Tivoli Netcool Supports Guide to the MTTrapd [SNMP] Probe

by
Jim Hutchinson

Document release: 4.0

Table of Contents

1 Introduction

1.1 Overview

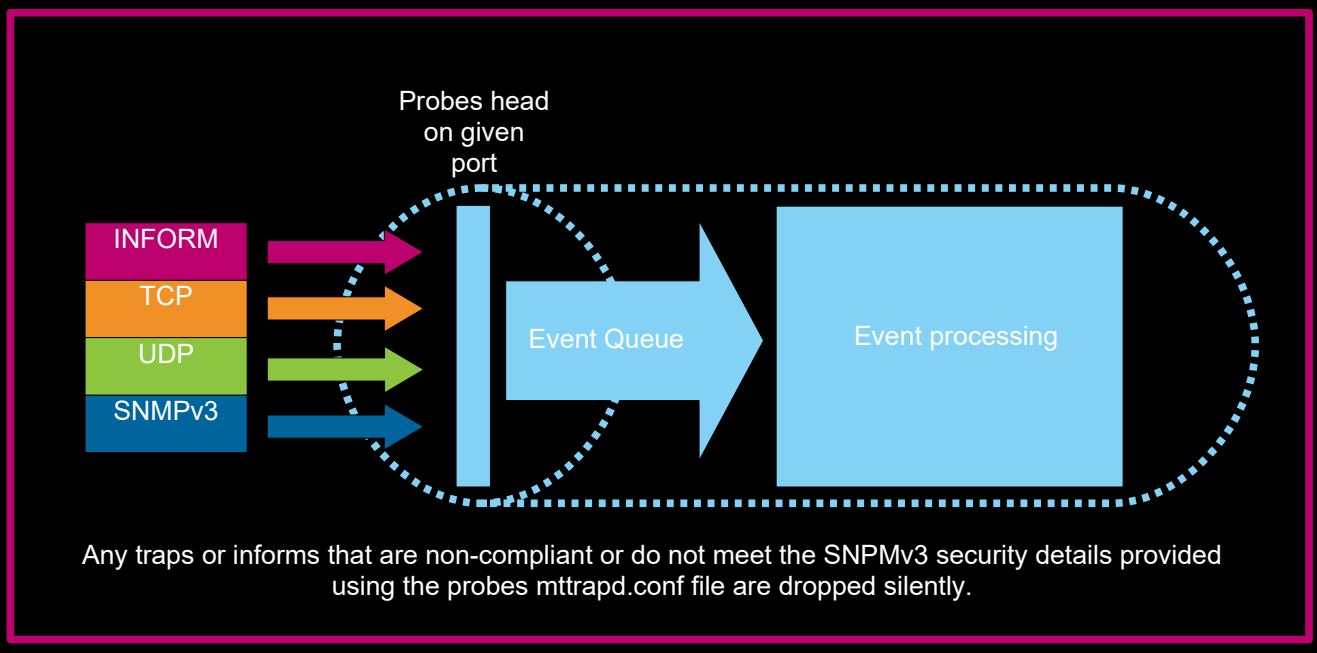
The MTTrapd (SNMP) probe is a generic probe used to process SNMP traps and informs. It is a multi-threaded probe and supports both UDP and TCP.

The SNMP versions supported are;

- SNMPv1 traps
- SNMPv2c traps
- SNMPv3 traps
- SNMPv2c informs
- SNMPv3 informs

1.2 Event processing

The Multi-threaded nature of the MTTrap probe receives traps from various source types, parses them using the Net-SNMP API and places the SNMP RFC Compliant traps on the probes queue for processing:



2 Performance

2.1 Name Resolution

Name resolution takes time, therefore turning it off improves performance;
UNIX

```
NoNameResolution : 1
```

Windows:

```
NoNetbiosLookups : 1
```

The latest version of the probe supports a new feature that allows DNS lookups to be managed via the probe property settings. This allows the DNS lookups to be performed.

For example

```
#
# Built-in Hostname cache
#
NoNameResolution          : 0
HostnameTableSize         : 50000
ActiveHostnameDuration    : 30
RefreshHostnameInterval   : 1440
```

These settings allow the probe to hold 50,000 lookups in memory, and refresh them every 24 hours.
Please refer to the probes manual to better understand these settings and their impact.

2.2 UDP Traps

UDP traps do not report an error at the client or server if they are lost or dropped. It is therefore recommended that TCP traps are used where possible. You can increase the probes socket size for UDP traps using the SocketSize property. For the 32-bit probe the maximum setting is 65535.

For the 64-bit probe this limit is a lot higher, with a recommended setting of up to '256000', although the value should not be set higher than the operating systems maximum allowed size.

You can check the operating systems buffer settings using operating system commands.

For example in Linux check the buffer sizes use:

```
$ sysctl net.core.rmem_max
net.core.rmem_max = 212992
$ sysctl net.core.rmem_default
net.core.rmem_default = 212992
```

To increase these operating system settings to higher values use the /etc/sysctl.conf file:

```
net.core.rmem_max=26214400
net.core.rmem_default=26214400
```

To update the settings manually use the command settings:

```
$ sudo sysctl -w net.core.rmem_max=26214400
net.core.rmem_max = 26214400
$ sudo sysctl -w net.core.rmem_default=26214400
net.core.rmem_default = 26214400
```

2.2.1 Dropping UDP traps silently

UDP traps can be dropped silently at the operating system. The interface statistics will indicate when packets are being dropped [e.g. ifconfig -a]. On Linux the /proc/net/udp and /proc/net/udp6 files can be used to monitor the 'drops' for a specific probe instance, based on the runtime userid [uid].

Example /proc/net/udp file details.

```
  sl  local_address rem_address   st tx_queue rx_queue tr tm->when retrnsmt   uid  timeout inode ref pointer
drops
  370: 00000000:065D 00000000:0000 07 00000000:00000000 00:00000000 00000000   810        0 54779 2
0000000000000000 0
```

If traps are being dropped at the interface, in the operating system, it indicates that the probe is having problems processing the events. Since the probe is unable to read the traps from the UDP socket quickly enough. Increasing the operating systems and probes UDP socket size allows the probe to consume more or larger traps. However the probe must have enough time to process the traps into events, and forward them to the object server as well.

Consider splitting the trap load over multiple MTTrapd probes where the event processing is a major factor, as the rules file processing is a single threaded operation. Using multiple MTTrapd probes allows this bottleneck to be overcome.

Investigate using a load balancer or trap forwarder. Remember to consider the impact of probe outages, and the way in which the sending of UDP traps are managed.

2.3 Replacing hostname calls

You can reduce the number of calls made to hostname and the probes hosts IP Address using a one time lookup and use static variables instead when the values are required.

```
if(match(%MyHostname,""))
{
        %MyHostname = hostname()
        %MyIPHostname = gethostaddr(%MyHostname)
}
#
# Replaced host functions that were called for each event
# with the %static host variables
#
@Manager = %Manager + "@" + %MyIPHostname

# Logging for debug and testing only
log(debug,"MYDEBUG: %MyHostname = " + %MyHostname)
log(debug,"MYDEBUG: %MyIPHostname = " + %MyIPHostname )
```

2.4 Buffering

Enabling buffering on probes allows a busy probe to send events in blocks, based on BufferSize, which is much more efficient for an object server that needs to manage more than a few clients. Ensure the FlushBufferInterval is set to a value that is suitable for the environment [Typically from 1-60];

e.g.

```
Buffering                   : 1
BufferSize                  : 200
FlushBufferInterval         : 9
```

The above example is for an expected event rate of 20 events per second, with the FlushBufferInterval set to an odd value close to the desired period [10s] so as to reduce timing clashes.

e.g. 20 * 10 = 200 events seen in 10s

2.5 SNMPv3 configuration

It is recommended to create a directory to hold the source SNMPv3 security configuration file per probe instance (ConfPath). This allows the source configuration file to be maintained, rather than overwritten. The ConfPath and PersistentDir should be secured as required using the operating systems file system permissions.

e.g.

```
mkdir -p $OMNIHOME/var/snmpv3/conf
vi $OMNIHOME/var/snmpv3/conf/mttrapd.conf
###
# createUser -e engineid username [MD5|SHA|SHA256] passphrase [DES|AES|AES256] passphrase
#
createUser -e 010203040506 trapuser  MD5 md5password DES despassword
# EOF
```

:wq

```
vi $OMNIHOME/probes/Solaris2/mttrapd.props

ConfPath                 : '$OMNIHOME/var/snmpv3/conf'
PersistentDir            : '$OMNIHOME/var/snmpv3'
```

:wq

2.5.1 SNMPv3 traps

SNMPv3 traps entries in the ConfPath mttrapd.conf file require unique engineid entries.
This can cause the mttrapd.conf to become large, depending upon the number of trap senders.
If there are more than 20k engineid's to be processed, it may be necessary to split the load over multiple MTTrapd probes, since it can take the probe a significant amount of time to create the PersistentDir mttrapd.conf file. After which, it is ready to focus on processing the incoming traps.

```
###
# createUser -e engineid username [MD5|SHA|SHA256] passphrase [DES|AES|AES256] passphrase
#
createUser -e 010203040501 trapuser1  MD5 md5password123 DES despassword123
createUser -e 010203040503 trapuser3  SHA md5password123 AES despassword123
createUser -e 010203040505 trapuser5  SHA256 md5password123 AES256 despassword123
```

2.5.2 SNMPv3 informs

SNMPv3 INFORMS require two way communication with the INFORM senders, so do not require SNMPv3 engineid's for each sender. Since the sender verifies it sent the events details.

```
###
# createUser -e engineid username [MD5|SHA|SHA256] passphrase [DES|AES|AES256] passphrase
#
createUser informuser1  MD5 md5password123 DES despassword123
createUser informuser2  SHA md5password123 AES despassword123
createUser informuser3  SHA256 md5password123 AES256 despassword123
```

2.5.3 SnmpConfigChangeDetectionInterval

The property SnmpConfigChangeDetectionInterval determines the periodicity of the checks on the ConfPath mttrapd.conf file. The default is to check every minute and see if there are new entries to parse. At message level informational the probe will log these checks.

- No updates

```
Config File is old and will not be parsed.
```

- ConfPath mttrapd.conf file was read

```
Processing config file completed.
User credentials have been refreshed.
```

With the default property settings you can update an existing engineid by first removing the entry from the ConfPath mttrapd.conf file, and waiting for the 'refreshed' message. After this the entry can be reinstated into the ConfPath mttrapd.conf, after which another 'refreshed' message is seen, and the new details added to the probes PersistentDir mttrapd.conf. This can be confirmed by checking the entry disappearing from the PersistentDir mttrapd.conf when it is commented out in the ConfPath mttrapd.conf file. Ensure that the process control agent uses the higher values for RetryTime and RogueTimeout.

2.6 Measuring loads

The MTTrapd probes performance can be measured using rules file commands.
e.g.

```
if ( match(%load,"") )
{
        %load = "60.60"
}
else
{
        %load = updateload(%load)
        $current_load = getload(%load)
        log(INFO,"Average Events per second = " + $current_load)
}
```

Alternatively they can be calculated manually using maths functions.

```
e.g.
if ( match(%eps_counter,"") )
{
        %eps_counter = 1
        %start_time=getdate
}
else
{
        %end_time=getdate
        $time_elapsed = real(int(%end_time) - int(%start_time))

        if (int($time_elapsed) > 59 )
        {
                $current_load = real(%eps_counter) / real($time_elapsed)
                log(warn,"EPS: " + $current_load + " " + (%eps_counter) + " [ " + $time_elapsed + " ]")
                %eps_counter = 1
                %start_time=getdate
        }
        else
        {
                %eps_counter = int(%eps_counter) + 1
        }
}
```

If more complex load gathering is required, please refer to the Netcool/OMNIbus probes and gateway manual under ‘Enabling self monitoring of probes’.

2.7 Monitoring load

The probe does allow specific features to be monitored using the probe property values.
Enable the property LogStatisticsInterval, by setting a time in seconds, to see the queue size and number of traps processed.
e.g.

```
LogStatisticsInterval : 60


Error: E-UNK-000-000: Trap queue size is 0
Error: E-UNK-000-000: Inform queue size is 0
Error: E-UNK-000-000: Number of traps read in the last 59 seconds: 0
Error: E-UNK-000-000: Number of traps processed in the last 59 seconds: 0
```

2.8 TrapStat property

The MTTrapd probe has the TrapStat property for logging further information at MessageLevel warning.

Set the TrapStat probe property to 1 to use the features.

```
TrapStat                : 1
```

2.8.1 Internal queue used

The following rules file logic can be used to log the internal queue size, when It is used at message level warning.

```
# Log when internal queue is used
# Value returned is an absolute value
$tf_inqueue = get_queue_size()
if ( int($tf_inqueue) > 0 )
{
   log( WARN, "TRAPSTATS : " + $tf_inqueue + " traps in the queue." )
}
```

2.8.2 Traps dropped

The following rules files logic can be used to log the number of traps dropped from an Ipaddress.

```
# The value returned is the running total for the given source IP Address
#
if ( exists($IPaddress) )
{
  $tf_drop = read_drop_count($IPaddress)
  if ( int($tf_drop) > 0 )
  {
    log( WARN, "TRAPSTATS : " + $tf_drop + " traps dropped from " + $IPaddress)
  }
}
```

2.8.3 Trap statistics reporting

The details from the TrapStats data can be logged in a report format periodically. By storing the $IPaddress in an array, the periods data can be pulled for each one, after a period of time. Additionally counting the number of traps per $IPaddress is useful when attempting to determine which senders are causing high event rates.

The example given allows these details to be logged for probe load analyis.

```
Warning: W-UNK-000-000: EPS: 78.363636 (Total=2156746)
Warning: W-UNK-000-000: TRAPSTATS : Traps from 192.168.192.61 = 213
Warning: W-UNK-000-000: TRAPSTATS : Traps from 192.168.192.64 = 212
Warning: W-UNK-000-000: TRAPSTATS : Traps from 192.168.192.66 = 217
Warning: W-UNK-000-000: TRAPSTATS : Traps from 192.168.192.32 = 204
```

```
# Arrays must be at the top of main rules file
array ArrayOfIPAddress
array ArrayOfTrapCounts

# Storing IPaddress details
if (exists($IPaddress))
{
# Array of IPaddress
  if ( match(ArrayOfIPAddress[$IPaddress],"") )
  {
   ArrayOfIPAddress[$IPaddress] = $IPaddress
  }

# Counting traps
  if (match(ArrayOfTrapCounts[$IPaddress], "" )) {
   ArrayOfTrapCounts[$IPaddress] = 1
  } else {
   ArrayOfTrapCounts[$IPaddress] = int(ArrayOfTrapCounts[$IPaddress]) + 1
  }
}
# Determine the events per second
if ( match(%event_counter,"") )
{
        %start_time=getdate
        %event_counter = 1
        %total_event_counter = 1
}
else
{
        %end_time=getdate
        $time_elapsed = int(%end_time) - int(%start_time)
        %total_event_counter = int(%total_event_counter) + 1

        if (int($time_elapsed) > 10 )
        {
                $calculated_load = real(%event_counter) / real(int(%end_time) - int(%start_time))
                log(WARN,"EPS: " + $calculated_load + " (Total=" + %total_event_counter + ")" )
                %event_counter = 1
                %start_time=getdate
        }
        else
        {
                %event_counter = int(%event_counter) + 1
        }
}
# Use the $time_elapsed) to log a report for the trap queue and dropped traps
if (int($time_elapsed) > 10 )
{
# Log if internal queue is used
# Value returned is an absolute value
$tf_inqueue = get_queue_size()
if ( int($tf_inqueue) > 0 )
{
log( WARN, "TRAPSTATS : " + $tf_inqueue + " traps in the queue." )
}
# Log out dropped traps
 foreach ( n in ArrayOfIPAddress )
 {
     $IPaddress = ArrayOfIPAddress[n]
     $tf_drop = read_drop_count($IPaddress)
     if ( int($tf_drop) > 0 )
     {
       log( WARN, "TRAPSTATS : " + $tf_drop + " traps dropped from " + $IPaddress)
     }
 }
# High number of traps from $IPaddress
 foreach ( n in ArrayOfIPAddress )
 {
  $IPaddress = ArrayOfIPAddress[n]
  if ( int(ArrayOfTrapCounts[$IPaddress]) > 100 )
  {
       log( WARN, "TRAPSTATS : Traps from " + $IPaddress + " = " + ArrayOfTrapCounts[$IPaddress])
  }
# Null arrays for period
  ArrayOfTrapCounts[$IPaddress]=0
  ArrayOfIPAddress[n] = ""
 }
}
```

2.8.4 Event rates per Ipaddress

Logging out an event rate for a node is useful for troubleshooting.
In this case the period is defined by the probe property ProbeWatchHeartbeatInterval, at which time the arrays are reset.
The event rate is calculated when the elapsed time exceeds the eps_min_period setting, and the log message sent when the calculated event rate exceeds the eps_threshold.
This method reduces the risk false reporting.

```
File : probewatch_node_load.rules

####################################
# Arrays at the top
####################################
array a_node_start_time;
array a_node_counter;
###
# Define the EPS settings
# ProbeWatchHeartbeatInterval > eps_min_period
# eps_min_period is a factor of ProbeWatchHeartbeatInterval
# eps_threshold is the event rate to report on
#
$eps_min_period = 10
$eps_threshold  = 20

###
# ProbeWatch logic
#
# case "Heartbeat ...": is where the statistics reset is
#
if( match( @Manager, "ProbeWatch" ) )
{
# Default Probewatch handling
switch(@Summary)
{
case "Running ...":
        @Severity = 1
        @AlertGroup = "probestat"
        @Type = 2
case "Going Down ...":
        @Severity = 5
        @AlertGroup = "probestat"
        @Type = 1
case "Rules file reread upon SIGHUP successful ...":
        @Severity = 1
        @AlertGroup = "Probe management"
        @Type = 2
####################################
# Heartbeat event - ProbeWatchHeartbeatInterval
####################################
case "Heartbeat ...":
# Reset Node load counters
clear(a_node_start_time)
clear(a_node_counter)
%end_time=getdate
log(WARN,"EPS: Reset @ " + %end_time )
####################################
# End Heartbeat event
####################################
default:
        @Severity = 1

} # End of switch(@Summary)
####################################
# Main event processing
####################################
} else {

###
# Main rules file logic processing
```

```
###
# EPS Node logging
####################################
# Node load processing
####################################
# Set the Node load token to use to measure on
###
# $IPaddress is used bythe mttrapd probe
$NodeLoad = $IPaddress

# Initialise arrays
if ( match(a_node_counter[$NodeLoad],"") )
{
   a_node_start_time[$NodeLoad] = getdate
   a_node_counter[$NodeLoad] = 1
# Count events per unique $NodeLoad
} else {
   a_node_counter[$NodeLoad] = int(a_node_counter[$NodeLoad]) + 1

  %end_time=getdate
  $time_elapsed = int(%end_time) - int(a_node_start_time[$NodeLoad])

# At least DELTA seconds must elapse for logging
# ProbeWatchHeartbeatInterval > DELTA
  if (int($time_elapsed ) > int($eps_min_period) )
  {
    $calculated_load = real(a_node_counter[$NodeLoad]) / real ($time_elapsed)
###
# Report issue for Node when EPS exceeds eps_threshold value
    if (int($calculated_load) > int($eps_threshold) )
    {
      log(WARN,"EPS: Node = " + $NodeLoad + " : " + $calculated_load + " : [ " + a_node_counter[$NodeLoad] +
" ]" )
# Reset this Node load
      a_node_start_time[$NodeLoad] = getdate
      a_node_counter[$NodeLoad] = 1
    }
  }
}

}
# EOF
```

2.9 Architectures

The MTTrapd probe is susceptible to trap sender problems, as it listens on an open for and queues events for processing on a single queue. When deploying the MTTrapd probe it is important to consider the volume of traps the probe is going to process, the traps sources performance, network congestion and vulnerability.

2.9.1 Mixing Traps and INFORMS

Using a single MTTrapd probe probe or P2P failover pair for all traps and informs is not recommended.

INFORMs require a response from the MTTrapd probe, which can cause delays in event processing. Therefore it is recommended that a dedicated probe or P2P failover pair is used for INFORMs, especially when the volume of SNMP senders is large.

2.9.2 Mixing protocols

Although the MTTrapd probe is capable of receiving both TCP and UDP at the same time, this may cause problems with event reception, under high loads. TCP is the best protocol to use, where event deliver needs to be guaranteed. TCP is best for SNMPv3 traps as UDP SNMPv3 traps will be discarded silently.

2.9.3 Mixing SNMP versions

SNMPv3 is a secure variant of SNMP with SNMP traps being the most secure. With this in mind, it makes sense to use a dedicated MTTrapd probe or P2P failover pair for SNMPv3, and configure the probes properties to ensure only SNMPv3 traps are received.

- Snmpv3ONLY
- snmpv3MinSecurityLevel

2.9.4 Understanding SNMP data

SNMP traps and INFORMs can be created periodically, during a system restart or shutdown, excessively due to misconfiguration, and contain large volumes of textual data. When designing the collection of SNMP data it is important to understand the impact of missing data, and how best to manage unusual circumstances.
When the MTTrapd probe is placed into debug mode, it logs all of the tokens for each event, as well as event processing details. During the planning phase of the MTTrapd probe deployment it is recommended that several hours of MTTrapd probe logs are collected from the network for analysis; Although it is understood that typically, this type of analysis will occur after deployment, and usually after problems have already occurred.

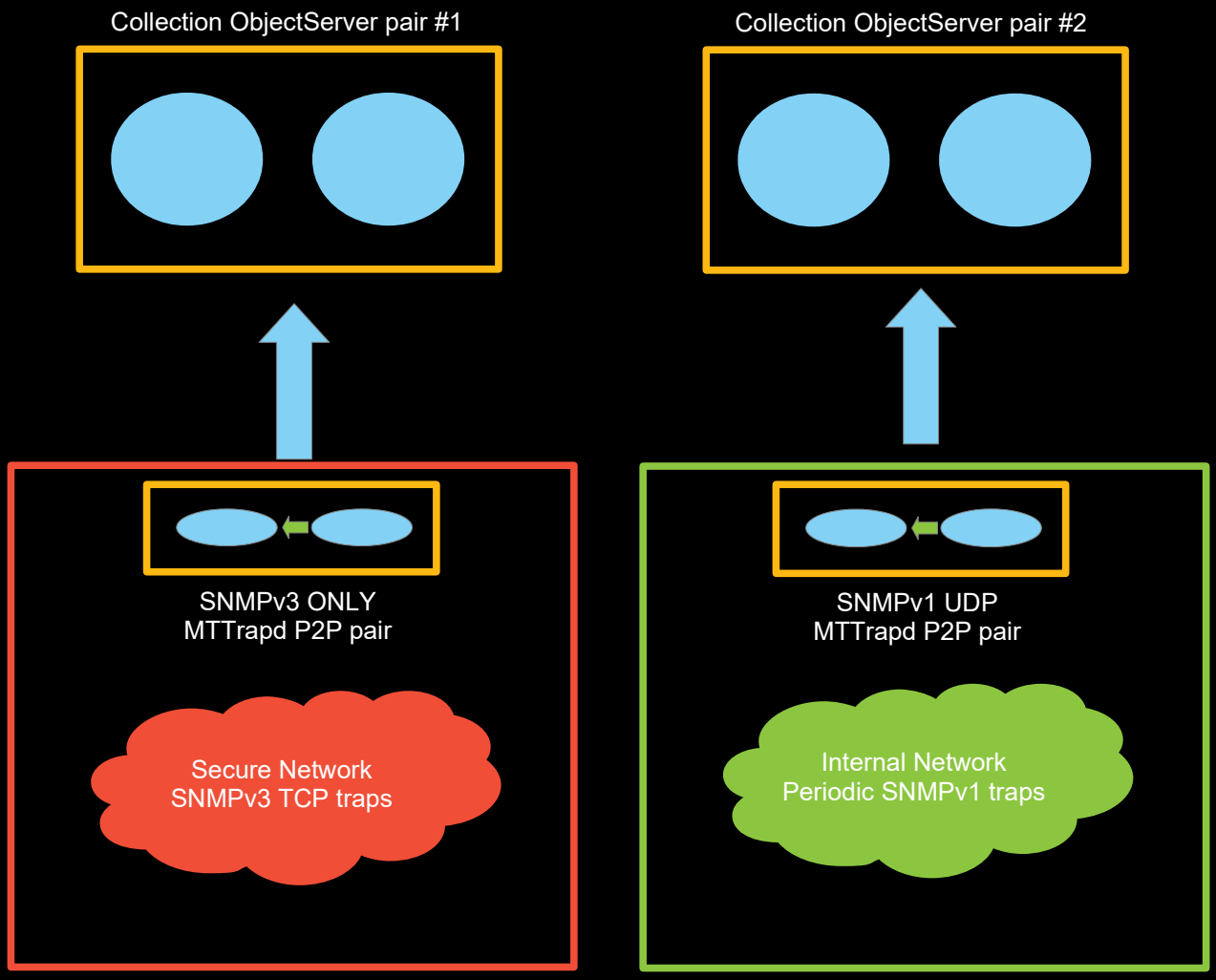
Ways of segregating SNMP data

- Periodic
- Event driven
- Data size
- Number of senders
- Number of traps

Before deciding how to segregate data, it is best to create a test environment and examine the impact of each type of SNMP data on the probe and collection object server.

2.9.5 Example Architecture

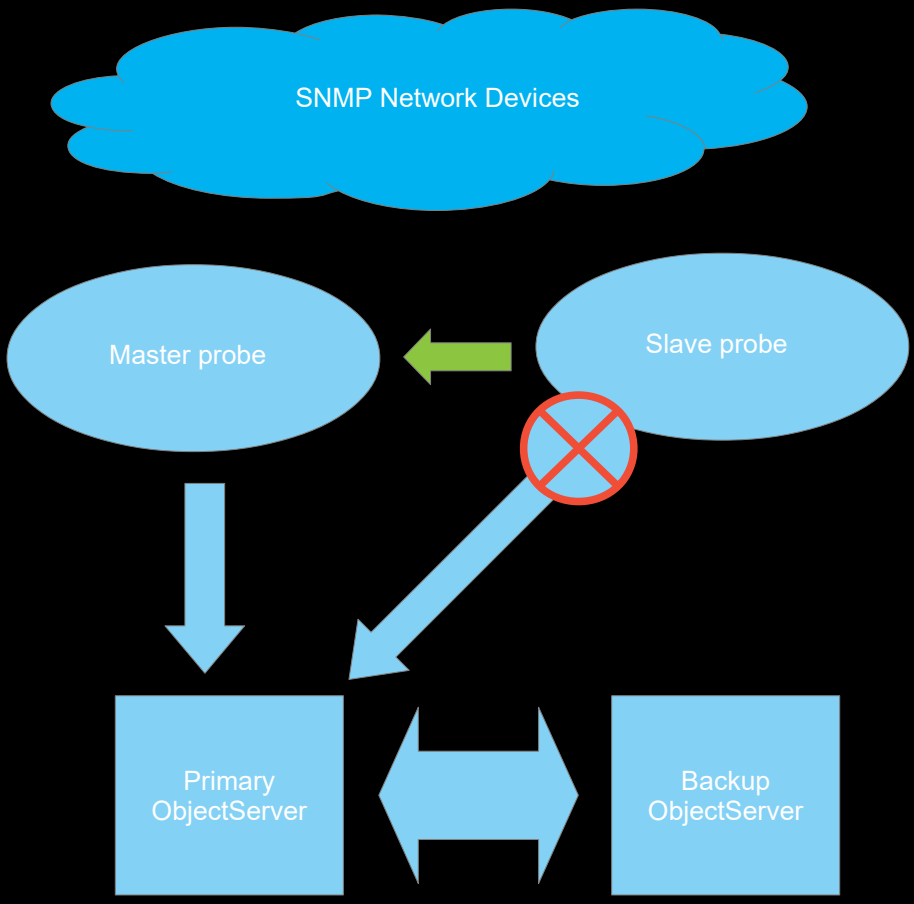
In the example given below, the customer identified that they were required to monitor two distinct networks, with each network sending two distinct types of traps. Due to the nature of the secure network, it was decided to implement a dedicated collection layer object server pair to prevent performance problems at the collection layer, and to allow for custom event handling. A firewall was configured on the two MTTrapd probe servers to prevent unauthorized IP addresses from sending data to the MTTrapd probes port. The internal network was found to be susceptible to SNMP trap flooding, and additional logic was added to the dedicated collection object server pair to prevent event floods from affecting the aggregation layer.



3 Resilience

3.1 Overview

With Peer to Peer failover (P2P), two probes are used with the events from the slave probe being discarded. Should the master probe become unavailable, the slave probe begins to forward its SNMP events to the Object Server. The configuration can be a dual site, or an dual resilient object server pair. The two probes can be configured with identical Server and ServerBackup settings to allow the SNMP events to be sent to the same Object Server, if this is required.



3.2 Failover and Failback

These are the common features that allow the MTTrapd probe to failover to a specified Object Server and then failback to the Primary Object Server;

```
Server                          : 'AGG_P'
ServerBackup                    : 'AGG_B'
NetworkTimeout                  : 30
PollServer                      : 60
```

On the back-up object server the property;

```
BackupObjectServer: TRUE
```

Must be set to allow failback.

The PollServer property must be larger than NetworkTimeout.
i.e.

```
PollServer = 2 * NetworkTimeout
```

3.3 Peer to Peer

When configuring the MTTrapd probe for failover the PeerHost should be different.

Example configuration:
MASTER HOST Properties file on master_host:

```
PeerHost          : 'slave_host'
PeerPort          : 6789
Mode              : 'master'
BeatInterval      : 11
BeatThreshold     : 3
Port              : 162
```

SLAVE HOST Properties file on slave_host:

```
PeerHost          : 'master_host'
PeerPort          : 6789
Mode              : 'slave'
BeatInterval      : 11
BeatThreshold     : 3
Port              : 162
```

3.3.1 BeatThreshold and BeatInterval

The BeatThreshold and BeatInterval are used to manage the timings of the heartbeats from the slave probe to the master probe. The BeatInterval must always be larger than the BeatThreshold, with the BeatThreshold adjusted to allow probe time to respond to heartbeat requests. The probe may not be able to respond to heartbeat requests if it is extremely busy, or CPU starved. Ensure that the Buffering is configured correctly before increasing the BeatThreshold and BeatInterval.
e.g.

```
BeatInterval  : 25
BeatThreshold : 5
```

3.3.2 Single probe server

It is possible to run two MTTrapd probes on the same host in failover mode. In order to do this you would need to use different trap ports

e.g.

MASTER [Primary] HOST Properties file on myhost:

```
PeerHost            : 'slave_host'
PeerPort            : 6789
Mode                : 'master'
BeatInterval        : 25
BeatThreshold       : 5
MessageLevel        : 'debug'
MessageLog          : '/opt/Omnibus/log/mttrapd_master.log'
Port                : 162
```

SLAVE [Secondary] HOST Properties file on myhost:

```
PeerHost            : 'master_host'
PeerPort            : 6789
Mode                : 'slave'
BeatInterval        : 25
BeatInterval        : 5
MessageLevel        : 'debug'
MessageLog          : '/opt/Omnibus/log/mttrapd_slave.log'
Port                : 1620
```

3.4 Multiple probe instances

The common property file settings that need to be set uniquely are.

- Name
- RulesFile
- NHttpd.ListeningPort
- NHttpd.AccessLog

The probe-specific property settings that need to be set uniquely are.

- Port
- ConfPath
- PersistentDir

The RulesFile and ConfPath can be shared between the two probe instances when required.
The second probe run will exit if the same Port or Nhttpd.ListeningPort are used, as the listener cannot be started.
The PersistentDir property points to a directory which is used by the probe instance to cache the ConfPath mttrapd.conf details. It is recommended to never allow MTTrapd probes to share the same PersistentDir. Although the probe will run, there is a risk of problems occurring especially when the ConfPath contains a mttrapd.conf that is being updated, whether or not the details are being shared between probes.

3.4.1 MTTrapd probe instance#1

To create the mttrapd.conf file in UNIX.

```
mkdir $NCHOME/omnibus/var/snmpv3-mttrapd_001
mkdir $NCHOME/omnibus/var/snmpv3-mttrapd_001/conf
vi $NCHOME/omnibus/var/snmpv3-mttrapd_001/conf/mttrapd.conf
################################################################################
# createUser -e engineid username MD5|SHA|SHA256 passphrase DES|AES|AES256 passphrase
################################################################################
# EOF
:wq
```

The probe property settings.

```
###
# Instance name : mttrapd_001
Name                      : 'mttrapd_001'
###
# Rulesfile
RulesFile                 : '$NCHOME/omnibus/probes/linux2x86/mttrapd.rules'
###
# Define the port uniquely - above 1024 for non-root usage
Port                    : 1621
###
# Command port
NHttpd.EnableHTTP               : TRUE
NHttpd.ListeningHostname        : 'localhost'
NHttpd.ListeningPort            : 11621
NHttpd.AccessLog                : "$NCHOME/omnibus/log/mttrapd_001.11621.nhttpd.access.log"
####
# SNMPv3 Conf
ConfPath : '$NCHOME/omnibus/var/snmpv3-mttrapd_001/conf'
PersistentDir : '$NCHOME/omnibus/var/snmpv3-mttrapd_001'
###
#MessageLevel             : 'debug'
MessageLevel            : 'info'
###
# Object Servers
Server                  : 'AGG_P'
ServerBackup            : 'AGG_B'
###
# Best practice
NetworkTimeout          : 15
PollServer              : 60
# Buffering for 20 EPS
Buffering               : 1
BufferSize              : 200
FlushBufferInterval     : 9
# Performance tuning
# Max for 32-bit probe - 65535
# SocketSize                    : 65535
# For 64-bit
# sysctl net.core.rmem_max
# net.core.rmem_max = 212992
SocketSize              : 212992
DisableDetails          : 1
#NoNameResolution       : 1
# Heartbeating
ProbeWatchHeartbeatInterval : 60
Heartbeat               : 60
# EOF
```

3.4.2 MTTrapd probe instance#2

For a second instance, the instance name, mttrapd_001, and port settings are updated to mttrapd_002.
Remember to create the instance name ConfPath and PersistentDir directories, and the mttrapd.conf file.

```
mkdir $NCHOME/omnibus/var/snmpv3-mttrapd_002
mkdir $NCHOME/omnibus/var/snmpv3-mttrapd_002/conf
vi $NCHOME/omnibus/var/snmpv3-mttrapd_002/conf/mttrapd.conf
################################################################################
# createUser -e engineid username MD5|SHA|SHA256 passphrase DES|AES|AES256 passphrase
################################################################################
# EOF
:wq
```

Probe property file settings

```
###
# Instance#2
Name                        : 'mttrapd_002'
###
# Rulesfile
RulesFile                   : '$NCHOME/omnibus/probes/linux2x86/mttrapd.rules'
###
# Define the port uniquely - above 1024 for non-root usage
Port                      : 1622
###
# Command port
NHttpd.EnableHTTP               : TRUE
NHttpd.ListeningHostname        : 'localhost'
NHttpd.ListeningPort            : 11622
NHttpd.AccessLog                : "$NCHOME/omnibus/log/mttrapd_002.11622.nhttpd.access.log"
####
# SNMPv3 Conf
ConfPath : '$NCHOME/omnibus/var/snmpv3-mttrapd_002/conf'
PersistentDir : '$NCHOME/omnibus/var/snmpv3-mttrapd_002'
```

Confirm port availability using netstat on UNIX.

```
netstat -na | grep 1621
netstat -na | grep 11622

netstat -na | grep 1621
netstat -na | grep 11622
```

3.5 Ipv4 and Ipv6

An example of using two probe instances is when Ipv4 and Ipv6 is required.
The MTTrapd probe can only listen on Ipv4 or Ipv6, not both.
The Ipv6 Protocol property settings are TCP6, UPD6, or ALLIPV6.

Example probe property settings, in addition to the multiple instance probe property settings.

```
Probe Instance#1:
Name                    : 'mttrapd_ipv4'
RulesFile               : '$OMNIHOME/probes/linux2x86/mttrapd.rules'
BindAddress             : '127.0.0.1'
Protocol                : 'ALL'
Port                    : 1620

Probe Instance#2:
Name                    : 'mttrapd_ipv6'
RulesFile               : '$OMNIHOME/probes/linux2x86/mttrapd.rules'
BindAddress             : '::1'
Protocol                : 'ALLIPV6'
Port                    : 1620
```

Setting the BindAddress to the Ipv4 or Ipv6 IP address allows the same Port property value to be used. Since the probe starts either an Ipv4 or Ipv6 listener, depending upon the BindAddress value.

4 Testing

4.1 UDP Traps

Sending a single trap to the localhost on Solaris;

```
/usr/sbin/snmp_trapsend -g 6 -s 12 -a ".1.3.6.1.4.1.42.1.1.2 STRING (Testing the trap utility)"
```

Sending a periodic trap;

```
/bin/sh
while true
do
/usr/sbin/snmp_trapsend -g 6 -s 12 -a ".1.3.6.1.4.1.42.1.1.2 STRING (Testing the trap utility)"
sleep 10
done
```

Sending a trap storm;

```
/bin/sh
while true
do
/usr/sbin/snmp_trapsend -g 6 -s 12 -a ".1.3.6.1.4.1.42.1.1.2 STRING (Testing the trap utility)"
done
```

4.2 TCP Traps

The SNMP gateway supports the sending of TCP SNMP traps and so can be used alongside a test object server being fed by a simnet probe to generate large amounts of TCP SNMP traps.

Example configuration;

```
CREATE MAPPING SNMP_MAP
(
        0 = '@Summary',
        1 = '@Severity',
        2 = '@Location',
        3 = '@Node',
        4 = '@AlertGroup'
);

# Start up the reader - connect to the Object Server NCOMS
START READER NCOMS_READER CONNECT TO NCOMS1;
# Start up the writer
START WRITER SNMP_WRITER
(
        TYPE = SNMP,
        REVISION = 1,
        GATEWAY = 'snmp-server',
        PORT = 1620,
        PROTOCOL = 'TCP',
        MAP = SNMP_MAP
);

ADD ROUTE FROM NCOMS_READER TO SNMP_WRITER;
```

4.3 WireShark RAW TCP data

The WireShark program can be used to export an actual trap for replication of specific issues.

http://www.wireshark.org/

To extract the Trap[s]:

- Load snoop/ethereal into the GUI
- Select SNMP packet to export at [+] SNMP
- File-> Export Selected Packet Bytes-> Filename [trap#.bin]

To replay the trap[s]:

- Start the MTTrapd probe

```
$OMNIHOME/probes/nco_p_mttrapd -all -port port
```

- Cat the binary trap file to the probes port using netcat

```
cat trap#.bin | nc localhost port
```

IMPORTANT : For SNMPv3 it is not recommended to replay traps out of sequence or repeatedly, as this can break the SNMPv3 security. If traps need to be replayed more than once, the MTTrapd probe should be restarted before the traps are replayed.

4.4 WireShark SNMPv3 trap inspection

WireShark allows SNMPv3 traps to be inspected through its Graphical User Interface.

Select the trap you would like to inspect, in the trap you can select an item such as the engineid and copy the value using the right menu, or the keys control-shift-V.
e.g.

```
snmp.msgAuthoritativeEngineID == 00:11:22:33:44:55:66:77:88:99:00:11:22:33:44:55:66:77:88:99:00
```

This can be used to set the engineid in the mttrapd.conf file.
e.g.

```
createUser -e 0011223344556677889900112233445566778899000 myusername SHA mySHApassphrase DES myDESpassphrase
```

You can check the contents of the SNMPv3 trap by selecting the trap and opening the right menu:
Protocol preferences → Open Simple Network Management Protocol Preferences
Pop-Up → User Table [Edit]
Pop-Up → +
Engine ID, Username, MD5|SHA1, passphrase, DES|AES, passphrase
[ok]
[ok]

If successful, the GUI will show the contents of the encrypted trap.
If there is a problem, check the SNMPv3 details again.

4.5 SNMP v3 traps and informs

NET-SNMP is a free software package which allows the sending of traps and informs using snmptrap. It is available from these websites:

http://www.net-snmp.org/

Solaris download : http://www.sunfreeware.net/introduction.html

- Start the mttrapd probe from the command line in debug mode when testing the probe

```
$OMNIHOME/probe/nco_p_mttrapd -messagelevel debug -messagelog stdout -all
```

- From another host send the traps, to test send an SNMPv1 trap

SNMPv1 trap;

```
snmptrap -v 1 -c public PROBEHOST "" "" 6 99 ""e
```

The **PROBEHOST** can be set to have the protocol and port set.
e.g. TCP and 1620 on host 192.168.20.18 would be;

```
TCP:192.168.20.18:1620
```

4.5.1 Sending an SNMPv3 INFORM trap

Add the following line to the $OMNIHOME/var/mttrapd.conf file and restart the probe;

```
# INFORM
createUser informuser SHA test1234 AES test1234
```

Use the snmpinform command.

```
snmpinform -M -v3 -u informuser -a SHA -A test1234 -x AES -X test1234 -l authPriv
TCP:0.0.0.0:1620 "" coldStart.0
```

INFORM traps reply to the snmptrap application which allows the user/pass pairs to be tested, once they have been added to the mttrapd.conf file.

4.5.2 Sending an SNMPv3 trap

For normal SNMPv3 traps specify the **ENGINE ID** in the both the snmptrap and mttrapd.conf file;

mttrapd.conf :

```
createUser -e 800000010203040506 jack MD5 jackjill
```

To send the trap use;

```
snmptrap -v 3 -e 0x800000010203040506 -a MD5 -x DES -l noAuthNoPriv -u jack -A "jackjill"
-X "jackjill" TCP:0.0.0.0:1620 "" .1.3.6.1.4.1.2021.251.1
```

4.6 More on SNMPv3 Traps

Here are the common questions that arise when creating the mttrapd.conf file.

Note: The probes mttrapd.conf file uses SHA for SHA1 and AES for AES128.

4.6.1 Special characters in a passphrase

The MTTrapd probe does accept non-alpha numeric characters in the passphrase, however, it is recommended that any complex passphrase string is tested using snmptrap, or some other client to ensure there are not any issues.

ConfPath mttrapd.conf entries:

```
createUser -e 8000000000000000000000001 someuser SHA PASSW0RD@123 AES PASSW0RD@123
createUser -e 8000000000000000000000002 SomeUSER SHA PASSW0RD@123 AES PASSW0RD@123
createUser -e 8000000000000000000000003 Some_USER SHA PASSW0RD@123 AES PASSW0RD@123
```

To send the test traps:

```
snmptrap -M $NETSNMP/share/snmp/mibs -e 8000000000000000000000001 -v3 -u someuser -a
SHA -A PASSW0RD@123  -x AES -X PASSW0RD@123 -l authPriv  TCP:localhost:1620  "" IF-
MIB::linkUp IF-MIB::ifAlias s "alias:123"
$ snmptrap -M $NETSNMP/share/snmp/mibs -e 8000000000000000000000002  -v3 -u SomeUSER
-a SHA -A PASSW0RD@123  -x AES -X PASSW0RD@123 -l authPriv  TCP:localhost:1620  ""
IF-MIB::linkUp IF-MIB::ifAlias s "alias:123"
$ snmptrap -M $NETSNMP/share/snmp/mibs -e 8000000000000000000000003 -v3 -u Some_USER
-a SHA -A PASSW0RD@123  -x AES -X PASSW0RD@123 -l authPriv  TCP:localhost:1620  ""
IF-MIB::linkUp IF-MIB::ifAlias s "alias:123"
```

Example multiple special characters in the passphrase test.

ConfPath mttrapd.conf entry:

```
createUser -e 8000000000000000000000004 SPECIALCHARS SHA "!$%^&*()_+=-?><,.#;~:}{][" AES
"!$%^&*()_+=-?><,.#;~:}{]["
```

To send the test trap:

```
snmptrap -e 8000000000000000000000004 -v3 -u SPECIALCHARS -a SHA -A "!$%^&*()_+=-?
><,.#;~:}{][" -x AES -X "!$%^&*()_+=-?><,.#;~:}{][" -l authPriv TCP:localhost:1620 ""
coldStart.0
```

4.6.2 Maximum characters in a passphrase

The Net-SNMP API supports passphrases up to 64 characters.

ConfPath mttrapd.conf entry:

```
createUser -e 0x80001234567864 maxpassword SHA
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
06464 AES
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
06464
```

To send the test trap:

```
snmptrap -e 0x80001234567864 -v3 -u maxpassword -a SHA -A
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
06464 -x AES -X
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
06464 -l authPriv TCP:localhost:1620 '' 1.3.6.1.6.3.1.1.5.3
```

4.6.3 Extended encryption support

The MTTrapd probe now supports extended encryption types:

```
authtype    MD5, SHA, or SHA256
privtype    DES, AES, AES192 or AES256 [and AES256C with the latest test fix patch]
```

However, the Net-SNMP snmptrap command only supports privtype DES and AES, so cannot be used to test extended privtype encryption types, even though these can be set and used in the mttrapd.conf file.

```
-a PROTOCOL   set authentication protocol (MD5|SHA|SHA-224|SHA-256|SHA-384|SHA-512)
-x PROTOCOL   set privacy protocol (DES|AES)
```

File : mttrapd.conf

```
createUser -e 8000000000000000000000010 SHA256User1234 SHA256 PASSW0RD123567890 AES
PASSW0RD123567890
```

sending the trap:

```
snmptrap -M $NETSNMP/share/snmp/mibs -e 8000000000000000000000010 -v3 -u
SHA256User1234 -a SHA256 -A PASSW0RD123567890 -x AES -X PASSW0RD123567890 -l authPriv
TCP:localhost:1620  "" IF-MIB::linkUp IF-MIB::ifAlias s "alias:123"
```

4.7 SNMP Flooding

The Netcool/SSM agent is excellent at sending vast quantities of traps and can be used to test loading of the MTTrapd probe.

e.g.

```
cd /opt/netcool/ssm/config
vi monitors.cfg
…
#
process attr=averageCpu
process filtertype=name                filter=".*"
process interval=1                     sampletype=delta
process oper=ge                        thresh=0
process actioneventstatus=alwaysReady  actionevent=$psevent
process create
…
:wq

cd /opt/netcool/ssm/bin
./init.ssmagent stop
./init.ssmagent start
./ssmcons
trapdest add <IP ADDRESS>:<PORT> public
trapdest add <IP ADDRESS>:<PORT> public
trapdest add <IP ADDRESS>:<PORT> public
trapdest add <IP ADDRESS>:<PORT> public
```

5 Troubleshooting

5.1 Performance

The performance of the SNMP probe is most affected by name resolution.

5.1.1 Name resolution

The following methods for obtaining hostnames/ip addresses are available to the operating system:

Files [/etc/hosts]
DNS
NIS

If the property 'NoNameResolution' is set to '1' and there still appears to be a performance issue at the probe, check the probe rules files for the commands:

- gethostname
- gethostaddr

As these impede the probes performance with respect to name resolution.

Adding hosts to the /etc/hosts file can improve performance, where DNS is slow.
Use DNS caching when DNS is required and if the /etc/hosts file begins to exceed few hundred lines, as DNS/NIS is a memory resident lookup, which is significantly faster than using files.

5.1.1.1 Rules File Considerations

With the NoNameResolution set to 1, the probe will not use naming resolution, which improves the probes overall performance. However, the rules files may include repetitive host naming lookups, such as looking up the probe servers hostname and IP Address. The way to reduce this load is to use static variables as shown:

```
if(match(%MyHostname,""))
{
        %MyHostname = hostname()
        %MyIPHostname = gethostaddr(%MyHostname)
}
@Manager = %Manager + "@" + %MyIPHostname
```

5.1.2 SNMP INFORMS

SNMP INFORMS require two way communication between the probe and INFORM sender. The MTTrapd probes uses a separate head for TCP/UDP INFORMS, however, the probe still needs to process each INFORM, and it should be expected that INFORMS are at least 50% slower than the equivalent TRAPS.

5.1.3 Object Server

The object server's performance can affect how the probe performs if the object server is continually unresponsive, causing the probe to switch to Store And Forward or event failover to the backup object server. Therefore it is important to check that the object server is able to manage the load from the MTTrapd probe under peak loading, by checking its profiler and trigger statistics.

5.2 Locale problems

When the #_text string does not show the actual locale text, but the #_raw tokens do; This indicates that the probes LANG and LC_ALL settings are not set correctly. **The LANG and LC_ALL environment settings need to be set consistently within the Netcool/OMNIbus environment.** Create a probe environment file to resolve the issue specifically for the MTTrapd probe as shown here.

```
File : $NCHOME/omnibus/probes/linux2x86/nco_p_mttrapd.env

LANG=es_ES.UTF-8
LC_ALL=es_ES.UTF-8
export LANG LC_ALL
echo "LC_ALL=$LC_ALL"
echo "LANG=$LANG"
# EOF


OID1:       .1.3.6.1.4.45.32
1: Test#1 trap : CÃDR
1_raw:      Test#1 trap : CÃDR
1_text:     Test#1 trap : CÃDR
1_hex:      54 65 73 74 23 31 20 74 72 61 70 20 3a 20 43 c3 83 44 52
```

Separate probe instances with different LANG and LC_ALL settings can be used to manage trap senders whose locale settings cannot be catered for, using a single locale setting. In this case, unique users with LANG and LC_ALL settings may be appropriate to use.

5.3 Logging dropped traps

The MTTrapd probe includes a method to log what traps dropped, when the trap queue is full, so as to allow further troubleshooting and diagnosis.

DSALog

Use this property to specify whether the probe logs traps that are lost because the trap queue has become full [1]

DSAPeriod

Use this property to specify the time, in seconds, that traps are logged when the DSALog property has a value of 1 [default is 30 seconds].

5.4 Missing EngineId's

Sometimes the EngineId's of the devices sending the traps is unknown due to legacy issues. SNMPv3 traps need a engineId to be given in the mttrapd.conf file for them to be read by the probe. If the username and password is known, then TSHARK [part of the Wireshark package] can be used to collect the engineId's arriving at the probes port.

For example:

```
tshark -V -i eth0 -d tcp.port==1620,snmp | grep -i msgAuthoritativeEngineID:
```

Will show lines with msgAuthoritativeEngineID: string for all the TCP SNMP traps on port 1620.

```
msgAuthoritativeEngineID: 010203040506
```

These can then be used in the mttrapd.conf file to create valid createUser entries when the user and SNMPv3 trap details are known.

```
 createUser -e 010203040506 trapuser  MD5 md5password DES despassword
```

5.4.1 ConfPath engineid entries

The ConfPath mttrapd.conf file can have multiple entries for the same engineid.

For example.

```
createUser -e 01020304050607080910 authprivtest_noauthnopriv
createUser -e 01020304050607080910 authprivtest_authnopriv SHA DUMMYPASSPHRASE2
createUser -e 01020304050607080910 authprivtest_authpriv SHA DUMMYPASSPHRASE3 AES DUMMYPASSPHRASE3
```

This is not recommended.

The recommendation is to only have one entry for each engineid and to use upper case letters in the ConfPath mttrapd.conf file.

This is because only noAutnoPriv SNMPv3 traps can be used, when there are multiple entires.
Unless the boot parameters are synchronized or unset between the traps senders.

The command line interfaces print_engines output is in upper case for the engineids.

```
engineid:01020304050607080910  boots:1  time:123  host:
engineid:0102030405060708090A  boots:1  time:123  host:
```

Although the probes logging uses lower case engineids.

You can check for duplicate engineid's using the 'tr' command.
For example, a unique sort of the mttrapd.conf file returns different results than a sort.

```
grep -v '^#' mttrapd.conf | grep '\-e' | awk '{print $3}' | tr '[a-z]' 'A-Z]' | sort | wc -l
13

grep -v '^#' mttrapd.conf | grep '\-e' | awk '{print $3}' | tr '[a-z]' 'A-Z]' | sort -u | wc -l
8
```

Note that some entries may include '0x' so the results need to be checked manually as well.

5.5 Preventing trap loss

The use of noAuth/noPriv traps will reduce the risk of trap loss and mttrapd.conf file configuration. Typically these types of SNMPv3 traps are used in an environment where minimal intransit security is required, or when customers have migrated from SNMPv2c to SNMPv3 and security settings are unimportant.
For noAuth/noPriv SNMPv3 traps, only the engineid and username are required to decode the traps body. This detail can be obtained from the SNMPv3 traps header.

Probe property settings:

ConfPath : '$OMNIHOME/var/snmpv3/conf'
PersistentDir : '$OMNIHOME/var/snmpv3'

File : $OMNIHOME/var/snmpv3/conf/mttrapd.conf

```
createUser -e 8000ffffffffffffffffffff01 snmpv3User
createUser -e 8000ffffffffffffffffffff02 snmpv3User
createUser -e 8000ffffffffffffffffffff03 snmpv3User
```

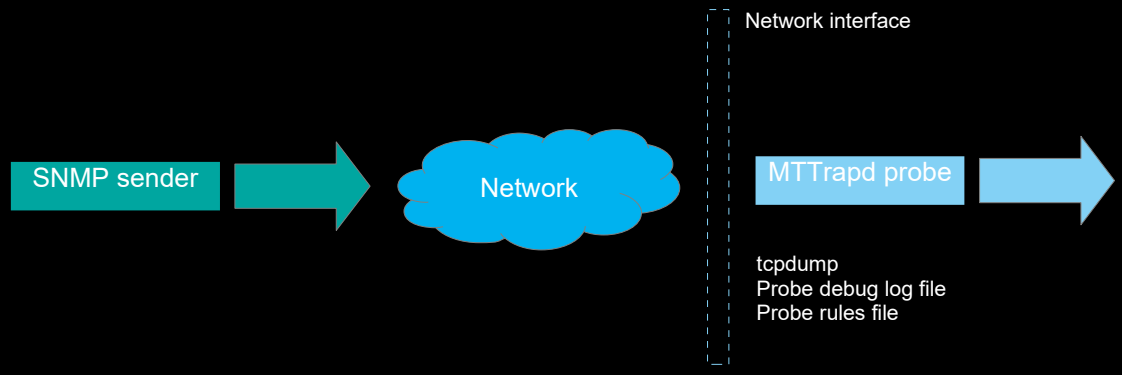
Example trap details:

```
snmptrap -e 8000ffffffffffffffffffff01 -v3 -u snmpv3User -l noauthnoPriv
TCP:localhost:162 "" coldStart.0
snmptrap -e 8000ffffffffffffffffffff02 -v3 -u snmpv3User -l noauthnoPriv
TCP:localhost:162 "" coldStart.0
snmptrap -e 8000ffffffffffffffffffff03 -v3 -u snmpv3User -l noauthnoPriv
TCP:localhost:162 "" coldStart.0
```

```
tshark -s 0 -V -i any port 162 -d tcp.port==162,snmp | egrep -e msgUserName: -e
msgAuthoritativeEngineID: -e msgUserName: -e msgAuthoritativeEngineBoots: -e
msgAuthoritativeEngineTime:
```

5.6 Missing traps

Traps can be lost anywhere within the system, from the trap source to the object server. It is important to identify exactly where the traps are lost within the system.



Most systems allow tcpdump to be installed and can be used to confirm that the traps are arriving at the probe server:

```
tcpdump -s 0 -w /tmp/tcpdump_ems.pcap host snmp_sender_host
```

Use the resulting data to check and confirm that all the traps arrive and have the right details, such as port, protocol, SNMP settings, etc.
The probes debug log will include all the tokens for the traps that the probe is able to read. Traps that cannot be read will be discarded silently. If the probes debug log does not include a trap seen in the tcpdump data, then there are a number of reasons why this can happen. For SNMPv3 there is a separate section.

For SNMPv1 and SNMPv2c traps:

1. Traps must include uptime
2. The correct protocol must be set for the probe [udp/tcp]

If all the traps are seen in the debug log file, then the problem is with event processing rather than trap reception. The probe supports a raw capture mode that allows events to be saved to a file and replayed to the Standard Input probe in a test environment.

If the debug log file cannot be obtained it is possible to log specific probe tokens as required. For example:

log(warn,"SNMPV3Check:" + $PeerIPaddress + ":" + $IPaddress)

Note :

With the latest MTTtrapd probe test fix patch the property EngineInfoProbeWatch can be set to 1 to enable the SNMP_WATCH logging facilities. These allow extra logging for the SNMPv3 traps, which highlight specific issues with the incoming data, when compared to the PersistentDir mttrapd.conf file and the probes SNMPv3 engine cache.

5.7 SNMPv3 traps

SNMPv3 traps have the additional security requirements placed on them. These are detailed within the RFC's regarding SNMPv3. From the MTTrapd probes perspective, SNMPv3 traps must adhere to the RFC requirements as enforced by the NET-SNMP library. Typically, the MTTrapd probe will drop non-compliant traps silently, which can cause administration problems as it makes troubleshooting difficult. The reason why the probe does not perform any analysis of such traps is due to the way the traps are processed, within the NET-SNMP libraries, and the overall requirement to ensure high performance.

Typically traps will be dropped if they are malformed, inconsistent, or time delays past the allowed SNMPv3 tolerance window. For example, if SNMPv3 traps are sent from two or more sources with the same engineID, one or more could be discarded depending upon how these traps present themselves to the probe. It is therefore important to ensure that each element uses a unique engineID to prevent traps from being dropped.

Equally, it is important to administer the network such that only authorised elements are able to send traps to the MTTrapd probe server, to prevent denial of service.

IMPORTANT : The PersistentDir mttrapd.conf file is appended to rather than overwritten. Therefore it should be kept up to date, with unique entries, to prevent security issues. Whilst remembering that the probe will require time to create the PersistentDir mttrapd.conf file from the ConfPath mttrapd.conf file in memory. If the PersistentDir mttrapd.conf file is deleted it forces the probe to rebuild the PersistentDir mttrapd.conf file. The PersistentDir mttrapd.conf file is created when the probe exits, and is held in memory until then.

5.7.1 Using TSHARK

TSHARK can be used ot check trap in real time, and cross checked with the probes debug log file or custom logging during updates to the ConfPath mttrapd.conf file.

SNMPV3 trap headers:

```
msgVersion:
msgID:
msgMaxSize:
msgFlags:
msgSecurityModel:
msgAuthoritativeEngineID:
msgAuthoritativeEngineBoots:
msgAuthoritativeEngineTime:
msgUserName:
msgAuthenticationParameters:
msgPrivacyParameters:
msgData:
```

The main values to check are:

- msgAuthoritativeEngineID:
- msgAuthoritativeEngineBoots:
- msgAuthoritativeEngineTime:
- msgUserName:
- msgVersion:

This TSHARK command listens on eth0 for UDP traps on port 1620:

```
tshark -V -i eth0 -d udp.port==1620,snmp | egrep 'msgAuthoritativeEngineID:|
msgAuthoritativeEngineBoots:|msgAuthoritativeEngineTime:|msgUserName:|msgVersion:'
```

The msgVersion needs to be snmpv3, for SNMPv3 traps.

The msgAuthoritativeEngineID and msgUserName need to exist uniquely in the ConfPath mttrapd,conf file.

The msgAuthoritativeEngineBoots and msgAuthoritativeEngineTime should be increasing for any given msgAuthoritativeEngineID.

If the msgAuthoritativeEngineBoots is being reset, then the probes cached value for EngineBoots needs to be reset, using the new features discussed in the manual. You can restart the probe if necessary, to refresh the cached data.

The msgAuthoritativeEngineTime needs to increase incrementally, for a given msgAuthoritativeEngineID and msgAuthoritativeEngineBoots value. If this value is going backwards in value, then those traps will not be processed by the probes NET-SNMP libraries. This is part of the security features provided with SNMPv3, as packets could be replayed to the probe to create false events. There is some built-in leniency for traps delayed in the network, so not all traps will be discarded, only those with significant time delays (snmpEngineTime +/- 150 seconds).

5.7.2 SNMPv3 related log messages

With TCP traps and informs the MTTrapd probe logs a message to the probes log file, in debug mode.

```
Debug: D-UNK-000-000: Freeing TCP Session response structure
```

With UDP traps and informs the MTTrapd probe logs nothing, unless the engineid and user details are correct. The engineid and user details can be observed using TSHARK or reviewing the PCAP file in WireShark.

When the engineid and user are correctly defined in the probes ConfPath mttrapd.conf file, the following messages are seen for the UDP SNMPv3 traps with AuthPriv set:

Wrong AUTH passphrase:

```
Warning: W-UNK-000-000: SNMP Message (priority=4): Authentication failed for <SNMPv3 User>
Warning: W-UNK-000-000: Dropping SNMP packet because of snmp error, error msg: USM authentication
failure (incorrect password or key) (plaintext scopedPDU header type 00: s/b 30)
```

Wrong AUTH type:

```
Warning: W-UNK-000-000: SNMP Message (priority=4): Authentication failed for <SNMPv3 User>
Warning: W-UNK-000-000: Dropping SNMP packet because of snmp error, error msg: USM authentication
failure (incorrect password or key) (plaintext scopedPDU header type 00: s/b 30)
```

Wrong PRIV passphrase:

```
Warning: W-UNK-000-000: SNMP Message (priority=4): security service 3 error parsing ScopedPDU
```

Wrong PRIV type:

```
Warning: W-UNK-000-000: SNMP Message (priority=4): security service 3 error parsing ScopedPDU
```

Note that the MTTrapd probe will only log AUTH problems, until the AUTH issue is resolved, and afterwards log issues with PRIV.

5.7.3 Additional SNMPv3 security

The MTTrapd probe has two SNMPv3 specific properties to ensure that the probe is kept secure:

- snmpv3ONLY
- snmpv3MinSecurityLevel

Setting the probe to use only SNMPv3 traps at a specific security level, allows other traps to be ignored and prevent denial of service attacks.

For example, the highest security setting is:

```
snmpv3ONLY : 1
snmpv3MinSecurityLevel : 3
```

With snmpv3ONLY set to '1' the probe logs:

```
Dropping the trap since it is v1/v2c trap
```

With snmpv3MinSecurityLevel set to '2' or '3' the probe logs:

```
Dropping V3 traps/informs because it does not match required SNMPv3 security level
```

5.7.4 Multiple engineid entries

Source engineId's are not limited to a single user.
The Net-SNMP library allows multiple engineid's per host provided they use unique usernames.

For example.

```
createUser -e 0102030405060708091O trapuser001 SHA SHApassword001 DES DESpassword001
createUser -e 0102030405060708091O trapuser002 SHA SHApassword002 DES DESpassword002
createUser -e 0102030405060708091O trapuser003 SHA SHApassword003 DES DESpassword003
```

Although this is limited to the security level noAuthnoPriv, due to the boot parameter entries defined by trap sender.

The recommendation is to avoid having multiple entries in the ConfPath mttrapd.conf file for the same engineid's.

This behavour may be useful during passphrase updates, as it allows new and old passphrases to exist in the probes SNMPv3 engine, provided there is a new user as well.

For example.

```
createUser -e 0102030405060708091O oldtrapuser SHA oldpassphrase DES  oldpassphrase
createUser -e 0102030405060708091O newtrapuser SHA newpassphrase DES  newpassphrase
```

5.7.5 Example Secure SNMPv3 configuration

mttrapd.conf:

```
createUser -e 800000020109840311 shauser SHA shapassword DES despassword
```

5.7.5.1 Example trap#1

Trap sent:

```
snmptrap -e 0x800000020109840311 -v3 -u shauser -l noAuthnoPriv UDP:IPADDRESS:1621 0
coldStart.0
```

Working probe properties:
snmpv3ONLY : 1
snmpv3MinSecurityLevel : 1
Log Message: Trap is processed

Blocked probe properties:
snmpv3ONLY : 1
snmpv3MinSecurityLevel : 2

Log message:

```
Warning: W-UNK-000-000: Dropping V3 traps/informs because it does not match required
SNMPv3 security level, configured level = 2, pdu level = 1
```

5.7.5.2 Example trap#2

Trap sent:

```
snmptrap -e 0x800000020109840311 -v3 -u shauser -a SHA -A shapassword -l authNoPriv
UDP:IPADDRESS:1621 0 coldStart.0
```

Working probe properties:
snmpv3ONLY : 1
snmpv3MinSecurityLevel : 2
Log Message: Trap is processed

Blocked probe properties:
snmpv3ONLY : 1
snmpv3MinSecurityLevel : 3

Log message:

```
Warning: W-UNK-000-000: Dropping V3 traps/informs because it does not match required
SNMPv3 security level, configured level = 3, pdu level = 2
```

5.7.5.3 Example trap#3

Trap sent:

```
snmptrap -e 0x800000020109840311 -v3 -u shauser -a SHA -A shapassword -x DES -X
despassword -l authPriv UDP:IPADDRESS:1621  0 coldStart.0
```

Working probe properties:
snmpv3ONLY : 1
snmpv3MinSecurityLevel : 3
Log Message: Trap is processed

5.8 Firewall checks

5.8.1 Checking the object server connection

The probe requires access to the object server using the port specified in the omni.dat file. This port can be blocked by some firewalls, which allow access up to a certain data size. This can be problematic as it is harder to detect, as the object server will respond successfully to a ping.
e.g.

```
nco_ping NCOMS
```

The recommended test is to use a describe of the alerts.status table, as this is the command the probe will run on start-up and will expose issues with the firewall, that can cause the probe to fail on start-up.
e.g.

```
nco_sql -server NCOMS -user root
describe alerts.status;
go
```

5.8.2 Checking the probes port

When the probe is unable to receive trap even though the probe has Protocol set to 'ALL', and the traps are seen when snooping the probes port, there may be an issue with the probe servers firewall or routing.

Try using TCP traps from the same source system to see if further details are logged, either at the source or in the probes debug log file, as UDP traps will be dropped silently.

Traps can be captured and replayed using another system to confirm that the traps themselves are not the cause of the problem.

For Linux, use iptables to check the probe servers firewall settings.
e.g.

```
iptables -L > /tmp/iptables.txt
```

Check the number of interfaces and the routing on the probe server.
e.g.

```
ifconfig -a
netstat -nr
```

If necessary try setting the BindAddress property in the probes property file to fix the probes listening address and check the interfaces ifconfig dropped and received statistics.

6 Useful scripts

6.1 SendSNMPv1HB

```
#! /bin/sh

if [ $# -ne 2 ]
then
        echo "Usage : $0 [host] [port]"
        exit
fi

export counter
counter=1

while true
do
date

snmptrap -v 1 -c public TCP:$1:$2 .1.3.6.1.4.45 localhost 2 0 '0'  .1.3.6.1.4.45.32 s
"Test#${count} trap" &

counter=`expr $counter + 1`
sleep 60
done
#EOF
```

6.2 sendSNMPv2HB

```
#! /bin/sh

if [ $# -ne 2 ]
then
        echo "Usage : $0 [host] [port]"
        exit
fi
export counter
counter=1

while true
do
date
snmptrap -v 2c -c public TCP:${1}:${2} "" NET-SNMP-EXAMPLES-
MIB::netSnmpExampleHeartbeatNotification netSnmpExampleHeartbeatRate i $counter

counter=`expr $counter + 1`
sleep 10
done
#EOF
```

6.3 sendSNMPv3DES_noauthnopriv

```
#! /bin/sh
# mttrapd.conf entry:
# createUser -e 010203040506 trapuser  MD5 md5password DES despassword
#
export count HOST PORT NUMBER
count=1

if [ $# -ne 3 ]
then
        echo "Usage : $0 [host] [port] [number of traps]"
        exit
fi

HOST=$1
PORT=$2
NUMBER=$3

while [ $count -le $NUMBER ]
do
snmptrap -e 0x010203040506 -v3 -u trapuser TCP:${HOST}:${PORT} "" coldStart.0
count=`expr $count + 1`
done
#EOF
```

6.4 sendSNMPv3DES_authpriv

```
#! /bin/sh
# mttrapd.conf entry:
# createUser -e 010203040506 trapuser  MD5 md5password DES despassword
#
export count HOST PORT NUMBER
count=1

if [ $# -ne 3 ]
then
        echo "Usage : $0 [host] [port] [number of traps]"
        exit
fi

HOST=$1
PORT=$2
NUMBER=$3

while [ $count -le $NUMBER ]
do
snmptrap -e 0x010203040506 -v3 -u trapuser -a MD5 -A md5password -x DES -X despassword -l
authPriv TCP:${HOST}:${PORT} "" coldStart.0
count=`expr $count + 1`
done
#EOF
```

7 MIB MANAGER

7.1 Example usage

The files exported by MIB MANAGER include a README.txt that explains how to merge and use the exported rules files.

This is an example merge of some IBM MIBs for NcKL 4.2:

To create the NcKL rules file for the given MIBs:

- IMPORT MIBS into MIB Manager

Check they are all imported as expected using the pop-up and GUI.

Note: Usually it is easiest to remove all the mibs from MIB MANAGER, then import the base mibs, and then import the custom mibs.

- EXPORT MIBS as NcKL 3.0

This export creates a file in the chosen export directory like:
`<date&time>-nckl_3_0`

- Zip up the directory for copying to the probe server:

`<date&time>-nckl_3_0.zip`

Review the README.txt file:

```
MIB Manager NCKL Format Rulesfiles
------------------------------
Instructions For Use:

The rulesfile was designed for use with the NCKL 3.x format rules.
The current IBM recommended and supported rulesfile format
is NCKL. It is highly recommended all users make use of the NCKL
format if at all possible.

To use this NCKL format rulesfile simply place includes for the two
per vendor master files into your snmptrap.rules file.
Those files are named:
<vendor>/<vendor>.m2r.master.include.lookup
<vendor>/<vendor>.m2r.master.include.rules

If there already exists vendor specific NCKL rules for the vendor
then simply include the above two m2r.master files in

<vendor>/<vendor>.master.include.lookup
<vendor>/<vendor>.master.include.rules

and in this case the <vendor>/<vendor>-preclass.snmptrap.lookup
file should be merged into the existing preclass lookup file.
-------------------------------------------------------------------------
```

To install on the new rules files on probe server:

- Unpack the zip file in a temporary directory

e.g.

```
unzip <date&time>-nckl_3_0.zip
cd <date&time>-nckl_3_0
ls -R1 ibm
ibm:
ibm-IBM-3200-MIB_eventBrowserLogin.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventBrowserLogin.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventBrowserLogout.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventBrowserLogout.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventDoorOpen.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventDoorOpen.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventDriveError.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventDriveError.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventFaultPosted.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventFaultPosted.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventLoaderConfigChange.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventLoaderConfigChange.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventLoaderOK.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventLoaderOK.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventLoaderPasswordChange.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventLoaderPasswordChange.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventLoaderRetriesExcessive.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventLoaderRetriesExcessive.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventMailSlotAccessed.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventMailSlotAccessed.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventRequestDriveClean.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventRequestDriveClean.user.include.snmptrap.rules
ibm-IBM-3200-MIB_eventStatusChange.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_eventStatusChange.user.include.snmptrap.rules
ibm-IBM-3200-MIB_ibm3200Event.adv.include.snmptrap.rules
ibm-IBM-3200-MIB_ibm3200Event.user.include.snmptrap.rules
ibm-IBM-3200-MIB.include.snmptrap.lookup
ibm-IBM-3200-MIB.include.snmptrap.rules
ibm-IBM-3200-MIB.sev.snmptrap.lookup
ibm-IBM-ENETDISPATCHER-MIB.adv.include.snmptrap.rules
ibm-IBM-ENETDISPATCHER-MIB.include.snmptrap.lookup
ibm-IBM-ENETDISPATCHER-MIB.include.snmptrap.rules
ibm-IBM-ENETDISPATCHER-MIB.sev.snmptrap.lookup
ibm-IBM-ENETDISPATCHER-MIB.user.include.snmptrap.rules
ibm-IBM2210-MIB.adv.include.snmptrap.rules
ibm-IBM2210-MIB.include.snmptrap.rules
ibm-IBM2210-MIB.sev.snmptrap.lookup
ibm-IBM2210-MIB.user.include.snmptrap.rules
ibm-IBMIROC-MIB_ibmIROCtrapsels.adv.include.snmptrap.rules
ibm-IBMIROC-MIB_ibmIROCtrapsels.user.include.snmptrap.rules
ibm-IBMIROC-MIB_ibmIROCtrapsfr.adv.include.snmptrap.rules
ibm-IBMIROC-MIB_ibmIROCtrapsfr.user.include.snmptrap.rules
ibm-IBMIROC-MIB_ibmIROCtrapssys.adv.include.snmptrap.rules
ibm-IBMIROC-MIB_ibmIROCtrapssys.user.include.snmptrap.rules
ibm-IBMIROC-MIB.include.snmptrap.rules
ibm-IBMIROC-MIB.sev.snmptrap.lookup
ibm-IBMTCPIPMVS-MIB.adv.include.snmptrap.rules
ibm-IBMTCPIPMVS-MIB.include.snmptrap.rules
ibm-IBMTCPIPMVS-MIB.sev.snmptrap.lookup
ibm-IBMTCPIPMVS-MIB.user.include.snmptrap.rules
ibm-IPSECV1-MIB.adv.include.snmptrap.rules
ibm-IPSECV1-MIB.include.snmptrap.rules
ibm-IPSECV1-MIB.sev.snmptrap.lookup
ibm-IPSECV1-MIB.user.include.snmptrap.rules
ibm-L2TV1-MIB.adv.include.snmptrap.rules
ibm-L2TV1-MIB.include.snmptrap.lookup
ibm-L2TV1-MIB.include.snmptrap.rules
ibm-L2TV1-MIB.sev.snmptrap.lookup
ibm-L2TV1-MIB.user.include.snmptrap.rules
ibm-preclass.include.snmptrap.rules
ibm-preclass.snmptrap.lookup
ibm.m2r.master.include.lookup
ibm.m2r.master.include.rules
```

- Interactively copy the files to the vendor specific directory

```
cp -i * /opt/NcKL_42/rules/include-snmptrap/ibm
```

- Check and compare the pre-class files

Do not copy any files that already exist and check the differences:
check differences

```
diff /opt/NcKL_42/rules/include-snmptrap/ibm/ibm-preclass.include.snmptrap.rules ibm-
preclass.include.snmptrap.rules
diff /opt/NcKL_42/rules/include-snmptrap/ibm/ibm-preclass.snmptrap.lookup ibm-preclass.snmptrap.lookup
```

- Add in the master files

```
cd /opt/NcKL_42/rules/include-snmptrap/ibm
vim ibm.master.include.lookup

# NcKL lookups
include "$NC_RULES_HOME/include-snmptrap/ibm/ibm.m2r.master.include.lookup"
#EOF

vim ibm.master.include.rules
# NcKL Includes
include "$NC_RULES_HOME/include-snmptrap/ibm/ibm.m2r.master.include.rules"
#EOF
```

- Check the rules files syntax using nco_p_syntax

```
$OMNIHOME/probes/nco_p_syntax -rulesfile $NC_RULES_HOME/snmptrap.rules -server NCKL4
-messagelevel warn
```

- Make any required modifications

e.g.

```
vi ibm.m2r.master.include.lookup
#include "$NC_RULES_HOME/include-snmptrap/ibm/ibm-preclass.snmptrap.lookup"
```

The rules files can then be used with the working mttrapd probe.

Notes:
You should always perform rules file work after making a backup of the rules files, and preferably check the edits on a test system, before deployment.
Once installed, the MIB Manager, loaded with the vendor MIBs can be used to send test traps to the MTTrapd probe to confirm the rules files process the traps as required.

7.2 The MIBDirs MIBFile and MIBs Property

The MTTrapd probes MIB property settings correspond to the Net-SNMP library for MIBDIRS, MIBFILES and MIBS environment variables. The loaded MIBs are used by the Net-SNMP API, for data interpretation.

The Net-SNMP web site gives the example:

```
snmptranslate -m +CISCO-RHINO-MIB -IR ciscoLS1010ChassisFanLed
.1.3.6.1.4.1.9.5.11.1.1.12
```

From the MTTrapd probe manual:

MIBDirs
Use this property to specify where the probe searches for MIB modules.
This is in the form of a colon-separated list of directories.

The default is $OMNIHOME/common/mibs.
Note: The directories specified using this property must be separated by a colon (:) on UNIX or a semicolon (;) on Windows.

MIBFile
Use this property to specify the name of the MIB file.

The default is $OMNIHOME/probes/arch/mib.txt.
Note: If you are using a rules file generated by the trapd converter, you must set this property to point to an empty file; for example, /dev/null.

MIBs
Use this property to specify which MIB modules the probe loads. Your entry should be in the form of a colon-separated list of modules.

The default is ALL (this instructs the probe to load all modules available in the list of directories specified by the MIBDirs property).

These property settings correspond to the Net-SNMP environment variables.

Net-SNMP documentation:

mibdirs DIRLIST
Specifies a list of directories to search for MIB files.
This operates in the same way as the -M option - see snmpcmd(1) for details.
Note that this value can be overridden by the MIBDIRS environment variable, and the -M option.

mibs MIBLIST
Specifies a list of MIB modules (not files) that should be loaded.
This operates in the same way as the -m option - see snmpcmd(1) for details.
Note that this list can be overridden by the MIBS environment variable, and the -m option.

mibfile FILE
Specifies a (single) MIB file to load, in addition to the list read from the mibs token (or equivalent configuration).
Note that this value can be overridden by the MIBFILES environment variable.

8 The probes SNMPv3 Engine

8.1 SNMPv3 overview

SNMPv3 uses a number of features to ensure the secure transmission of network information.
In the probes configuration file the user can set engineid's against these features:

Security Username: An one character or more string

Authentication method: A fixed choice of MD5, SHA, or SHA256
Authentication passphrase: An 8 character or more string

Privacy method: A fixed choice of DES, AES or AES256
Privacy passphrase: An 8 character or more string

Example ConfPath mttrapd.conf file.

```
#############################################################################
# createUser -e engineid username MD5|SHA|SHA256 passphrase DES|AES|AES256 passphrase

createUser -e 800000000000000000000001 snmpv3boots  SHA SHApassword AES AESpassword

# EOF
```

SNMPv3 trap header for the incoming trap.

```
Simple Network Management Protocol
msgVersion: snmpv3 (3)
msgAuthoritativeEngineID: 800000000000000000000001
msgUserName: snmpv3boots
msgAuthoritativeEngineBoots: 1
msgAuthoritativeEngineTime: 100
```

The probes SNMPv3 engine cache is visible using the probes command line interface and the snmp_action "print_engines" after reading the SNMPv3 trap.

```
engineid:800000000000000000000001  boots:1  time:100  host:
```

The engineid details are captured event token created by the trap:

```
securityEngineID:  0x800000000000000000000001
```

The probe can be configured with all of the settings, and receive traps with noAuthnoPriv, AuthnoPriv, and noAuthnoPriv.

The engineid defines the trap sender, not the the source IP address.

8.1.1 Authentication and Privacy

The SNMPv3 protocol is fully enabled when the security level is set to authPriv.

Both the authentication and privacy settings in the trap are checked and compared with the probes SNMPv3 cache, as are the boots parameters.
The security name is used generally, and needs to be set.

When the traps security level is set to noAuthnoPriv, neither the authentication or privacy are checked by the probe. Additionally the boot parameters are ignored, allowing traps to be sent from different SNMPv3 agents while using the same engineid.

When the traps security level is set to AuthnoPriv, only the authentication is checked by the probe by the probe and the traps are sent in clear text format. Additionally the boot parameters are checked, with a small tolerance of less than 120 seconds for the boottime.

When the traps security level is set to AuthPriv, the authentication is checked by the probe, and the traps are sent encrypted using the privacy protocol settings. Additionally the boot parameters are checked.

Example SNMPv3 trap header details.

```
    msgVersion: snmpv3 (3)

    msgAuthoritativeEngineID: 800000000000000000000001
    msgAuthoritativeEngineBoots: 1
    msgAuthoritativeEngineTime: 123456
    msgUserName: snmpv3authpriv

    msgData: encryptedPDU (1)
```

Related snmptrap options.

security level:
`-l authPriv`

security name:
`-u snmpv3authpriv`

authentication protocol and passphrase:
`-a SHA -A SHApassphrase`

privacy protocol and passphrase:
`-x AES -X AESpassphrase`

security engine ID:
`-e 800000000000000000000001`

8.1.2 SNMPv3 engine

The SNMPv3 engine details consists of the engineid, engine boots, and engine boot time. These timings are incremental unless the SNMPv3 engine is reinitialised. This feature is designed to prevent spoofing and resending of traps by the network. There is a tolerance window of 180 seconds for delays. Traps outside this time are dropped, as are traps with engine boots less than the value stored in the probes SNMPv3 engine cache.

For a given SNMPv3 engine, defined by the engineid.

```
Boot : 1, Boot time : 1
Boot : 1, Boot time : 2
Boot : 1, Boot time : 3
...
Boot : 2, Boot time : 1  ← engine restart
Boot : 2, Boot time : 2
Boot : 2, Boot time : 3
...
Boot : 3, Boot time : 1  ← engine restart
Boot : 3, Boot time : 2
Boot : 3, Boot time : 3
etc.
```

If the SNMPv3 engine is reinitialised, the numbers are reset, and the boot time incremented per second as before.

```
Boot : 1, Boot time : 1
```

This behaviour causes problems for the MTTrapd probe, as it expects the traps with the engineid to continue to always increase.
The previous engine boot and boot time values are stored in the probes SNMPv3 engine cache, which is reset when the probe exits. The values for any given SNMPv3 engineid are defined by the first trap the probe receives that uses the given SNMPv3 engineid. It is therefore important to allocate unique SNMPv3 engineid's to eachSNMPv3 trap sender.

If unique a SNMPv3 engineid's cannot be used per SNMPv3 trap sender, the workaround is have the SNMPv3 senders use engine boot and engine boot time set to 0, for all instances using the SNMPv3 engineid. This is what the Net-SNMP command snmptrap does, unless the engine boot and engine boot time are set usig the command line option.

The MTTrapd probes command line interface supports a per engineid reset feature, which is leveraged by the SNMP_WATCH action tools [mttrapd_create_SnmpActionTools.sql]

The available “snmp_action”'s are.

- print_engines
- get_engine_info
- update_engine
- set_engine_correction

With the SNMPv3 engine reset being a special case for update_engine.

```
$OMNIHOME/bin/nco_http -uri http://localhost:<HTTP-PORT>/probe/common -datatype
application/json -data '{"eventfactory":
[{"snmp_action":"update_engine","engine_id":"<ENGINEID>",
"engine_boot":"","engine_time":"","engine_host":"", "send_probewatch":""}]}' -method post
```

8.1.3 SNMPv3 engine CLI Commands

To use the SNMPv3 engine CLI commands the command line interface needs to configured.

The simplest way is to enable the HTTP interface using the probe property file:

```
# Command port
NHttpd.EnableHTTP            : TRUE
NHttpd.ListeningPort         : 12001
NHttpd.AccessLog             : "$OMNIHOME/log/mttrapd.nhttpd.access.log"
```

Get the engine information for a specific engineid

NCO_HTTP command:

```
$OMNIHOME/bin/nco_http -uri http://localhost:12001/probe/common -datatype
application/json -data '{"eventfactory":
[{"snmp_action":"get_engine_info","engine_id":"00000000000000102030405"}]}' -method post
```

Example Debug: logging messages

```
$snmp_action -> get_engine_info
$engine_id -> 000000000000102030405
>>> Enter mttrapd.bidir.rules. >>>
Received get_engine_info request
Calling get_engine_info
SNMP Message (priority=7): snmp_get_engine_info: Received engineID '000000000000102030405'.
SNMP Message (priority=7): snmp_get_engine_info: Getting engine info of '000000000000102030405' completes.
get_engine_info returns 'EngineID:000000000000102030405  boots:0  time:0  host:'.
EngineID '000000000000102030405' info: EngineID:000000000000102030405  boots:0  time:0  host:
<<< Exit mttrapd.bidir.rules. <<<
```

Dump the probes list of engineid's to a file:

```
$OMNIHOME/bin/nco_http -uri http://localhost:12001/probe/common -datatype
application/json -data '{"eventfactory": [{"snmp_action":"print_engines",
"file_path":"/tmp/engineids.txt"}]}' -method post
```

Example Debug: logging messages:

```
$snmp_action -> print_engines
$file_path -> /tmp/engines.txt
Flushing events to object servers
0 buffered alerts
>>> Enter mttrapd.bidir.rules. >>>
Received print_engines request
Calling print_engines
SNMP Message (priority=7): snmp_print_engines: Received filename '/tmp/engines.txt'.
SNMP Message (priority=7): snmp_print_engines completes.
print_engines operation completes.
RULES>> print_engines request successful
<<< Exit mttrapd.bidir.rules. <<<
```

Example contents:
File : /tmp/engines.txt

```
engineid:000000000000102030405  boots:0  time:0  host:
engineid:80001F8880A862C66948DEFC5D00000000  boots:1  time:0  host:
```

Setting specific parameters for a given engineid:

```
$OMNIHOME/bin/nco_http -uri http://localhost:12001/probe/common -datatype
application/json -data '{"eventfactory":
[{"snmp_action":"update_engine","engine_id":"00000000000000102030405",
"engine_boot":"","engine_time":"", "engine_host":"", "send_probewatch":""}]}' -method
post
```

Example Debug: logging mesages:

```
$snmp_action -> update_engine
$engine_id -> 000000000000000102030405
$engine_boot ->
$engine_time ->
$engine_host ->
$send_probewatch ->
Flushing events to object servers
0 buffered alerts
>>> Enter mttrapd.bidir.rules. >>>
Received update_engine request
Calling update_engine
2019-12-20T07:36:49: Information: I-UNK-000-000: SNMP Message (priority=6):
snmp_update_engine: Received engineID '000000000000000102030405' eboot_update (0)
eboot_time(0) ipaddr ''.
2019-12-20T07:36:49: Information: I-UNK-000-000: SNMP Message (priority=6):
snmp_update_engine: Updating engine '000000000000000102030405' completes.
update_engine operation completes.
RULES>> update_engine request successful
<<< Exit mttrapd.bidir.rules. <<<
```

8.2 SNMPv3 engine commands stand-alone

The MTTrapd probes command line interface supports HTTP and HTTPS for extra security. Using localhost for Nhttpd.ListeningHostname provides better security for stand-alone usage.

The available functions are

- print_engines – prints engineids to the given file
- get_engine_info – logs details to probes log file
- update_engine – logs results to probes log file

8.2.1 HTTP CLI property settings

The MTTrapd probe allows nco_http commands to be sent to the port defined by the Nhttpd property settings.

```
# Command line interface
NHttpd.EnableHTTP       : TRUE
NHttpd.ListeningHostname: '<FQDN>|localhost'
NHttpd.ListeningPort    : 5555
NHttpd.ExpireTimeout    : 15
NHttpd.AccessLog        : "$OMNIHOME/log/mttrapd.access.log"
# Probewatch messages
EngineInfoProbeWatch    : 1
```

8.2.2 Sending SNMPv3 Engine Commands

Print a list of SNMPv3 engine details to a file:

```
$OMNIHOME/bin/nco_http -uri http://localhost:5555/probe/common -datatype
application/json -data '{"eventfactory": [{"snmp_action":"print_engines",
"file_path":"/opt/IBM/tivoli/netcool/omnibus/var/snmpv3.engines.txt"}]}' -method post
```

Output to given file : /opt/IBM/tivoli/netcool/omnibus/var/snmpv3.engines.txt
Example:

```
engineid:01020304050607080912  boots:1  time:0  host:
engineid:01020304050607080910  boots:1  time:0  host:
engineid:01020304050607080911  boots:1  time:0  host:
```

Get SNMPv3 engine details:

```
$OMNIHOME/bin/nco_http -uri http://localhost:5555/probe/common -datatype application/json
-data '{"eventfactory":
[{"snmp_action":"get_engine_info","engine_id":"01020304050607080912"}]}' -method post
```

Logs to probes log file:

```
Debug: D-UNK-000-000: SNMP Message (priority=7): snmp_get_engine_info: Received engineID
'01020304050607080912'.
Debug: D-UNK-000-000: SNMP Message (priority=7): snmp_get_engine_info: Getting engine info of
'01020304050607080912' completes.
Debug: D-UNK-000-000: get_engine_info returns 'EngineID:01020304050607080912  boots:1  time:0  host:'.
Debug: D-UNK-000-000: EngineID '01020304050607080912' info: EngineID:01020304050607080912  boots:1  time:0
host:
```

Set SNMPv3 engine details:

```
$OMNIHOME/bin/nco_http -uri http://localhost:5555/probe/common -datatype application/json
-data '{"eventfactory":
[{"snmp_action":"update_engine","engine_id":"01020304050607080911",
"engine_boot":"","engine_time":"","$engine_host":"localhost", "send_probewatch":""}]}'
-method
```

Logs to probes logfile:

```
Debug: D-UNK-000-000: Received update_engine request
Debug: D-UNK-000-000: Calling update_engine
Information: I-UNK-000-000: SNMP Message (priority=6): snmp_update_engine: Received
engineID '01020304050607080911' eboot_update (123) eboot_time(4567890) ipaddr ''.
Information: I-UNK-000-000: SNMP Message (priority=6): snmp_update_engine: Updating
engine '01020304050607080911' completes.
Debug: D-UNK-000-000: update_engine operation completes.
```

Entry updated when checked:

```
engineid:01020304050607080911  boots:123  time:4567890  host:
```

8.3 SNMP_WATCH - SNMPv3 engine facilities

8.3.1 Configuration Overview

Please refer to the product manuals for complete descriptions of how to enable process control and the new SNMPv3 engine features. The following is provided as an overview of the required actions.

In order to use the new facilities the following needs to be done:

Enable the property setting:

```
EngineInfoProbeWatch    : 1
```

Enable the probes command line interface:

```
# Command port
NHttpd.EnableHTTP           : TRUE
NHttpd.ListeningPort        : 12001
NHttpd.AccessLog            : "$OMNIHOME/log/mttrapd.nhttpd.access.log"
```

Load the SQL file into the object server[s] where the external action is to run:

```
cat mttrapd_create_SnmpActionTools.sql | nco_sql -server AGG_P -user root -password ''
```

Ensure that external actions are configured in the object server[s] where the trigger and procedure is installed:

File : $NCHOME/omnibus/etc/AGG_P.props

```
PA.Name: 'NCO_PA'
PA.Password: 'CIEGAKHGBFGBBIGEDNBJCABFGEBCBKGE'
PA.Username: 'root'
```

Ensure the object server[s] are run under process control:

File : $NCHOME/omnibus/etc/NCO_PA.conf

```
Command '$OMNIHOME/bin/nco_objserv -name MTTRAPD -pa NCO_PA' run as 'nrv81'
```

Ensure that the PERL commands and script are installed on the object server[s] running the external action:

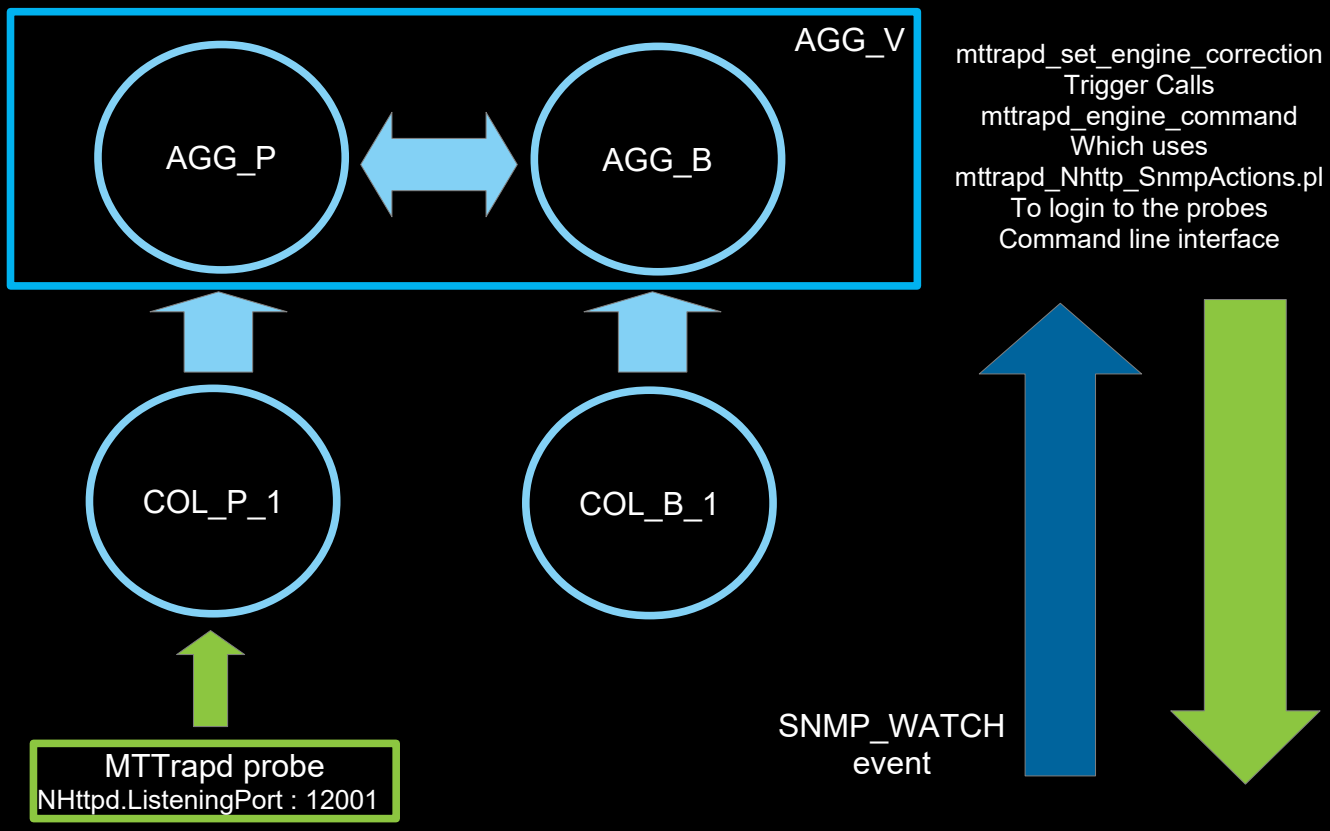
```
$OMNIHOME/probes/<platform>/mttrapd_Nhttp_SnmpActions.pl
```

Ensure the MTTrapd probes rules files to use the SNMPv3 engine integration include files:

```
include "mttrapd.bidir.rules"
include "mttrapd.snmpwatch.rules"
```

8.3.2 SNMP_WATCH Design Overview

Standard multitier configuration
In the standard configuration, the Aggregation layer is configured to run the SNMP_WATCH trigger, although the Collection layer object servers can be used instead, when required. The object servers running the SNMP_WATCH trigger need the mttrapd_Nhttp_SnmpActions.pl installed in the $NCHOME/omnibus/probes/<platform> directory.



8.4 Managing large SNMPv3 ConfPath mttrapd.conf files

The number of rows in the ConfPath mttrapd.conf file depends on the number of SNMPv3 trap sources the probe is expected to manage.

SNMP INFORM's do not use engineid's.

8.4.1 MTTrapd probe initial start-up time

The MTTrapd probe reads the ConfPath mttrapd.conf file on probe start-up. If the PersistentDir mttrapd.conf file does not exist, the probe takes longer to construct the SNMPv3 user engine cache.

8.4.2 MTTrapd probe exit times

With large ConfPath mttrapd.conf files the MTTrapd probe takes a significant amount of time to exit. This is due to the writing of data to the PersistentDir mttrapd.conf file when exiting. Using a hard kill command [kill -9] may corrupt the PersistentDir mttrapd.conf, resulting in significant probe downtime.
Ensure that only soft kills [kill -15] are used on the probe process, and that process automation does not kill the probe process unexpectedly due to a rogue timeout.

Example process control property file setting.

```
File: '$NCHOME/omnibus/etc/nco_pa.props'

# Set RetryTime > 10 seconds
# Set high RogueTimeout setting for large mttrapd.conf files
RetryTime: 30
RogueTimeout: 300
KillProcessGroup: TRUE
# End of File
```

8.4.3 Configuring PersistentDir mttrapd.conf file start-up behaviour

The MTTrapd probe manual discusses the two probe property settings UsmUserBase and ReuseEngineBoots.

For a probe configuration that reduces the probes ready to receive traps time, it is best to work from the PersistentDir mttrapd.conf file. However, these settings prevent updates to the SNMPv3 user cache, and are useful only to reduce trap reception times, where the ConfPath mttrapd.conf is not being updated.

```
UsmUserBase      : 1
ReuseEngineBoots : 1
```

Example logging for 10,000 createUser rows.

```
Information: I-UNK-000-000: SNMP Message (priority=6): Realtime Update Config's
effective interval to check config file status is set to (1) mins.
[10 seconds later]
Information: I-UNK-000-000: SNMP Message (priority=6): Startup Config Parser thread
is running now
[30 seconds]
Information: I-UNK-000-000: SNMP Message (priority=6): Realtime Config Timer thread
is running now
```

9 New features

This chapter outlines some of the new MTTrapd probe features available in the latest probe release.

9.1 Heartbeating

9.1.1 Heartbeat Property

The MTrapd probe could be disconnected from the target system if the connection between them becomes unavailable. The Heartbeat property periodically checks that the connection to the target Object Server, if it detects that the connection is unavailable, it shuts down.

9.1.2 ProbeWatchHeartbeatInterval property

The ProbeWatchHeartbeatInterval property is used by the Return On Investment self monitoring for the probes. It can be set to an interval in seconds:
e.g.

```
ProbeWatchHeartbeatInterval : 60
```

The Return On Investment self monitoring extension is part of the Netcool/OMNIbus extensions, and can be found in the directory:

```
$NCHOME/omnibus/extensions/roi/

Omnibus_TDW_Reports_ROI.zip
probemanagement.sql
probestats.sql
probewatch.include
```

The ProbeWatchHeartbeatInterval property populates the %OplStats variables, which can be used outside of the ROI extension to log specific performance indicators.
e.g.

```
if ( int(%OplStatsNumberEvents) > 0 )
{
# Log OplStats every 100 events
 if ( ((int(%OplStatsNumberEvents)/100)*100) == int(%OplStatsNumberEvents) )
 {
# Calculate the Average usec timings
  $average_usec = int((real(%OplStatsRulesFileTimeSec) +
(real(%OplStatsRulesFileTimeUsec)/1000000) /  real(%OplStatsNumberEvents)) * 1000000)
# Log required details
  log(info,"%OplStatsRulesFileTime = " + %OplStatsRulesFileTimeSec + "." +
%OplStatsRulesFileTimeUsec)
  log(info,"%OplStatsNumberEvents  = " + %OplStatsNumberEvents)
  log(info,"Average USEC timing    = " + $average_usec + " usec.")
 }
}
```

9.2 Host Naming facilities

The host lookups have always been an issue with the MTTrapd probe. A new feature to manage this performance affecting facility is the internal hostname hash table. These features are enabled when the NoNameResolution property is set to 0. Note that these new features have their limitations, and in general, it is best to disable the naming resolution and use the probes rules files logic instead.

NoNameResolution : 0

When enabled, the probe writes resolved host names and discarded host names to flat files in the $NCHOME/omnibus/var directory.

Related properties:

- ActiveHostnameDuration
- RefreshHostnameInterval
- HostnameTableSize

When the host name table has reached its maximum size the probe will stop parsing the active list at the list entry where it detects that the table is full.
The probe will not perform any instantaneous host name resolution for any traps received from new hosts and the IP Address will not be stored in the hash hosts table. The event tokens will use numeric IP Addresses only.

Reading from the IP host flat file
When the probe starts up, it reads from the stored hosts in the file named:
`$OMNIHOME/var/instance_active_iphost.list`
Where *instance* identifies the probe instance uniquely.

9.3 ConfPath mttrapd.conf encryption

New property file settings:

```
# "AES" or "AES_FIPS".
# ConfigCryptoAlgo          : ''
# The key file used in encrypting and decrypting the mttrapd.conf.
#  Key file must be generated from nco_keygen command.
# ConfigCryptoKeyFile       : ''
# Control crypto config processing: 0 - disable; 1 - enable.
# EnableCryptoConfig        : 0
```

Example creation of the MTTrapd probe property settings:

Create encryption key file:

```
$NCHOME/omnibus/bin/nco_keygen -o $NCHOME/etc/security/keys/mttrapd.conf.keyfile
```

Update MTTrapd probe property file:

```
# SNMPv3 Conf
ConfPath                 : '$OMNIHOME/var/snmpv3-encrypted/conf'
PersistentDir            : '$OMNIHOME/var/snmpv3-encrypted'
# Conf encryption
ConfigCryptoAlgo          : 'AES'
ConfigCryptoKeyFile       : '$NCHOME/etc/security/keys/mttrapd.conf.keyfile'
EnableCryptoConfig        : 1
```

Rename the original file:

e.g.

```
mv $OMNIHOME/var/snmpv3-encrypted/conf/mttrapd.conf \
$OMNIHOME/var/snmpv3-encrypted/conf/mttrapd.conf.plaintext
```

Example file contents:

File : $OMNIHOME/var/snmpv3-encrypted/conf/mttrapd.conf.plaintext

```
createUser -e 01020304050607080910 netcoolTrap SHA SHA128_Password AES AES128_Password
createUser -e 01020304050607080911 netcoolTrap SHA SHA128_Password AES AES128_Password
createUser -e 01020304050607080912 netcoolTrap SHA SHA128_Password AES AES128_Password
createUser -e 01020304050607080913 netcoolTrap SHA SHA128_Password AES AES128_Password
createUser -e 01020304050607080914 netcoolTrap SHA SHA128_Password AES AES128_Password
createUser -e 01020304050607080915 netcoolTrap SHA SHA128_Password AES AES128_Password
#EOF
```

Encrypt the mttrapd.conf.plaintext file to the default file name, mttrapd.conf:

```
$OMNIHOME/bin/nco_aes_crypt -k $NCHOME/etc/security/keys/mttrapd.conf.keyfile \
-f $OMNIHOME/var/snmpv3-encrypted/conf/mttrapd.conf.plaintext \
-o $OMNIHOME/var/snmpv3-encrypted/conf/mttrapd.conf
```

9.4 SNMP_WATCH - Unknown EngineId

There is a new feature to allow the MTTrapd probe to notify the object server of SNMPv3 trap|inform senders with unknown engineid's. Unknown senders are those not identified in the probes ConfPath mttrapd.conf file. This feature gives the option to create an automated update to the ConfPath mttrapd.conf file after a suitable security validation. The SNMP_WATCH message does not provide any Auth or Priv details. To simplify the discussion, only traps are mentioned, although both traps and informs are meant. Note that the feature makes use of the SNMPv3 engine integration, which should be applied beforehand.

Related fix.

```
APAR - IJ41114
MTTrapd probe : Detect traps with Unknown engineid's to allow user to update the
ConfPath mttrapd.conf file
```

A warning 'SNMP Message' is displayed in the mttrapd probes log file whenever there is an incoming SNMPv3 trap with an unknown engineid/user name.

```
Warning: W-UNK-000-000: SNMP Message (priority=4): SNMP Trap/Inform was dropped due
to Unknown User/Engine ID: User ID=SomeUser,Engine ID=80000001234567890123,IP
Address=192.168.2.1,Protocol=udp,Port=1620
```

To use the new feature set the probe property EngineInfoProbeWatch on:

```
EngineInfoProbeWatch : 1
```

If EngineInfoProbeWatch is off, the probe will log SNMP Message warning:

```
Warning: W-UNK-000-000: SNMP Message (priority=4): notify_unk_id: probe watch sending
is not enabled.
```

The EngineInfoProbeWatch feature makes use of the ExtendedAttr field in alerts.status table to store useful SNMPv3 details:

- unknown engine id
- unknown/associated user name
- IP address of sender
- IP port
- protocol

The details are stored in the ExtendedAttr field as:

```
engine_id="80000001234567890123";
user_id="SomeUser";
ip_addr="192.168.2.1":
port="1620";
protocol="udp";
detected_time="1659417283"
```

The EngineInfoProbeWatch feature makes use of the existing fields SNMPAction and SNMPError introduced to handle engine correction (mttrapd_create_SnmpActionTools.sql), to indicate this is another action by the probe related to SNMP.

The EngineInfoProbeWatch SNMPAction code is 2, and SNMPError stores a related error message.

The SNMPWatch events for the dropped traps with the unknown engineid are sent to the object server via the probe rules file. These events can be used to trigger actions to add the engineid and user to the ConfPath mttrapd.conf file, based upon the systems security requirements.

9.4.1 Object Server SQL functions

Object server SQL functions can be used to query and set name value pairs

Example usage.

1) Extract all unknown engine id, user id and port received and populate it in a table

```
select nvp_get(ExtendedAttr, 'engine_id') as engine_id,
nvp_get(ExtendedAttr, 'user_id') as user_id,
nvp_get(ExtendedAttr, 'port') as port
from status where SNMPAction=2
```

Result:

```
engine_id user_id port
80000001234567890123 SomeUser 1620
```

2) Update user_id and ip_addr based on engine_id received:

```
update status
set ExtendedAttr = nvp_set(ExtendedAttr, 'user_id', 'AnotherUser'),
ExtendedAttr = nvp_set(ExtendedAttr,'ip_addr','192.168.100.101')
where
lower((nvp_get(ExtendedAttr, 'engine_id')) = '80000001234567890123')
```

3) Check for existence of engine_id before perform an operation :

```
select SNMPError
where nvp_exists(ExtendedAttr, "<some_engine_id>")
```

4) List information about dropped traps sorted by detected time :

```
select nvp_get(ExtendedAttr, 'engine_id') as engine_id,
nvp_get(ExtendedAttr, 'user_id') as user_id,
nvp_get(ExtendedAttr, 'ip_addr') as ip_addr,
nvp_get(ExtendedAttr, 'port') as port,
nvp_get(ExtendedAttr, 'protocol') as protocol,
nvp_get(ExtendedAttr, 'detected_time') as detected_time
from status where SNMPAction=2 order by detected_time
```

9.4.2 SNMP_WATCH Event Checking

The mttrapd.snmpwatch.rules is updated to handle the module 'SNMPUSM_UNKID', and creates an event for every unknown engineid. The event is not cleared and expected to be managed manually. Adding an ExpireTime in the probes rules file is useful for automatically clearing unmanaged events.

The SNMP_WATCH ProbeWatch fields are.

```
SNMPAction=2
SNMPError=SNMP Trap/Inform was dropped due to Unknown User/Engine ID ...
```

The Unknown EngineID events can be selected using the SNMPAction field.

```
select Identifier, SNMPError, AlertKey, ExpireTime, Severity, Summary, Tally from
alerts.status where SNMPAction = 2;
go
```

9.5 SNMP_WATCH – SNMPv3 Engine

The MTTrapd probe includes a feature to allow the probes SNMPv3 cache to be cleared for a specific engine id. This feature uses mttrapd.snmpwatch.rules and mttrapd.bidir.rules, along with the mttrapd_set_engine_correction, which calls the procedure mttrapd_engine_command to resolve issues automatically using the MTTrapd probes command line interface.

Sequence of events

- SNMPv3 trap triggers a SNMP_WATCH ProbeWatch event
- The rules file logic creates a unique event
- The SNMP_WATCH event triggers the procedure mttrapd_engine_command as SNMPAction is set to 1
- The engine id correction perl script is called $OMNIHOME/probes/linux2x86/mttrapd_Nhttp_SnmpActions.pl to resolve the issue by resetting the probes SNMPv3 engine cache for the given engine id
- The MTTrapd probes SNMPv3 engine cache is reset for the given engine id

The main problem with this sequence of events is that relies on the various parts working as expected. If one of the parts does not work, there is a large build up of events in Object Server as the $time token is added to the events Identifier. This is required, as the mttrapd_engine_command trigger is an insert trigger, and the inserted ProbeWatch event does not use ExpireTime, and is only cleared by the successfully completed action.

```
@Identifier = @Manager + " " + @AlertGroup + " " + $engine_id + " " + $time
```

The quick fix is to add a reinsert trigger to the object server manage the reinserted SNMP_WATCH events and remove $time from the Identifier.

```
create or replace trigger mttrapd_set_engine_correction_reinsert
group mttrapd_triggers
priority 20
comment 'Mttrapd trigger for engine correction'
after reinsert on alerts.status
for each row
begin
    if (new.SNMPAction = 1)
    then
        execute jinsert( new.Serial, %user.user_id, getdate(), 'Executing engine
correction on Error:' + new.SNMPError + ' engine_info:' + new.ExtendedAttr);
        execute mttrapd_engine_command(new.URL, 'set_engine_correction', new.SNMPError,
new.ExtendedAttr, new.Identifier);
    end if;
end;
go
```

Update the Identifier settings in the mttrapd.snmpwatch.rules where $time is being used, for SNMPAction=1 ProbeWatch messages.

```
###
#  Creates an event per $time
#  @Identifier = @Manager + " " + @AlertGroup + " " + $engine_id + " " + $time

# Requires mttrapd_set_engine_correction_reinsert trigger to work
#
   @Identifier = @Manager + " " + @AlertGroup + " " + $engine_id
```

With the original SNMP_WATCH event needing to be removed to allow the action to occur, following any issue.

9.5.1 High event volume systems

Ideally the mttrapd_engine_command procedure should only be called once, to prevent repeated SNMPv3 engine resets. The updated SNMP_WATCH triggers use a new table to store the requests for SNMPv3 engine resets. This prevents repeated calls, and introduces a batching facility to limit the number of requests made within a given period. These enhancements make the feature tunable for high event volume systems.

The SNMPv3 engineid reset requests are placed in a table, rather than sent directly to the probe.

```
create table queues.snmpv3_engine_correction persistent
(
        Identifier varchar(255) primary key,
        SNMPError varchar(128),
        URL varchar(128),
        ExtendedAttr varchar(255),
        Timestamp utc
);
```

The temporal trigger nc_snmpv3_engine_correction reads the queues.snmpv3_engine_correction periodically, default 11 seconds.
The number of requests sent per period, is limited by a counter, default 100.

```
    if (connections_count < 100) then
      set connections_count = connections_count + 1;
      execute mttrapd_engine_command(URL, 'set_engine_correction', SNMPError,
ExtendedAttr, Identifier);
      delete  from queues.snmpv3_engine_correction via correction.Identifier;
    else
      cancel;
    end if;
```

The procedure mttrapd_engine_command uses the perl script provided by the MTTrapd probe, and is configured for linux2x86 by default.

```
executable '$OMNIHOME/probes/linux2x86/mttrapd_Nhttp_SnmpActions.pl' host 'localhost'
```

10Appendix

This section of the Support's guide to the MTTrapd probe goes into more detail and provides additional detail to topics already covered.

10.1 MTTrapd probe property file

Example best practice settings for receiving SNMPv3 traps and INFORMs.

```
Server                  : 'AGG_P'
ServerBackup            : 'AGG_B'
# Best practice
NetworkTimeout          : 15
PollServer              : 60
# Buffering - 20 events per second
Buffering               : 1
BufferSize              : 200
FlushBufferInterval     : 9
# Performance tuning
# For 64-bit
# sysctl net.core.rmem_max
# net.core.rmem_max = 212992
SocketSize              : 212992
DisableDetails          : 1
#NoNameResolution       : 1
# Heartbeating
ProbeWatchHeartbeatInterval : 60
Heartbeat               : 60
# SNMPv3 Conf
ConfPath                : '$NCHOME/omnibus/var/snmpv3/conf'
PersistentDir           : '$NCHOME/omnibus/var/snmpv3'
#
Port                    : 1620
Protocol                : 'ALL'
# MIBs
MIBDirs                 : '$NCHOME/omnibus/var/mibmanager/mibs'
MIBFile                 : '$NCHOME/omnibus/probes/linux2x86/mib.txt'
MIBs                    : 'ALL'
# SNMP_WATCH details
EngineInfoProbeWatch    : 1
#
# Command line interface Command line interface and SNMPWATCH tools
#
NHttpd.EnableHTTP       : TRUE
NHttpd.ListeningHostname: 'localhost'
NHttpd.ListeningPort    : 10101
NHttpd.AccessLog        : "$NCHOME/omnibus/log/mttrapd.10101.nhttpd.access.log"
```

The ConfPath mttrapd.conf file must already exist for the probe to run.

```
mkdir $NCHOME/omnibus/var/snmpv3
mkdir $NCHOME/omnibus/var/snmpv3/conf
vi $NCHOME/omnibus/var/snmpv3/conf/mttrapd.conf
################################################################################
# createUser -e engineid username MD5|SHA|SHA256 passphrase DES|AES|AES192|AES256 passphrase
################################################################################
:wq
```

10.2 Sending SNMPv3 INFORMS

File : $NCHOME/omnibus/var/snmpv3/conf/mttrapd.conf

```
###
# Test INFORM user
createUser snmpv3informauthpriv SHA SHApassphrase AES AESpassphrase
```

Start the probe:

```
$NCHOME/omnibus/probes/nco_p_mttrapd -messagelog stdout -messagelevel debug
```

Check the PersistentDir mttrapd.conf file.

```
grep snmpv3informauthpriv $NCHOME/omnibus/var/snmpv3/mttrapd.conf

usmUser 1 3 0x80001f88808340ba7c507efe6600000000 "snmpv3informauthpriv"
"snmpv3informauthpriv" NULL .1.3.6.1.6.3.10.1.1.3
0x58d7d53f127752a4dd27699538ffb0a70dcc8710 .1.3.6.1.6.3.10.1.2.4
0x70c0a0f0225a9891ec81c165adf4d5ad ""
```

To send an SNMP INFORM using the snmpinform command.

Start a new Bourne-shell.

```
sh
```

For AuthPriv.

```
snmpinform -M /usr/share/snmp/mibs -v3 -n netsnmp \
-u snmpv3informauthpriv \
-a SHA -A SHApassphrase -x AES -X AESpassphrase \
-l authPriv \
TCP:0.0.0.0:1620 \
"" coldStart.0
```

For noAuthnoPriv.

```
snmpinform -M /usr/share/snmp/mibs -v3 -n netsnmp \
-u snmpv3informauthpriv \
-l noauthnoPriv \
TCP:0.0.0.0:1620 \
"" coldStart.0
```

The MTTrapd probe tokens include the securityName for the inform.

```
securityName:      snmpv3informauthpriv
```

10.3 Sending SNMPv3 traps

File : $NCHOME/omnibus/var/snmpv3/conf/mttrapd.conf

```
###
# Test trap user
createUser -e 800000000000011111 snmpv3authpriv SHA SHApassphrase AES AESpassphrase
```

Start the probe:

```
$NCHOME/omnibus/probes/nco_p_mttrapd -messagelog stdout -messagelevel debug
```

Check the PersistentDir mttrapd.conf file.

```
grep snmpv3authpriv $NCHOME/omnibus/var/snmpv3/mttrapd.conf

usmUser 1 3 0x800000000000011111 "snmpv3authpriv" "snmpv3authpriv" NULL .
1.3.6.1.6.3.10.1.1.3 0x5eec677abfd59e4c1d41d6f923d5177c6b934d20 .1.3.6.1.6.3.10.1.2.4
0x0304649f23eef2cb65cf317166bf87e3 ""
```

To send an SNMP trap using the snmptrap command.

Start a new Bourne-shell.

```
sh
```

For AuthPriv.

```
snmptrap -M /usr/share/snmp/mibs -v3 -n netsnmp \
-e  800000000000011111 \
-u snmpv3authpriv \
-a SHA -A SHApassphrase -x AES -X AESpassphrase \
-l authPriv \
TCP:0.0.0.0:1620 \
"" coldStart.0
```

For noAuthnoPriv.

```
snmptrap -M /usr/share/snmp/mibs -v3 -n netsnmp \
-e  800000000000011111 \
-u snmpv3authpriv \
-l noauthnoPriv \
TCP:0.0.0.0:1620 \
"" coldStart.0
```

The MTTrapd probe tokens include the securityName and securityEngineID for the trap.

```
securityName:      snmpv3authpriv
securityEngineID:  0x800000000000011111
```

10.4 Same SNMPv3 engineid behaviour

The noAuthnoPriv security level does not check the boots parameters, boots and boottime.

ConfPath mttrapd.conf setting.

```
# Multiple engineid entries - Requires unique users and security level noAuthnoPriv
createUser -e 0102030405060708091O trapuser001 SHA256 SHApassword001 AES256 AESpassword001
createUser -e 0102030405060708091O trapuser002 SHA256 SHApassword002 AES256 AESpassword002
createUser -e 0102030405060708091O trapuser003 SHA256 SHApassword003 AES256 AESpassword003
```

The authPriv security level rejects traps with the same engineid and different boot parameters, accepting only the traps with boot parameters within the higher range.

```
snmptrap -e 0102030405060708091O -Z 1,123 -v3 -u trapuser001 -a SHA256 -A SHApassword001
-x AES256 -X AESpassword001 -l authPriv TCP:${HOST}:${PORT} "" coldStart.0
snmptrap -e 0102030405060708091O -Z 2,1234 -v3 -u trapuser002 -a SHA256 -A SHApassword002
-x AES256 -X AESpassword002 -l authPriv TCP:${HOST}:${PORT} "" coldStart.0
snmptrap -e 0102030405060708091O -Z 3,1235 -v3 -u trapuser003 -a SHA256 -A SHApassword003
-x AES256 -X AESpassword003 -l authPriv TCP:${HOST}:${PORT} "" coldStart.0
```

The authnoPriv security level rejects traps with the same engineid and different boot parameters, accepting only the traps with boot parameters within the higher range.

```
snmptrap -e 0102030405060708091O -Z 1,123 -v3 -u trapuser001 -a SHA256 -A SHApassword001
-x AES256 -X AESpassword001 -l authnoPriv TCP:${HOST}:${PORT} "" coldStart.0
snmptrap -e 0102030405060708091O -Z 2,1234 -v3 -u trapuser002 -a SHA256 -A SHApassword002
-x AES256 -X AESpassword002 -l authnoPriv TCP:${HOST}:${PORT} "" coldStart.0
snmptrap -e 0102030405060708091O -Z 3,1235 -v3 -u trapuser003 -a SHA256 -A SHApassword003
-x AES256 -X AESpassword003 -l authnoPriv TCP:${HOST}:${PORT} "" coldStart.0
```

The noAuthnoPriv security level accepts traps with the same engineid and different boot parameters, provided the trap user is unique.

```
snmptrap -e 0102030405060708091O -Z 1,123 -v3 -u trapuser001 -a SHA256 -A SHApassword001
-x AES256 -X AESpassword001 -l noauthnoPriv TCP:${HOST}:${PORT} "" coldStart.0
snmptrap -e 0102030405060708091O -Z 2,1234 -v3 -u trapuser002 -a SHA256 -A SHApassword002
-x AES256 -X AESpassword002 -l noauthnoPriv TCP:${HOST}:${PORT} "" coldStart.0
snmptrap -e 0102030405060708091O -Z 3,1235 -v3 -u trapuser003 -a SHA256 -A SHApassword003
-x AES256 -X AESpassword003 -l noauthnoPriv TCP:${HOST}:${PORT} "" coldStart.0
count=`expr $count + 1`
```