

SNMP Trap Forwarder User's Manual

Copyright 2004, Xenadyne Inc, All rights Reserved

1. Overview

The trapfwd program is designed to receive SNMP traps (UDP datagrams) and forward each trap to one or more destinations. The data content of the traps can also be written to a program that is spawned by the application. Decisions about how to handle SNMP messages that have been received are made according to a set of rules and associated actions. The application supports logging via syslog, dumping of network data, and statistics reporting.

Network management systems that utilize the SNMP protocol can face various difficult issues in the handling of trap messages:

- It is often necessary to redirect traps to one of a number of management applications for further processing,
- It can be useful to send multiple copies of a trap to different management applications, either for redundancy or for other purposes,
- It is often necessary to redirect traps from the public internet to hosts inside a private subnetwork.

The Xenadyne SNMP trap forwarder provides a general, flexible solution to these problems. The trap forwarder is a software tool that will receive traps and forward them, based on a set of rules, to another destination, without modifying the source address. This tool will forward packets intelligently based on the source and destination addresses and ports, and it is capable of making duplicate copies of traps in order to support redundant configurations.

2. Administration

2.1 General Operation

The trapfwd program operates in an event-driven manner, where the event sources are incoming SNMP traps. The processing of these incoming packets is based on a set of rules defined by the user. Each rule defines upon which packets a processing action or actions will be performed. Packet processing actions include forwarding the packet to another location, dropping the packet, or starting a command and sending the data portion of the packet to the command's standard input.

2.1.1 Starting, Stopping, and Restarting

We have provided the 'trapfwd_ctl' shell script to simplify the operations of starting, stopping and restarting the trapfwd daemon. To use this script, you must first edit it to set correct paths to your configuration and runtime files. The trapfwd_ctl script takes arguments 'start', 'stop' or 'restart', as in these examples:

```
# trapfwd_ctl start
# trapfwd_ctl stop
# trapfwd_ctl restart
```

To automatically start the trap forwarder daemon on system boot, you should install the shell script 'trapfwd_ctl' in your /etc/rc3.d directory. Be sure to edit this file to provide correct paths to the executable and configuration files.

2.2 Logging and Error Handling

2.2.1 Setup

Before using this application you must configure the syslog.conf file. The application uses the syslog facility LOG_LOCAL0, so an entry for this facility (or other facilities that are used) must exist in the /etc/syslog.conf file. Minimally you add an entry that looks like the following:

```
local0.*                /var/log/trapfwd.log
```

This says to write all entries, for the local0 facility, to the file "/var/log/trapfwd.log". See the notes below before configuring syslog.

2.2.2 Using Syslog

- ⇒ In /etc/syslog.conf, put your log specification entry BEFORE any lines beginning with '!'.
- ⇒ The syslogd daemon reads its configuration file when it starts up and whenever it receives a hangup (SIGHUP) signal. You must create the necessary entry in the /etc/syslog.conf file, and you must also create the log file (touch /var/log/logfilename) before restarting syslogd (pkill -HUP syslogd) before syslog can be used to write to the log file. The syslogd daemon will not create the new log file for you.
- ⇒ The file permissions on the log file can be "644 root wheel" because syslogd is running as root.

2.3 User Interface

2.3.1 Synopsis

```
trapfwd [-chkFW] [-f file] [-l log_sev] [-p pid_file ] [-k signal]
```

-c Write log entries to stderr also.

-f Use given config-file instead of /usr/local/trapfwd/trapfwd.conf

-h Print help message.

-k restart | shutdown | report | parse

Parse configuration file, then send signal to running copy (except -k parse) and exit.

- p Use given pid-file instead of /var/run/trapfwd.pid
- F Run in foreground, do not become daemon.
- W Wait for debugger to attach.
- X Force full debugging.

2.3.3 Options

- c This option causes log messages to be written to the standard error output as well as the system log. For this option to be available, the system's syslog must support the LOG_PERROR option.

-f file

This option specifies a configuration file instead of /usr/local/trapfwd/trapfwd.conf.

- h Print help information as it appears in the above synopsis.

-k restart | shutdown | report | parse

Parse configuration file, then send signal to running copy (except -k parse) and exit.

The "restart" option causes the running copy to parse and reload the configuration file and recreate all sockets.

The "shutdown" option sends a signal to shutdown the running copy. The "report" option causes the running copy to generate a statistics report.

The "parse" option only parses the configuration file in order to check for errors.

-p file

This option specifies to use the given pid file instead of /var/run/trapfwd.pid.

- F This option causes the trapfwd application to run in the foreground and not become a daemon. Running in the foreground will also cause statistics reports and data dumps to be written to the console instead of their respective files.

- W This option will cause the application to enter an infinite wait loop to allow a debugger to attach at a convenient point of execution. This point is located immediately after the location where the application becomes a daemon.

- X This option causes detailed messages to be written to the system log. This option should only be enabled while debugging, because of the large volume of data that is written to the log.

2.3.4 Files

/usr/local/trapfwd/trapfwd.conf

The main configuration file. You must initially create this file for trapfwd to work. This document describes how to construct this file, and sample configuration files are provided with this software.

`/var/run/trapfwd.pid`

This is where the process ID (pid) of the currently running application is stored. When the application is terminated, this file is deleted.

`/var/log/trapfwd.dump`

Data dump output file. Data from datagrams matching a rule with the "dump" flag set will be written as a hex dump to this file. Each dump will be preceded by the datagram's network parameters and the time of receipt.

`/var/log/trapfwd.report`

Statistics report output. When the trapfwd application is running as a daemon and is asked to generate a statistics report, the report is written to this file. Then trapfwd is not a daemon, but is running in the foreground, the report data is written to the console.

3. Configuration

3.1 Description

The trapfwd application is configured using a configuration file. This file specifies how to listen for datagrams, what datagrams to receive, and what to do with received datagrams. The application will look for the configuration file, `/usr/local/trapfwd/trapfwd.conf`, unless otherwise specified using the `-f` command line option.

3.1.1 Where to Listen for Incoming Data

The application listens on one or more sockets specified by the binding parameters that follow the "listen:" line in the configuration file. The binding parameters for a socket (a binding) is the IP address of a network interface and a port number, separated by a colon. Any number of bindings may be specified (up to the limit of your system's settings). The network interface specified in each binding must be a valid interface for the host.

3.1.2 Datagram Matching Rules

Following the bindings section of the configuration file is the rules section. This section consists of a set of datagram matching rules and actions associated with each rule. A matching rule is a set of network parameters that specify what incoming datagrams will match the rule. The application will match a rule if all parameters specified in the rule match all network parameters of an incoming datagram. These parameters to match are as follows:

- source address/netmask
- source port
- destination address/netmask
- destination port

When specifying addresses and masks, both source and destination, a number of different formats can be used. The address can be an IP address represented in dot ('.') notation, or it can be a hostname. The netmask is optional, and is assumed to be a host mask (all 32 bits set) if not specified. The netmask can be specified as a decimal value from 0 to 32. This value specifies the number of bits to

compare in the address specified in the matching and the corresponding address of the incoming datagram. If the netmask is 0, then this address parameter will match the corresponding address parameter of any datagram. This is equivalent to using the "any" keyword which can also be used in place of an address/netmask.

Ports are specified by numbers from 0 to 65535. A port number 0 is equivalent to specifying "any" for the port, which will match the corresponding port of any incoming datagram.

Although the "any" keyword is not strictly necessary for either the address/netmask or port values, it is provided for convenience and to increase readability of rules.

Each matching rule may be prefixed with optional flags. These flags are "log" and "dump". Specifying the log flag will cause a log entry to be made whenever a datagram matches the rule. The dump flag specifies that the data content of the datagram will be written to the dump file whenever a datagram matches the rule. For more information on the dump file, see section [4.3 Data Dump].

3.1.3 Action List - What to Do With Network Data

A match rule has an associated list of one or more actions. Then these actions will be performed on each datagram that matches the rule. Each action in the action list may specify one of the following:

- Forward the trap to another host.
- Run a program to process the trap data.
- Drop (ignore) the trap datagram.

Any number of action may be specified for each rule. The actions will be executed in the order listed in the configuration file.

3.1.3.1 Forward Action

The "forward" action is specified by a target address and an optional port. If the port is omitted then the original destination port will be used as the new destination port. If multiple forward actions are specified, then the datagram will be forwarded to each destination listed. Do not forward to the same interfaces and ports that forwarded datagrams were received on, as this may result in the application spinning in an endless receive and forward loop.

3.1.3.2 Program Action

The program action begins with a pipe '|' character followed by a program name and an optional login name. If a login name is present, then the program will be run as that user, otherwise it will run as root. The program will run as a separate process and the packet contents will be sent to the program's standard input.

3.1.3.3 Drop Action

If the action specified it "drop" , then the packet is discarded after performing and any tasks indicated by the optional flags, "log" or "dump". This action is usually useful as a "catch-all" to log any traps not caught by previous rules. It can also be useful for filtering out the reception of any specific unwanted datagrams.

3.2 SNMP Forwarder Configuration File Syntax

This section presents a more precise description of the configuration file syntax by specifying a grammar that describes the syntax. The syntax is as follows:

```
listen_spec rule_list

listen_spec:      listen: bind_list

bind_list:       binding
                   | binding bind_list

binding:         ip_addr : portnum

rule_list:       rule
                   | rule rule_list

rule:            flag_list match_spec action_list
                   | match_spec action_list

flag_list:       flag
                   | flag flag_list

flag:            log | dump

match_spec:      src_addr src_port dst_addr dst_port

action_list:     action
                   | action action_list

action:          -> target_addr target_port
                   | -> | program
                   | -> drop

src_addr:        match_addr

dst_addr:        match_addr

src_port:        match_port

dst_port:        match_port

match_addr:     any | net | address

match_port:     any | portnum

target_addr:    address

target_port:    <empty> | portnum

address:         ip_addr
                   | name
```

```

net:          ip_addr/mask

ip_addr:     [0-255].[0-255].[0-255].[0-255]

mask:        [0-32]

portnum:     [0-65535]

```

3.3 Example Configuration File:

```

# Recv traps on these ports and interfaces (optionally specified)
listen: 162 2701 192.168.0.44:16200 192.168.0.43:16200

# Fwd traps from 192.168.0.22 to 172.16.3.31
192.168.0.22 any any any -> 172.16.3.31 162

# Fwd traps from 192.168.0.23 to 172.16.3.31 and to 172.16.3.32
192.168.0.23 any any any -> 172.16.3.31 162
                    -> 172.16.2.32 162

# Fwd traps from 192.168.X.X network to 172.16.3.33 on port 162
192.168.0.0/16 any any any -> 172.16.3.33 162

# Send any trap sent to port 2727, to the alerter program
any any any 2727 -> | /usr/local/sbin/alerter

# Fwd and log all traps sent to .44 address (on this host)
# to 10.1.2.3 on original destination port.
# Also sent to stdin of snmpprinter program.
log any any 192.168.0.44 any -> 10.1.2.3
                    -> | /usr/local/sbin/snmpprinter

# Log and dump contents of any trap not matched by previous rules.
log dump any any any any -> drop

```

Note: In the case of a trap from 192.168.0.22, it would be forwarded to 172.16.3.31. This is because the first rule matched is the one that is followed.

3.4 Parsing the Configuration File

The trapfwd application offers a command line option (-k parse) to do nothing more than parse the configuration file. This option is useful to check the syntax of a configuration file before actually running the application. If the configuration file is successfully parsed, then the application will print out the listen specification, and the matching rules and the associated actions in a normalized format. If there is an error in the configuration file then the line number of the error will be printed. This option can be used at any time whether the application is running or not.

4. Diagnostics

4.1 Event Logging

Event logging is accomplished using the UNIX syslog facility. This decision was made for a number of reasons. This system has the flexibility to allow operations staff have the ability to configure the destination of messages sent from the processes running on the system. The system can be configured so that events that are received by the event log process can be logged to different files and also displayed on the console of the device. It is also possible for the event log process to be configured to forward the messages across a network to the event log process on another machine.

Furthermore, syslog is a stable process as it has to be to serve as the operating system's and many other applications' primary means of logging events and errors. It is a standard component on UNIX platforms which means very little development time is required to implement the log facility, and little or no testing time is required to qualify syslog as operational.

4.1.1 Log Levels

Event logging will be configurable to record log entries at or below the configured severity level. This means that if a log entry is tagged with a certain severity level indicator it will be recorded only if this level is less than or equal to the level that the log module is configured to log at. Every log message will have a severity level indicator represented by an integer value. The syslog severity levels are specified in RFC 3164. These are described below along with their numerical values.

- 0 Emergency
 Situation so severe that the system is unstable.

- 1 Alert
 Situation requires that action be taken immediately. Sending notification to system administrators should be mandatory for this severity level.

- 2 Critical
 Critical conditions that will cause the system or component to stop working, generally in a manner that is not automatically recoverable.

- 3 Error
 Error conditions or failures, usually sufficiently critical to prevent the program from executing one or more intended tasks.

- 4 Warning
 Undesirable conditions that should not normally occur during proper execution with ideal configuration, but that are not critical enough to stop the program from executing intended tasks.

- 5 Notice
 Events or conditions that are important to record in the log file. This level is appropriate for required audit trail data.

- 6 Informational
 Data that is informative, but its record is not crucial.

- 7 Debug

Very detailed information the is only for debugging program logic or configuration. This level is normally only useful to developers.

4.2 Reporting

Reports of current operating statistics are available at any time while the trapfwd application is running. Using the "-k report" command line option will cause the running instance of the application to generate a statistics report. This report is written to the report file, or to the console if the application is running in the foreground. Following is a sample statistics report:

```
***** trapfwd report *****
running 3 days 5:21 -- since Sat Feb  1 03:02:24 2003
receiving traps on: 192.168.0.4:162 192.168.0.4:2701
Total traps received: 20
Total traps matches by rules: 18
```

Hit count for each rule:

```
-----
8      1) 192.168.0.22/0xffffffff any any any
2      2) 192.168.0.23/0xffffffff 9999 any any
3      3) 192.168.0.0/0xffff0000 any any any
4      4) any any any 2727
0      5) log any any 192.168.0.4/0xffffffff any
1      6) log dump any any any any
```