://www.mrtghelp.com/examplemrtg.htm] has a page of useful examples of things you can see in MRTG graphs; have a look at them so you can recognise particular events. Note that MRTG can also be used to graph things like system load, mail-server throughput, and a host of others.

6. Trap Notifications

In this section, we enable the receipt of trap messages, which can help us to understand problems that our network elements may want to tell us about.

Network elements, such as wireless access points, have no user interface on which to print messages that might be useful for diagnosing problems — such as why the access-point might need rebooting — although it's not going to be able to give you much detailed information. Most of the time, the interesting trap messages will be either about authentication problems or interfaces going up or down. Such devices can and do often also have the capability to log to a remote syslog server as well.

Trap messages often end up either getting logged to somewhere (such as being forwarded to syslog), or conditionally being actioned (such as paging an administrator).

To set up the Net-SNMP trap receiver, edit `/etc/default/snmpd` and change `TRAPDRUN` to `yes`. However, if this is the only thing you do, you will see this in your system logs.

```
… snmptrapd[…]: Warning: no access control information configured.
 This receiver will *NOT* accept any incoming notifications.
```

So clearly there is more work to do; we at least need to configure the community string for traps. See the manual page snmptrapd.conf(5) for the specifications of what can go into `snmptrapd.conf`. Have a brief read of that document to ensure you understand what the following configuration does, and put this configuration into `snmptrapd.conf`.

```
snmpTrapdAddr udp:0.0.0.0:162,udp6:[::]:162
disableAuthorization yes   This is a potential security problem
doNotRetainNotificationLogs yes
doNotLogTraps no
doNotFork no
authCommunity log,execute,net public
```

Restart the SNMP services using the init-script, and check the logs for any errors. Check that **snmpd** and **snmptrapd** are listening:

```
# sudo lsof -Pni | grep snmp
snmpd     31823   snmp   7u  IPv4 3810858  UDP *:161
snmpd     31823   snmp   9u  IPv6 3810863  UDP *:161
snmpd     31823   snmp  10u  IPv4 3810864  TCP *:161 (LISTEN)
snmpd     31823   snmp  11u  IPv6 3810866  TCP *:161 (LISTEN)
snmptrapd 31826   root   8u  IPv4 3810878  UDP *:162
snmptrapd 31826   root   9u  IPv6 3810882  UDP *:162
```

Okay, so now we have our trap receiver running we should now test it by sending a trap message. We could generate a trap message ourselves using **snmptrap**, but that gets complicated and confusing; typically we would be using something that can generate its own traps. Agents typically are able to generate traps for authentication failures; agents can also generate traps for when an interface changes operational state (goes up or down). Net-

SNMP can also monitor the various tables it supports, such as the process table, and send a trap whenever something is in an error state, which could be useful. However, that requires the use of SNMPv3 internally, which I'm avoiding for this lab, so we shall simply use the authentication failure traps as an example.

So, configure **snmpd** with the following section in its configuration file `snmpd.conf` and restart it. Here I have configured trap messages to simply be sent to localhost; typically, you would configure a manager that is on an element to send to some remote NMS which is operating a trap receiver.

```
##########################################
# SECTION: Traps and Notifications
#

trapsink      localhost   public

authtrapenable  1
```

Because **snmptrapd** also uses TCP Wrappers, you will need to add a suitable line to `hosts.allow`; the service name is, as you should expect by now, `snmptrapd`.

Testing is easy, simply try to retrieve something using the wrong community string. Remember that `-c` specifies which community string to use. We're deliberate using the `wrong` community string, which should generate an authentication trap:

```
$ snmpget -c wrong localhost SNMPv2-MIB::sysUpTime.0
… should eventually timeout, probably resending …
```

Now check your logs, you should see messages similar to the following, which I have reformatted to flow nicely onto multiple lines:

```
Jan 8 15:54:00 server1 snmptrapd[21918]: 2015-01-08 15:54:00 ↩
    server-ip-address(via UDP: [127.0.0.1]:39352->[127.0.0.1]:162) ↩
    TRAP, SNMP v1, community public ↩
    #012#011.iso.3.6.1.4.1.8072.3.2.10 Authentication Failure Trap (0) ↩
    Uptime: 0:00:23.44#012
```

When you can, I suggest you configure an SNMP trap handler in a real network and get any network elements such as routers and wireless access points, to send any notications to it.

We've only touched on traps in this brief introduction. Our trap-receiver can also run handlers when particular traps arrive; a handler may do something like send an e-mail or page, which is much faster than waiting for any log-processing to happen periodically.

7. Optional: Installing the Microsoft Windows SNMP Agent

Windows comes with an optional component which allows you to query information about the host using SNMP. This section shows you how you can install and configure the required components. By the end of this optional section, you should have gained some small amount of Windows service administrative ability and have gained experience with another piece of agent software.

The instructions for this procedure can be found in Microsoft's Knowledge Base article Q324263 [http://support.microsoft.com/kb/324263].

8. Self-Assessment

This is an optional lab, therefore there are no marks available for this lab. However, it would be useful to check your knowledge, as having done this lab will cement your knowledge about the topic, which can appear in the exam.

1. Briefly describe the following terms as they relate to the SNMP conceptual model and architecture: ‘agent’, ‘network element’, ‘manager’, ‘network management station’, ‘community string’, ‘SMI tree’, ‘OID’, ‘MIB’ and ‘trap’.

Describe how the process of authentication is handled in SNMP versions 1 and 2c.

Describe the SNMP messages used to walk a part of the tree for both SNMP versions 1 and 2c.

2. What is the difference between **snmpget** and **snmpgetnext**?

Why would an OID specified as ‘sysContact’ *not* work with **snmpget**? What would be needed instead?

Does this mean we should use **snmpgetnext** instead of **snmpget**?

3. Ensure you have taken screenshots of using an MIB browser to do the following operations:

- Getting the value of ‘sysName’.
- Walking the ‘system’ sub-tree.
- Viewing the description of ‘ipInDiscards’. Why might you be interested in graphing this value?

4. When would short-term graphing be more useful compared to long-term graphing?

5. The teaching staff should have pointed you to some real-life MRTG graphs which may (or may not) show some useful information.

Look at the daily, weekly, monthly and yearly graphs provided, and write down as many things as you can reasonably infer from them.

9. Final Words

There are a number of useful things we have not looked at in this lab. The first is SNMPv3, which changes the authentication model to something much more secure. The second is extending the Net-SNMP agent to enable monitoring of things not otherwise supported by the agent. The Net-SNMP web-site has a number of tutorials, such as for SNMPv3 Options [http://www.net-snmp.org/wiki/index.php/TUT:SNMPv3_Options] and a useful page with examples [http://www.net-snmp.org/wiki/index.php/Net-snmp_extensions] of the various extension mechanisms.

Finally, SNMP is only one part of a network monitoring solution. Take a look at a more full-blown solution such as OpenNMS or Nagios.