Processing messages stuck in the disk queue files of syslog-ng Premium Edition

June 19, 2018

Table of Contents

Problem

When you change the configuration of a syslog-ng PE host that uses disk-based buffering (also called dis queue), syslog-ng PE may start new disk buffer files for the destinations that you have changed. In such case, syslog-ng PE abandons the old disk queue files. If there were unsent log messages in the disk queue files, these messages remain in the disk queue files, and will not be sent to the destinations.

This document explains the steps required to find, examine, and flush the log messages from such orphaned disk queue files.

1. Procedure – Recover log messages from orphaned disk queue files

Overview:

1. *Identify the active queue files*
2. *Identify which queue files still hold valid data*
3. *Configure a separate syslog instance to send queue files to the processing application*

Steps:

Step 1. **Identify the active queue files.**
The syslog-ng PE application keeps track of active disk queue files, and the internal state of its source drivers in the `syslog-ng.persist` file. While running, syslog-ng PE uses the `mmap()` system call to map the file's contents into physical memory. This means that the actual contents of the file may not always contain the up-to-date internal state of syslog-ng PE. For this reason, while you are working with the `syslog-ng.persist` file, stop syslog-ng PE.

The following command lists the destinations and the related queue files.

```
# /opt/syslog-ng/bin/persist-tool dump /opt/syslog-ng/var/syslog-ng.persist
 | fgrep qfile
```

The output if this command is similar to the following:

```
afsocket_dd_qfile(stream,127.0.0.1:601) = { "queue_file":
"\/\/syslog-ng-00001.rqf" }
```

Step 2. **Identify which queue files hold valid data.**
To identify which queue files hold unsent data, use the following two commands for your disk queue files (the example shows a single file called `syslog-ng-00000.rqf`):

```
# /opt/syslog-ng/bin/dqtool info syslog-ng-00000.rqf
# /opt/syslog-ng/bin/dqtool cat syslog-ng-00000.rqf
```

```
root@server:/# /opt/syslog-ng/bin/dqtool info syslog-ng-00000.rqf
Reliable disk-buffer state loaded; filename='syslog-ng-00000.rqf',
queue_length='138', size='71962'
root@server:/# /opt/syslog-ng/bin/dqtool cat syslog-ng-00000.rqf | tail -n
3
Reliable disk-buffer state loaded; filename='syslog-ng-00000.rqf',
queue_length='138', size='71962'
Feb 20 17:22:14.776 server -- MARK --
Feb 20 17:42:14.777 server -- MARK --
Feb 20 18:02:14.778 server -- MARK --
root@server:/#
```

To identify queue files with valid data in them, use the following command. This command prints the names of disk queue files which hold valid data.

```
# for q in *.rqf; do /opt/syslog-ng/bin/dqtool info ${q} 2>&1 | fgrep
queue_length; done | awk -F \' '{ if ($4 > 0) { print $2; } }'
```

Verify that the contents of the queue files are indeed valid. If you want to forward the messages to an application, verify that the receiving application will be able to parse them.

Step 3. **Configure a separate instance of syslog-ng PE to send queue files to the processing application.** In order to process the queue files that contain valid data, you must configure and temporarily run a separate syslog-ng PE instance.

The configuration must include a source that will definitely not receive any logs, connected to a network destination that points to the desired IP address and port number, and has disk queue configured. When you start syslog-ng PE with this configuration, it will generate a persist file that you can modify later on.

Step a. Create an appropriate configuration for your environment that matches the above criteria. For example:

```
@version:6.0
@include "scl.conf"
#
# sample configuration file for syslog-ng on AIX
# users should customize to fit their needs
#

options {
        threaded(yes);
        keep-hostname(yes);
        keep-timestamp(yes);
};

source nofile {
        file (
                "/no_such_file_or.dir"
        );
};

destination extra_listener {
        syslog(
                "127.0.0.1"
                port(10641)
                disk-buffer(
                        disk-buf-size(1048576)
                        reliable(yes)
                )
        );
};

log {
        source(nofile);
        destination(extra_listener);
};
```

Step b. **Start syslog-ng PE briefly from the command-line to generate a persist file.**
Make sure to use the configuration file you created in the previous step using the *--cfgfile* option, and to use a non-existing persist file (to avoid overwriting the persist file of your regular syslog-ng PE instance). The following command uses the `/root/syslog/syslog-ng.conf` configuration file, and the `/root/syslog/syslog-ng.persist` persist file.

```
root@server:~/syslog# /opt/syslog-ng/sbin/syslog-ng --foreground
 --enable-core --no-caps --cfgfile /root/syslog/syslog-ng.conf
--pidfile /root/syslog/syslog-ng.sender.pid --control
/root/syslog/syslog-ng.sender.ctl --persist-file
/root/syslog/syslog-ng.persist --qdisk-dir /root/syslog/
```

After syslog-ng PE starts up and generates the persist file, press **CTRL+C** to stop syslog-ng PE.

Step c. **Edit the persist file to include the location of the orphaned disk queue files.**
Use the following `/opt/syslog-ng/bin/persist-tool dump /root/syslog/syslog-ng.persist` command to display the contents of the persist file generated in the previous step, for example:

```
root@server:~/syslog# /opt/syslog-ng/bin/persist-tool dump
/root/syslog/syslog-ng.persist
afsocket_dd_qfile(stream,127.0.0.1:10641) = { "queue_file":
"\/root\/syslog\/\/syslog-ng-00000.rqf" }

affile_sd_curpos(/no_such_file_or.dir) = { "version": 1,
"big_endian": false, "raw_buffer_leftover_size": 0, "buffer_pos":
 0, "pending_buffer_end": 0, "buffer_size": 0,
"buffer_cached_eol": 0, "pending_buffer_pos": 0, "raw_stream_pos":
 0, "pending_raw_stream_pos": 0, "raw_buffer_size": 0,
"pending_raw_buffer_size": 0, "file_size": 0, "file_inode": 0,
"run_id": 1 }

run_id = { "value": "01 00 00 00" }
```

Issue the following commands to modify the persist file.

```
/opt/syslog-ng/bin/persist-tool dump
/root/syslog/syslog-ng.persist > persist.dump
sed -i -e 's:syslog-ng-00000:full:' -e '/^run_id/ d' -e '/^$/ d'
 persist.dump
rm syslog-ng.persist
/opt/syslog-ng/bin/persist-tool add persist.dump -o .
```

As a result, references to the `syslog-ng-00000.rqf` disk queue file should change to `full.rqf`. Display the contents of the persist file again to verify this.

```
root@server:~/syslog# /opt/syslog-ng/bin/persist-tool dump
/root/syslog/syslog-ng.persist
affile_sd_curpos(/no_such_file_or.dir) = { "version": 1,
"big_endian": false, "raw_buffer_leftover_size": 0, "buffer_pos":
 0, "pending_buffer_end": 0, "buffer_size": 0,
"buffer_cached_eol": 0, "pending_buffer_pos": 0, "raw_stream_pos":
 0, "pending_raw_stream_pos": 0, "raw_buffer_size": 0,
"pending_raw_buffer_size": 0, "file_size": 0, "file_inode": 0,
"run_id": 1 }

afsocket_dd_qfile(stream,127.0.0.1:10641) = { "queue_file":
"\/root\/syslog\/\/full.rqf" }
```

Step d. **Rename the queue file to the filename set in the persist file previously.**

```
root@server:~/syslog# ls -l *.rqf
-rw------- 1 root root  4096 febr  21 23:57 full.rqf
-rw------- 1 root root 78506 febr  22 20:45 syslog-ng-00000.rqf
root@server:~/syslog# cp /syslog-ng-00000.rqf full.rqf
root@server:~/syslog# ls -l *.rqf
-rw------- 1 root root 78506 febr  22 20:45 full.rqf
-rw------- 1 root root 78506 febr  22 20:45 syslog-ng-00000.rqf
```

Step e. **Start the new syslog-ng instance.**

Start the new syslog-ng instance, and let it run until size of the queue file decreases to 4 KB. After that, press **Ctrl+C** to stop the syslog-ng instance.

```
/opt/syslog-ng/sbin/syslog-ng --foreground --enable-core --no-caps
 --cfgfile /root/syslog/syslog-ng.conf --pidfile
/root/syslog/syslog-ng.sender.pid --control
/root/syslog/syslog-ng.sender.ctl --persist-file
/root/syslog/syslog-ng.persist --qdisk-dir /root/syslog/
^C
root@server:~/syslog# ls -l *.rqf
-rw------- 1 root root  4096 febr  22 22:19 full.rqf
-rw------- 1 root root 78506 febr  22 20:45 syslog-ng-00000.rqf
root@server:~/syslog#
```

If you wish to verify or debug syslog-ng PE sending the queue file contents, use the additional `--verbose --debug --stderr` options, for example:

```
/opt/syslog-ng/sbin/syslog-ng --foreground --verbose --debug
--stderr --enable-core --no-caps --cfgfile
/root/syslog/syslog-ng.conf --pidfile
/root/syslog/syslog-ng.sender.pid --control
/root/syslog/syslog-ng.sender.ctl --persist-file
/root/syslog/syslog-ng.persist --qdisk-dir /root/syslog/
```

Step f. **Repeat these steps for the other left-over queue files.**

After you have processed all left-over queue files this way, all the missing recoverable logs should have found their way to their intended destinations.

2. About One Identity

One Identity LLC, is a leading provider of Privileged Access Management (PAM) and Log Management solutions. Founded in 2000, One Identity has a proven track record of helping businesses reduce the risk of data breaches associated with privileged accounts. With offices in the United States and Europe, and a global client list that includes 25 Fortune 100 companies, One Identity and its network of reseller partners serves more than 1,000,000 corporate users worldwide.

For more information, visit *syslog-ng.com*, read the *syslog-ng blog*, or follow us on Twitter via @balabit, LinkedIn or Facebook.

To learn more about commercial and open source One Identity products, request an evaluation version, or find a reseller, visit the following links:

- *The syslog-ng homepage*
- *syslog-ng Documentation page*
- *Contact us and request an evaluation version*

About One Identity

One Identity helps organizations optimize identity and access management (IAM). Our combination of offerings, including a portfolio of identity governance, access management, privileged management and identity as a service solutions, enables organizations to achieve their full potential — unimpeded by security, yet safeguarded against threats. For more information, visit *oneidentity.com*.

Appendix A. About disk queue files

Normal and reliable queue files

The key difference between disk queue files that employ the `reliable(yes)` option and not is the strategy they employ. Reliable disk queues guarantee that all the messages passing through them are written to disk first, and removed from the queue only after the destination has confirmed that the message has been successfully received. This prevents message loss, for example, due to syslog-ng PE crashes if the the source side of the destination server uses RLTP™. Of course, this introduces a significant performance penalty as well. Reliable disk queues employ an in-memory cache buffer, the content of which is also written to the disk, and which is intended to speed up the process of reading back data from the queue.

Normal disk queues work in a different way: they employ an in-memory output buffer (set in *`qout-size()`*) and an in-memory overflow queue (set in *`mem-buf-length()`*). The disk buffer file itself is only used if the overflow buffer is filled up completely. This approach has better performance (because of less disk IO operations), but also carries the risk of losing a maximum of *`qout-size()`* plus *`mem-buf-length()`* number of messages in case of an unexpected power failure or application crash.

Size and truncation of queue files

Disk queue files tend to grow. Each may take up to *`disk-buf-size()`* bytes on the disk. Due to the nature of reliable queue files, all the messages traversing the queue are written to disk, constantly increasing the size of the queue file. Truncation only occurs if the read and write heads of the queue reach the same position. Given that new messages arrive all the time, at least a small number of messages will almost always be stored in the queue file at all times. As a result, the queue file is not truncated automatically, but grows until it reaches the maximal configured size, after which the write head will wrap around, later followed by the read head.

In case of normal disk queue files, growth in size is not so apparent, as the disk-based queue file is only used if the in-memory overflow buffer fills up. Once the destination sends messages faster than the incoming message rate, the queue will start to empty, and when the read and write heads of the queue reach the same position, the queue files are finally truncated.

Note that if a queue file becomes corrupt, syslog-ng PE starts a new one. This might lead to the queue files consuming more space in total than their maximal configured size and the number of configured queue files multiplied together.