High Volume SNMP Trap Processing

Related Topics

- SNMP Traps with Cisco and Other devices
- opEvents - Centralized Logging Solution
- opEvents - Syslog Handling - Adding a New Vendor
- opEvents - Syslog Handling - Adding a New Format

Table of Contents

Purpose

Provide a SNMP trap handling solution that can scale to 300 traps per second.

Overview

This solution leverages snmptrapd to initially pull the trap off the wire, apply access control, translate, then forward it to rsyslog. rsyslog then puts the translated trap in a log file to be processed by opEvents. opEvents then applies filtering, parsing and actions as appropriate.

```
SNMP Trap Processing - Line Diagram

snmptrapd--> rsyslog--> /var/log/nmis/snmptrap.log --> opEvents --> Blacklist --> EventParserRules -->
SnmpTrapParserPlugin.pm
```

Deployment Steps

Step #1 - Configure snmptrapd to forward traps to syslog

RHEL/Centos - Edit /etc/sysconfig/snmptrapd

Below is an example of configuring snmptrapd to send traps to rsyslog. The '-Ls' flag tells snmptrapd to send logging output to syslog. Using '-Ls2' specifies that snmptrapd will send it with the local2 facility value. The facility value is what rsyslog keys on for routing decisions. Please review the snmptrapd and snmpcmd man pages.

/etc/sysconfig/snmptrapd

```
OPTIONS="-n --OQ Ls2 -p /var/run/snmptrapd.pid -m ALL -M /usr/local/nmis8/mibs/traps"
```

Debian - Edit /etc/default/snmptrapd

Below is an example of configuring snmptrapd to send traps to rsyslog. The '-Ls' flag tells snmptrapd to send logging output to syslog. Using '-Ls2' specifies that snmptrapd will send it with the local2 facility value. The facility value is what rsyslog keys on for routing decisions. Please review the snmptrapd and snmpcmd man pages.

/etc/sysconfig/snmptrapd

```
 TRAPDOPTS="-n --OQ Ls2 -p /var/run/snmptrapd.pid -m ALL -M /usr/local/nmis8/mibs/traps"
/etc/systemd/system/snmptrapd.service.d/override.conf
-n -OQ -Ls2 -p /var/run/snmptrapd.pid -m ALL -M /usr/local/nmis9/mibs/traps
[Service]
ExecStart=
ExecStart=/usr/sbin/snmptrapd -n -OQ -Ls2 -p /var/run/snmptrapd.pid -m ALL -M /usr/local/nmis9/mibs/traps
```

Verify /etc/snmp/snmptrapd.conf

Verify there is not reference to a traphandle in /etc/snmp/snmptrapd.conf

Step #2 - Configure rsyslog to route traps into a specified log file

We need the traps to be placed into a specified log file that opEvents will process. The following example states that all messages with a facility of local2 will be placed in the /usr/local/nmis8/logs/snmptrap.log file. Please review the rsyslog.conf man page.

/etc/rsyslog.d/nmis.conf

```
local2.*                /usr/local/nmis8/logs/snmptrap.log
```

Most likely we will not want these messages to also go to /var/log/messages. We can edit /etc/rsyslog.conf to prevent this from happening. An example for facility local2 follows; notice the pwd

' statement.

/etc/rsyslog.conf

```
 *.info;mail.none;authpriv.none;cron.none;local2.none                /var/log/messages
```

Step #3 - Configure opEvents to process SNMP trap log file using a plugin

Modify opCommon.nmis

We need to tell opEvents to process the newly created snmptrap.log file. This is done in /usr/local/omk/conf/opCommon.nmis. Be careful with this file; in reality it is a perl hash, any syntax error will render the OMK server dead. After modifying this file check it for syntax errors with the following command 'perl -c /usr/local/omk/conf/opCommon.nmis'. If you are not scared you should be 🙂

Something like the following example needs to be added to the opevents section of opCommon.nmis.

/usr/local/omk/conf/opCommon.nmis

```
'opevents_logs' => {
 'snmptraps' => [
   '<nmis_logs>/snmptrap.log'
  ],
```

In the case of opEvents 3, it needs to be adapted to json format.

```
     "opevents_logs" : {
        "snmptraps" : [
          "<nmis9_logs>/snmptrap.log"
        ],
```

Modify EventParserRules.nmis

EventParserRules.nmis is where parsing generally occurs. In this case we are anticipating some complex maneuvers; so we are going to tell EventParserRules to send this to an opEvents plugin where complexity is better dealt with. Remember all that big bad syntax talk? Same applies here.

Something like the following example needs to be added to EventParserRules.nmis

/usr/local/omk/conf/EventParserRules.nmis

```
%hash = (
        'snmptraps' => {
                1 => {
                        IF => 1,
                        THEN => "plugin(snmpTrap)"
                },
        },
```

Install SNMP trap parser plugin

Install an opEvents parser plugin such as: snmpTrap.pm . This perl module will be placed in /usr/local/omk/conf/parser_plugins.

The plugin is not always needed. Traps can be processed using the event handler nmis traplog, but the plugin can parse more complex snmp traps.

Step #4 - Restart associated daemons

Restart the following daemons:

- rsyslog
- snmptrapd
- opeventsd

Step #5 - Verify

- Use tcpdump to observe snmptraps being recieved by the server
- Use the ps command to ensure snmptrapd, rsyslog, omkd, and opeventsd are running with the proper options
- Tail /usr/local/nmis/logs/snmptraps.log file
- Tail /usr/local/omk/log/opEvents.log
- Via the GUI; check opEvents views-> raw logs
- Via the GUI; check opEvents views -> events

Alternative Solution

A plugin is not always needed for snmp trap processing. The plugin should be necessary just when we need to process really complex traps.

Using the built in traplog parser, we would modify the Step 3 for the following:

Step #3 - Configure opEvents to process SNMP trap log file using a built in parser

Modify opCommon.nmis (abi3)/opCommon.json (abi4)

We need to tell opEvents to process the newly created snmptrap.log file. This is done in /usr/local/omk/conf/opCommon.nmis. Be careful with this file; in reality it is a perl hash, any syntax error will render the OMK server dead. After modifying this file check it for syntax errors (Just for the .nmis file) with the following command 'perl -c /usr/local/omk/conf/opCommon.nmis'. If you are not scared you should be 🙂

```
    "opevents_logs" : {
       "traplog" : [
         "<nmis9_logs>/snmptrap.log"
       ],
```

Modify EventParserRules.nmis

EventParserRules.nmis is where parsing generally occurs. In this case we are anticipating some complex maneuvers; so we are going to tell EventParserRules to send this to an opEvents plugin where complexity is better dealt with. Remember all that big bad syntax talk? Same applies here.

We would need to review the trap format. Usually they look like the following:

```
May 14 16:59:21 localhost snmptrapd[17772]: 2021-05-14 17:04:21 UDP: [127.0.0.1]:38166->[127.0.0.1]:162 [UDP:
[127.0.0.1]:38166->[127.0.0.1]:162]:#012RFC1213-MIB::sysUpTime.0 = 0:0:00:00.00#011SNMPv2-MIB::snmpTrapOID.0 =
BGP4-MIB::bgpBackwardTransition#011OPMANTEK-MIB::omkNotifications = "Events"
```

Based on this, we will need to add the following rules to EventParserRules.nmis/EventParserRules.json, in order to be processed:

```
     "traplog" : {
      "1" : {
         "IF" : "SNMPv2-MIB::snmpTrapOID",
         "THEN" : {
            "6" : {
               "THEN" : [
                  "capture(date)"
               ],
               "DESCRIPTION" : "first match date/time",
               "IF" : "(\\d{4}-\\d\\d-\\d\\d \\d\\d:\\d\\d:\\d\\d)"
            },
            "12" : {
               "THEN" : [
                  "capture(host)"
               ],
               "DESCRIPTION" : "host captured",
               "IF" : "(\\d+\\.\\d+\\.\\d+\\.\\d+)"
            },
            "68" : {
               "THEN" : [
                  "set.event(OMK Notifications)",
                  "set.stateful(OMK Notifications)",
                  "set.state(up)",
                  "set.priority(2)"
               ],
               "IF" : "OPMANTEK-MIB::omkNotifications"
            }
...
```

We can add as many rules and captures as we need. Here you can find further information.