NAME

Munin::Plugin::SNMP − Net::SNMP subclass for Munin plugins

SYNOPSIS

The Munin::Plugin::SNMP module extends Net::SNMP with methods useful for Munin plugins.

SNMP CONFIGURATION

SNMP plugins (that use this module) share a common configuration interface implemented in the function *session()*. Please see the documentation for that function for complete instructions and examples on how to configure SNMP. The documentation is located there to ensure that it is up to date and matches the code.

DEBUGGING

Additional debugging messages can be enabled by setting `$Munin::Plugin::SNMP::DEBUG`, `$Munin::Plugin::DEBUG`, or by exporting the `MUNIN_DEBUG` environment variable before running the plugin (by passing the `--pidebug` option to `munin-run`, for instance).

METHODS

config_session() − **Decode environment to get the needed plugin configuration parameters**

```
($host, $port, $version, $tail) = Munin::Plugin::SNMP->config_session();
```

This is a convenience function for the ‘‘config’’ part of the plugin − it decodes the environment/plugin name to retrieve the information needed in the configuration phase. It returns a 4 tuple consisting of:

1) the host name
2) the UDP port to use
3) the SNMP version to use (3 for version 3, 2 for version 1 or 2c)
4) the tail of the plugin name: whatever is left of the plugin name after ‘‘snmp_<host>_’’.

The tail can be interesting for the ‘‘fetch’’ part of the plugin as well.

session([optional Net::SNMP options]) − create new Munin::Plugin::SNMP object

```
$session = Munin::Plugin::SNMP->session();
```

This method overrides the Net::SNMP constructor to get the connection information from the plugin name and/or environment. Please note that no error string is returned. The function handles errors internally − giving a error message and calling die. Calling die is the right thing to do.

The host name is taken from the plugin symlink, which must be on the form `snmp[v3]_<hostname>_<plugin_name>[_args]`.

The ‘‘v3’’ form is taken to mean that SNMPv3 is to be used. It is also a name trick providing a separate ‘‘namespace’’ for devices that use SNMPv3 so it can be configured separately in munin/plugin−conf.d/ files. E.g.:

```
[snmp_*]
   env.version 2
   env.community public

[snmpv3_*]
   env.v3username snmpoperator
   env.v3authpassword s3cr1tpa55w0rd
```

See below for how to configure for each different case. The first case above shows Munin’s default configuration.

NOTE: munin-node-configure does not yet utilize the ‘‘v3’’ thing.

The following environment variables are consulted:

`env.host`

If the plugin name (symlink) does not contain the host name this is used as the host name to connect to.

The host name must be specified, but is usually specified in the plugin name. If the hostname somehow does not resolve in DNS (or the hosts file) it is possible to do this:

```
[snmp_*]
   env.version 2c
   env.community floppa

[snmp_switch1.langfeldt.net]
   env.host 192.168.2.45

[snmp_switch2.langfeldt.net]
   env.host 192.168.2.46
```

`env.port`
The port to connect to. Default 161.

`env.timeout`
The timeout in seconds to use. Default 5.

`env.version`
The SNMP version to use for the connection. One of 1, 2, 3, snmpv1, snmpv2c or snmpv3. SNMP v2 is better as it supports bulk operations. Therefore 2 is the default in `Munin::Plugin::SNMP`. If your device supports v3 that may be even better as it supports proper security − but the encryption may slow things down.

Security is handled differently for versions 1/2c and 3. See below.

SNMP 1/2c authentication

`env.community`
The community name for version 1 and 2c agents. The default is 'public'. If this works your device is probably very insecure and needs a security checkup.

SNMP 3 authentication

SNMP v3 has three security levels. Lowest is `noAuthNoPriv`, which provides neither authentication nor encryption. If a username and `authpassword` are given it goes up to `authNoPriv`, and the connection is authenticated. If `privpassword` is given the security level becomes `authPriv` − the connection is authenticated and encrypted.

Note: Encryption can slow down slow or heavily loaded network devices. For most uses `authNoPriv` will be secure enough — the password is sent over the network encrypted in any case.

`Munin::Plugin::SNMP` does not support ContextEngineIDs and such for authentication/privacy. If you see the need and know how it should be done please send patches!

For further reading on SNMP v3 security models please consult RFC3414 and the documentation for Net::SNMP.

If version is set to 3 or snmpv3 the following variables are used to define authentication:

`env.v3username`
Username. There is no default.

`env.v3authpassword`
Authentication password. Optional when encryption is also enabled, in which case defaults to the privacy password (`env.v3privpassword`). The password is sent encrypted (one way hash) over the network.

`env.v3authprotocol`
Authentication protocol. One of 'md5' or 'sha' (HMAC−MD5−96, RFC1321 and SHA−1/HMAC−SHA−96, NIST FIPS PIB 180, RFC2264). The default is 'md5'.

`env.v3privpassword`
Privacy password to enable encryption. An empty ('') password is considered as no password and will not enable encryption.

Privacy requires a v3privprotocol as well as a v3authprotocol and a v3authpassword, but all of these are defaulted (to 'des', 'md5', and the v3privpassword value, respectively) and may therefore be left unspecified.

`env.v3privprotocol`

If the v3privpassword is set this setting controls what kind of encryption is used to achieve privacy in the session. Only the very weak 'des' encryption method is supported officially. The default is 'des'.

The implementing perl module (Net::SNMP) also supports '3des' (CBC-3DES-EDE aka Triple-DES, NIST FIPS 46-3) as specified in IETF draft-reeder-snmpv3-usm-3desede. Whether or not this works with any particular device, we do not know.

get_hash() − **retrieve a table as a hash of hashes**

```
$result = $session->get_hash(
                        [-callback        => sub {},]     # non-blocking
                        [-delay           => $seconds,]   # non-blocking
                        [-contextengineid => $engine_id,] # v3
                        [-contextname     => $name,]      # v3
                        -baseoid          => $oid,
                        -cols             => \%columns
                      );
```

This method transforms the -baseoid and -cols to a array of -columns and calls `get_entries()` with all the other arguments. It then transforms the data into a hash of hashes in the following manner:

The keys of the main hash are the last element(s) of the OIDs, after `$oid` and the matching keys from `%columns` are removed. The values are hashes with keys corresponding to the values of `%columns` hash and values from the subtables corresponding to the keys of `%columns`.

For this to work, all the keys of `-cols` must have the same number of elements. Also, don't try to specify a next-to-next-to-leaf-node baseoid, the principle it breaks both `get_entries` and the logic in `get_hash`.

If (all) the OIDs are unavailable a defined but empty hashref is returned.

Example:

```
$session->get_hash(
             -baseoid => '1.3.6.1.2.1.2.2.1', # IF-MIB
             -cols    => {
                           1 => 'index',
                           2 => 'descr',
                           4 => 'mtu',
                         }
           );
```

given the following SNMP table:

```
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifDescr.1 = STRING: lo0
IF-MIB::ifDescr.2 = STRING: lna0
IF-MIB::ifType.1 = INTEGER: softwareLoopback(24)
IF-MIB::ifType.2 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifMtu.1 = INTEGER: 32768
IF-MIB::ifMtu.2 = INTEGER: 1500
...
```

will return a hash like this:

```
'1' => {
        'index' => '1',
        'mtu' => '32768',
        'descr' => 'lo0'
       },
'2' => {
        'index' => '2',
        'descr' => 'lna0',
        'mtu' => '1500'
       }
```

get_single() − **Retrieve a single value by OID**

```
$uptime = $session->get_single("1.3.6.1.2.1.1.3.0") || 'U';
```

If the call fails to get a value the above call sets `$uptime` to 'U' which Munin interprets as ''Undefined'' and handles accordingly.

If you stop to think about it you should probably use `get_hash()` (it gets too much, but is good for arrays) or `get_entries()` − it gets exactly what you want, so you mus

get_by_regex() − **Retrive table of values filtered by regex applied to the value**

This example shows the usage for a netstat plugin.

```
my $tcpConnState = "1.3.6.1.2.1.6.13.1.1.";
my $connections = $session->get_by_regex($tcpConnState, "[1-9]");
```

It gets all OIDs based at `$tcpConnState` and only returns the ones that contain a number in the value.

TODO

Lots.

BUGS

Ilmari wrote: `get_hash()` doesn't handle tables with sparse indices.

Nicolai Langfeldt: Actually I think it does.

SEE ALSO

Net::SNMP

AUTHOR

Dagfinn Ilmari Mannsåker, Nicolai Langfeldt Rune Nordbøe Skillingstad added timeout support.

COPYRIGHT/License.