Tivoli Netcool Supports Guide to Probe Event Replication

by

Jim Hutchinson

Document release: 2.1

Table of Contents

1 Introduction

1.1 Overview

In order to test complex systems, develop rules file code and replicate production issues involving probes, it is necessary to understand the available methods available to recreate events. This document covers the standard methods used for event replication and for the loading of object servers using probes.

The document covers the following probe types;

- Standard Input probe [rules file testing for all probes]
- SimNet probe [Object Server loading]
- Socket based probes [replaying TCP stream capture etc.]
- Tivoli EIF probe [posteifmsg]
- MTTrapd probe [TCP/UDP/PACKET DATA]

2 General event replication

2.1 Standard Input [stdin] probe

The standard input probe replaces the old Generic probe and can be used in a similar way to replay RAW capture data produced by all probes when the RawCaptureFile property is set to ‘1’.

For replaying RAW capture data the timestamp [RawCaptureTimeStamp] can be used to reflect the actual timestamp originally used by the probe;

```
@FirstOccurrence = $RawCaptureTimeStamp
@LastOccurrence  = @FirstOccurrence
```

RAW capture data can be generated manually by following the syntax and providing tokens that the probe rules file being designed for.
e.g.

```
Node = “node1”
Summary = "Machine has gone offline"
Severity = "4"


Node = “node1”
Summary = "Machine has gone online"
Severity = "1"
```

Normally the probe for which the rules file is being developed is used to capture the raw capture file, and this file is used to used to develop rules files. Such raw capture files can be pruned of their ‘Status_’ tokens, as these are generally not required.
e.g.

```
grep -v Status_ mttrapd.raw > mttrapd.cap
```

2.2 Simnet Probe

The simnet probe can be used to load an object server with events for testing the object server, gateways and event list performance.

Column	Description	Example#1	Example#2	Example#3
1	Node name (@Node)	Link100	Node200	Host300
2	Type of event 0 – Link up/down 1 – Node up/down 2 – Disk space 3 – Port error	0	1	2
3	Probability (0-100)	5	25	100

Here is a sample script to create a large simnet definition file;

```
#! /bin/ksh
export SIMNDETDEF MAXEVENTS
/usr/ucb/echo -n "How many nodes? (100) : "
read NODES
if [ -z "$NODES" ]
then
MAXEVENTS=100
else
        if [ $NODES -gt 0 ]
        then
                MAXEVENTS=$NODES
        else
                MAXEVENTS=100
        fi
fi
SIMNDETDEF=/tmp/simnet_${MAXEVENTS}.def
if [ -f $SIMNDETDEF ]
then
        rm  $SIMNDETDEF
fi
count=0
while [ $count -lt $MAXEVENTS ]
do
echo "mya${count}-myb${count} 0 1" >> ${SIMNDETDEF}
echo "myhost${count} 1 1"          >> ${SIMNDETDEF}
echo "myserver${count} 2 3"        >> ${SIMNDETDEF}
echo "myport${count} 3 5"          >> ${SIMNDETDEF}
count=`expr $count + 1`
done
ls -l ${SIMNDETDEF}
#EOF
```

2.3 Socket based probes

Socket based probes can be fed data using TCP sockets from freeware clients, such as netcat and perl. However, such clients cannot replicate devices that interact with probes or that manage multiple ports, as a service.

netcat (nc) is a freeware application (http://netcat.sourceforge.net/) that allows probes to connect to a port and receive a data stream.

e.g.

Xterm#1:
```
cat probe_stream_capture | nc -l -p 5678
```

Xterm#2:
```
nco_p_probe -host hostname -port 5678
```

For probes that require an active connection it is possible to use telnet (listening probes)

e.g.

Xterm#1:
```
nco_p_socket -host hostname -port 3456
```

Xterm#2:
```
telnet hostname 3456
<enter data manually>
```

netcat (nc) can also be used for active connections (listening probes).

e.g.

Xterm#1:
```
cat probe_stream_capture | nc localhost 5678
```

Xterm#2:
```
nco_p_probe -host hostname -port 5678
```

3 Event replication for specific probes

3.1 Tivoli EIF probe

The Tivoli EIF probe has a number of event simulators available, with the most commonly used being the post<x>msg clients. These are available with the Tivoli TEC software [postz|emsg]. The currently supported binary is available with Tivoli EIF SDK and is called posteifmsg.

To create a EIF sender script two files are required;

- send2myhost – script that calls the client application, e.g. posteifmsg
- myhost.conf – configuration file that defines the host and port to connect to plus other parameters

Example script and configuration file:-

```
vi send2myhost
#! /bin/sh
while true
do
posteifmsg -f myhost.conf severity=HARMLESS hostname=myhost physical_node=physical_host
host_type=VCS adapter_host=adapter_h' origin=myhost ERRORED_TEST TEST_NEWLINE &3600
msg='\n\r
sleep 1
done
#EOF
:wq

vi myhost.conf
ServerLocation=myhost.mydomain.com
ServerPort=9995
#EOF
:wq
```

3.2 MTTrapd probe

The MTTrapd (SNMP) probe is the most commonly used probe and there are a number of methods available to send traps and replicate production environments.

3.2.1 UDP traps on Solaris

Solaris comes with the snmp_trapsend client for sending UDP traps.

Sending a single trap to the localhost;

```
/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 periodic traps to the localhost;

```
vi sendsnmp2localhost
/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
#EOF
:wq
```

3.2.2 SNMP Gateway

The SNMP gateway supports the sending of TCP SNMP traps and can be used alongside a test object server fed by a simnet or other probe allowing the generation of a large number of TCP SNMP traps.

Example SNMP gateway 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;
```

3.2.3 MIB MANAGER

You can use the MIB MANAGER to send traps.
This usually done after generating rules files and running them against an MTTrapd probe.
e.g.

```
$OMNIHOME/probes/nco_p_mttrapd -port 1620 -rulesfile /tmp/mibmanager/mttrapd.rules
-messagelevel debug -messagelog stdout
```

Then with the MIB MANAGER loaded with the MIBs used to generate the rules file send traps to check the rules file usage.

Procedure

To generate an SNMP trap in the MIB MANAGER [from the manual]:

Go to the OID Tree tab.
In the OID Tree tab, from the View drop-down list, select one of the following view options and a related navigation tree displays for the view you select.

- All
- Traps/Notifications

Within the navigation tree, click a trap.

In the MIB Manager, click Trap icon !Trap.
In the Destination Address field, specify the host name or IP address of a computer running an SNMP Trapd service.
In the Destination Port field, specify the UDP port that the SNMP Trapd service is listening on.
In the Community field, specify a valid string for the Trapd manager.
Depending on the type of trap that you are generating, a Variable Bindings section might be visible in the Generate an SNMP Trap window. The available fields in the Variable Bindings section also depend on the type of trap being generated.

Point to the Variable Bindings field names for details of the field value syntax. You can specify multiple values, which are separated by commas. These values are randomly chosen to build the trap.
In the Number of Traps field, specify the number of traps that should be generated.

If you specify a large number, traps are generated in rapid succession and can cause an alarm storm at the probe. For values greater than one (> 1), and for values that are specified as semicolon separated lists, MIB Manager randomly selects from the specified lists and generates traps with the arbitrary values presented as varbinds.

Click Execute to generate the trap.

3.2.4 Wireshark

Wireshark, formally called Ethereal, can be used to extract one or more traps for use with netcat.

Method to extract an SNMP Trap|INFORM:

- Load the snoop/tcpdump file into the Wireshark GUI
- Select the trap to be exported/saved
- Select SNMP packet header in the trap detail window
- File -> Export selected packet byes → filename [e.g. packet1.bin]

To use the exported packet1.bin data file, start the mttrapd probe on a port and netcat the file to the probes port:
e.g.
In terminal#1:

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

In terminal#2:

```
cat packet1.bin | nc -w 2 localhost port
```

Note: SNMPv3 is a secure method for sending traps. You cannot send the same trap twice in the lifetime of the probe. To test sending SNMPv3 traps using this method you must restart the probe before sending the same trap again.

3.2.5 SNMPTRAP : SNMP v3 traps

The snmptrap client is available free from the NET-SNMP website or other vendor specific sites;

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

Download : http://sunfreeware.com

SNMPv3 security is added to the probe using the mttrapd.conf file as specified in the probes property file;
e.g.
Add the following line to the $OMNIHOME/var/mttrapd.conf file;

```
createUser jack MD5 jackjill
```

Start the mttrapd probe in debug mode sending its output to the terminal;

```
nco_p_mttrapd -all -messagelevel debug -messagelog stdout
```

From another host send a SNMPv1 trap to check behaviour;

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

To send a SNMPv3 INFORM use;

```
snmptrap -v 3 -C i -a MD5 -x DES  -l noAuthNoPriv -u jack -A "jackjill" -X "jackjill"
hostname "" .1.3.6.1.4.1.2021.251.1
```

INFORMs reply to the snmptrap application allowing user/password pairs to be tested, once they have been added to the mttrapd.conf file.

For SNMP traps, specify the ENGINE ID in the both the snmptrap and mttrapd.conf file;
mttrapd.conf :

```
createUser -e 800000020109840301 jack MD5 jackjill
```

To send an SNMP trap use:

```
snmptrap -v 3 -e 800000020109840301 -a MD5 -x DES -l noAuthNoPriv -u jack -A
"jackjill" -X "jackjill" hostname "" .1.3.6.1.4.1.2021.251.1
```

By default sendsnmp sends UDP traps, to send TCP traps use the hostname syntax '**TCP:hostname:port**' to send TCP traps to the specific port.

e.g.

```
snmptrap -v 3 -e 800000020109840301 -a MD5 -x DES -l noAuthNoPriv -u jack -A
"jackjill" -X "jackjill" TCP:hostname:port "" .1.3.6.1.4.1.2021.251.1
```

4 Useful Scripts

4.1 Sending stream capture events using posteifmsg

File create_postmsg

```
#! /bin/sh
if [ $# -ne 1 ]
then
 echo "Usage : `basename $0` [streamcapture file]"
 exit
fi
cat $1 | \
gawk -F\; '{ for (i=1;i<=NF;++i)\
if($i!=""){\
if($i!=""){\
if(i==1){printf "posteifmsg -f EIF.conf "} else
if($i=="END"){printf "TESTING TEST_%s\n",NR}\
else { printf "%s ",$i }
}\
}\
}'
echo
#EOF
```

4.2 Sending SNMPv1 traps

File : sendSNMPv1_problem

```
#! /bin/sh
export count
count=1

if [ $# -ne 3 ]
then
        echo "Usage : $0 [host] [port] [number of traps]"
        exit
fi
while [ $count -le $3 ]
do
snmptrap -v 1 -c public TCP:$1:$2 "" "$count" 2 123 ""e
count=`expr $count + 1`
done
#EOF
```

File : sendSNMPv1_resolution

```
#! /bin/sh
export count
count=1

if [ $# -ne 3 ]
then
        echo "Usage : $0 [host] [port] [number of traps]"
        exit
fi
while [ $count -le $3 ]
do
snmptrap -v 1 -c public TCP:$1:$2 "" "$count" 3 123 ""e
count=`expr $count + 1`
done
#EOF
```

4.3 Sending SNMPv3 traps

File : sendSNMPv3_MD5DES

```
#! /bin/sh
#
# createUser -e 800000020109840322 myuser MD5 apassword123 DES apassword123
#
export count HOST PORT
count=1

if [ $# -ne 3 ]
then
        echo "Usage : $0 [host] [port] [number]"
        exit
fi
HOST=$1
PORT=$2


while [ $count -le $3 ]
do
snmptrap -v 3 -e 0x800000020109840322 -a MD5 -x DES -l noAuthNoPriv -u myuser -A
"apassword" -X "apassword" TCP:$HOST:$PORT "" .1.3.6.1.4.1.2021.251.1

count=`expr $count + 1`
done
#EOF
```

File : sendSNMPv3DES_authpriv_linkdown

```
#! /bin/sh
#
# createUser -e 0102030405 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 $NUMBER ]
do
snmptrap -e 010203040506070809 -v3 -u trapuser -a MD5 -A md5password123 -x DES -X
despassword123 -l authPriv TCP:${HOST}:${PORT} "${count}" IF-MIB::linkDown IF-
MIB::ifAlias s "alias:$count"
count=`expr $count + 1`
done
#EOF
```

File : sendSNMPv3DES_authpriv_linkup

```
#! /bin/sh
#
# createUser -e 010203040506070809 trapuser  MD5 md5password123 DES despassword123
#
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 $NUMBER ]
do
snmptrap -e 010203040506070809 -v3 -u trapuser -a MD5 -A md5password123 -x DES -X
despassword123 -l authPriv TCP:${HOST}:${PORT} "" IF-MIB::linkUp IF-MIB::ifAlias s
"alias:$count"
count=`expr $count + 1`
done
#EOF
```

4.4 Open TCP Port Sending data

To mimic a server sending data, which might be useful for the socket probe, you can replay the stream capture file using netcat in a while loop.

File : runport5678.sh

```
#! /bin/sh

while true
do
cat start_end_event.str | nc -l -p 5678
date
done

#EOF
```

4.5 Open Socket

An open socket is useful to checking if the probe is connecting and sending data to the server.

File : open_socket.pl

```
#! /bin/perl -w
# Simple perl script to create an open socket
# on a given local host/port
#
# server using IO::Socket
#----------------------
use strict;
use IO::Socket;
use warnings;

&usage() unless @ARGV;

my @args;
my $my_host = "localhost";
my $my_port = "7777";

# Parse arguments
while (@ARGV) {
    for (shift(@ARGV)) {
        ($_ eq '-h' || $_ eq '--host') && do { $my_host =shift(@ARGV); last; };
        ($_ eq '-p' || $_ eq '--port') && do { $my_port = shift(@ARGV); last; };
        ($_ eq '--') && do { push(@args, @ARGV); undef @ARGV; last; };
        ($_ =~ m/^-.+/) && do { print "Unknown option: $_\n"; &usage(); };
        push(@args, $_);
    }
}

print "Creating open port on " , $my_host , " ", $my_port, "\n";

sub usage {
    $0 =~ m|[^/]+$|;
    print "Usage: $& [-h|--host hostname] [-p|--port port_number]\n";
    print "Creates an open socket on the local host\n";
    print "\n";
    exit 1;
}

# Create socket using my_host my_port
my $sock = new IO::Socket::INET(
LocalHost => $my_host,
LocalPort => $my_port,
Proto => 'tcp',
Listen => SOMAXCONN,
Reuse => 1);

$sock or die "no socket :$!";
$sock->autoflush(1);

# Loop forever
while (1)
{
}
#EOF
```

4.6 Message bus probe : Using CURL

File : send_curl_xml.sh

```
#! /bin/sh

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

HOST=$1
PORT=$2
export HOST PORT

curl -v -H "Content-Type: text/xml" -d '<?xml version="1.0" encoding="UTF-8"?
><tns:netcoolEvent type="delete"
xmlns:tns="http://item.tivoli.ibm.com/omnibus/netcool"><tns:netcoolField
name="ServerSerial" type="unsigned">198</tns:netcoolField><tns:netcoolField
name="ServerName" type="string">AGG_P</tns:netcoolField></tns:netcoolEvent>' -X POST
http://${HOST}:${PORT}

#EOF
```

File : send_curl_json.sh

```
#! /bin/sh

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

HOST=$1
PORT=$2
export HOST PORT

curl -v -H "Content-Type: application/json" -d '{"test":"Message"}' -X POST http://$
{HOST}:${PORT}

#EOF
```

File : send_curl_ssl_json

```
#! /bin/sh

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

HOST=$1
PORT=$2
export HOST PORT

if [ -f  ca.cert ]
then

curl --cacert ./ca.cert -v -H "Content-Type: application/json" -d '{"test":"Message
text sent via SSL"}' -X POST https://${HOST}:${PORT}

else
echo “HTTPS requires a valid CA certificate :  ca.cert”
fi

#EOF
```

File : send_curl_prometheus.sh

```
#! /bin/sh

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

HOST=$1
PORT=$2
export HOST PORT

curl -v -H "Content-Type: application/json" -d
'{"timestamp":1551884340000000,"timespan":60000000,"alert":
{"severity":1,"editUrl":"https://cloud-
api:443/#/alerts/122","scope":null,"name":"TEST","description":"Processor Load is too
high","id":123},"event":{"id":123456,"url":"https://cloud-
api:443/#/events/notifications/l:000800/00007/details"},"state":"ACTIVE","resolved":f
alse,"entities":[{"entity":"host.mac = '00:00:00:00:00:00'","metricValues":
[{"metric":"load.average.percpu.5m","aggregation":"avg","groupAggregation":"avg","val
ue":1.0000000}],"additionalInfo":
[{"metric":"host.hostName","value":"myhost.mycompany.com"}]}],"condition":"avg(avg(lo
ad.average.percpu.5m)) > 1","source":"Sysdig Cloud"}' -X POST http://${HOST}:$
{PORT}/probe/webhook/prometheus

#EOF
```