snmpproxy

The SNMP proxy is a SNMPv1 capable proxy. It will receive valid SNMPv1 packages and based on a simple set of rules return a value. If the package doesn't match any rule it will be relayed to the screened entity. The response will be returned to the client.
The proxy will only match *Get* and *GetNext* requests – other packages will be silently dropped.
When running as a daemon, errors and other unexpected behaviors are logged to the `syslog` daemon at facility `LOG_DAEMON`.

Usage: `snmpproxy [OPTIONS] configurationfile`

OPTIONS

`-D{acflot}`	Dump configuration.
`-d`	Turn on debugging.
`-F`	Stay in foreground – do not become a daemon.
`-h`	Write help.
`-v`	Write version information.

Configuration

Invoking snmpproxy with the `-D` switch will force snmpproxy to exit after parsing the configuration file. Depending on the tokens given to the `-D` switch the following diagnostics will be written:

Token	Description
`a`	Show all configuration details.
`c`	Show only the details about the directives belonging to the Global section.
`f`	Show Fake lists.
`l`	Show GetNext lists.
`o`	Show OID entries.
`t`	Show access lists.

It is possible to specify more than one token. F.ex. `-Dfl` will show the Fake lists and the GetNext lists.

Configuration File

The configuration file consists of 5 virtual sections. These are:

- Global
- Access & Security
- OID entries
- GetNext Lists
- Fake Lists

It is perfectly valid to mix the various sections. The sections do not have a section header. The term 'virtual section' only exists for documentary purposes.
All directives in the configuration file are case insensitive – that is, `Port`, `PORT` and `port` are all equal.
Directives in the Global and Access & Security sections can only be specified once unless otherwise noted. Directives in the other sections can be specified one or more times.

Section: Global

The Global section contains directives which influences on how the proxy is started and certain behavior while running:

Directive	Description
`Port`	This is the port number the proxy will listen on for incoming packages from the client. The default is **`16000.`**
`Debug`	Set to {`yes`|`on`|`true`|`1`} to turn debugging on. Set to {`no`|`off`|`false`|`0`} to turn debugging off. Default is **`0`**.
`DumpMessages`	Set to {`yes`|`on`|`true`|`1`} to turn dump message to the `CaptureFile` (see below). Set to {`no`|`off`|`false`|`0`} to suppress dumping of messages. Default is **`0`**.
`PidFile`	The path to where the proxy will write its Process ID. The path must be fully qualified (path + file name). Default is **`/var/run/snmpproxy.pid`**.
`CaptureFile`	The path to all debugging messages. Default is **`/tmp/snmpproxy.pkg`**.

`Peer`	This is the name and/or port of the screened entity. The format is *name:port*. The name can be expressed as a DNS name or an ip-address. The port can be expressed as a `/etc/services` name or an integer. Defaults to **`localhost:161`**.
`CacheName`	This directive controls how long a DNS is kept in the internal name cache. The time is measured in seconds. The default is **`86400`**.
`Retry`	This directive controls how many time the screened entity is queried before returning an error message to the client. By setting this value to `0` no retries will be performed. This will put the control in the hands of the client. Default is **`5`**.
`TimeOut`	This is the timeout between queries to the peer. This directive is only meaningful when Retry > 0! Default is **`1`**.

Section: Access & Security

This section contains directives which influence who can access this proxy and in what environment the proxy runs.

Directive	Description
`Trusted`	This directive can be used several times. This controls the networks or hosts which can access this proxy. The format of the directive is *network/mask*. The mask can be specified as a bit count (f.ex. 16) or in dotted quad notation (f.ex. 255.255.0.0). The network park is always an IP-address. The default is an empty list, permitting all access.
`Community`	The SNMPv1 community used. Default is **`public`**.
`User`	The user used to run the proxy as. Default is **`bin`**.

Section: OID entries

This section is where the rules for OID lookup is specified. The format is:

```
.oid action
```

The action is separated from the OID with one or more white spaces.
The OID must be specified in numeric form (eg. .1.4.3.1.1.1.6) and it must start with a dot before being recognized as an OID rule. The action can be of the following:

Action	Description
`.oid`	Masquerade as OID. The input OID will be rewritten to the supplied OID. The package is relayed to the screened entity. The reply package is rewritten by exchanging the supplied OID with the original OID. This is the method to mask/rewrite an OID.
`|/path/to/program`	Execute the specified program and return output. The program must return one line containing the action on stdout, The action is among the ones listed except for .oid and |/path/to/program.
`i:address`	Return an internet address (dotted quad notation).
`c:counter`	Return a counter value.
`g:gauge`	Return a gauge value.
`n:number`	Return an integer value.
`t:ticks`	Return a time tick.
`a:arbitrary`	Return a arbitrary value. Use to return binary data. The data is passed as is.
`o:object`	Return an OID.
`s:string`	Return a string.
`0`	Return a NULL type.

Examples

To return the number 100:

```
    .1.2.3.4.5.6.7 n:100
```

To return the number from a program:

```
    .1.2.3.4.5.6.7 |/path/to/myprogram.sh
```

myprogram.sh:

```
echo “n:100”
exit 0
```

To return the value of a masqueraded OID from the peer:

```
.1.2.3.4.5.6.7 .2.3.4.5.6.7.8
```

If the OID .2.3.4.5.6.7.8 on the peer return the string “Hello”, the return value when querying .1.2.3.4.5.6.7 will be s:”Hello”.

Section: GetNext Lists

This is where the GetNext lists are defined. The snmpproxy has no idea of MIBS and do not know the logical sequence of OIDs. A GetNext list defines the sequence of oids. The list contains a start entry and the OIDs:

```
GetNext .1.3.6.1.2.1.2.2.1.2 .1 .2 .3 .4 .393222 .40000
```

This defines the a GetNext list with the entries:

```
.1.3.6.1.2.1.2.2.1.2.1
.1.3.6.1.2.1.2.2.1.2.2
.1.3.6.1.2.1.2.2.1.2.3
.1.3.6.1.2.1.2.2.1.2.4
.1.3.6.1.2.1.2.2.1.2.393222
.1.3.6.1.2.1.2.2.1.2.40000
```

Section: Fake Lists

When doing a snmpwalk on nonexistent lists you will need to create a FakeList entry. The entry basically defines the start OID of a fake list:

```
FakeList  .1.3.6.1.2.1.31.1. .1.3.6.1.2.1.31.1.6000
```

This translates to: use .1.3.6.1.2.1.31.1.6000 as the start entry for the .1.3.6.1.2.1.31.1 list. This list should be defined using a GetNext directive.

Theory of Operation

When started, snmpproxy will first parse the command line. Then the configuration file is loaded. If the `-D` switch is given on the command line, dump the appropriate configuration(s) and exit. Otherwise, become a daemon unless the `-F` switch has been given.

Drop the privileges of the user, create the PID file and enter an infinite loop listening and acting on input packages from the network.

When a package has been received parse the package. If the request is a Get request lookup the OID in the OID entries. If a match is found, return the value. Otherwise send the OID to the screened entity and return the value from the entity.

If the request is a GetNext request, 'calculate' (by use of the GetNext lists and FakeList lists) the next OID. Look up the OID in the OID entries. If a match is found, return the value. Otherwise send the OID to the screened entity and return the value from the entity.

There is no soft limit on the number of OIDs in the input package. The output package is restricted to the limits as set forth in *RFC1155* and *RFC1157*.