

NETWORK SECURITY HACKS™

100 Industrial-Strength Tips & Tools



O'REILLY®

Andrew Lockhart

HACK
#59

Aggregate Logs from Remote Sites

Integrate collocated and other remote systems or networks into your central syslog infrastructure.

Monitoring the logs of a remote site or just a collocated server can often be overlooked when faced with the task of monitoring activity on your local network. You could use the traditional syslog facilities to send logging information from the remote network or systems, but since the syslog daemon uses UDP for sending to remote systems, this is not the ideal solution. UDP provides no reliability in its communications, and so you risk losing logging information. In addition, the traditional syslog daemon has no means to encrypt the traffic that it sends, so your logs might be viewable by anyone with access to the intermediary networks between you and your remote hosts or networks.

To get around these issues, you'll have to look beyond the syslog daemon that comes with your operating system and find a replacement. One such replacement syslog daemon is *syslog-ng* (<http://www.balabit.com/products/syslog-ng/>). *syslog-ng* is not only a fully functional replacement for the traditional syslog daemon, but also adds flexible message filtering capabilities, as well as support for logging to remote systems over TCP (in addition to support for the traditional UDP protocol). With the addition of TCP support, you can also employ *stunnel* or *ssh* to securely send the logs across untrusted networks.

To build *syslog-ng*, you will need the *libol* library package (<http://www.balabit.com/downloads/syslog-ng/libol/>) in addition to the *syslog-ng* package. After downloading these packages, unpack them and then build *libol*:

```
$ tar xzf libol-0.3.9.tar.gz
$ cd libol-0.3.9
$ ./configure && make
```

When you build *syslog-ng* you can have it statically link to *libol*, so there is no need to fully install the library.

And now to build *syslog-ng*:

```
$ tar xzf syslog-ng-1.5.26.tar.gz
$ cd syslog-ng-1.5.26
$ ./configure --with-libol=../libol-0.3.9
$ make
```

If you want to compile in TCP wrappers support, you can add the `--enable-tcp-wrapper` flag to the configure script. After *syslog-ng* is finished compiling, become root and run `make install`. This will install the *syslog-ng* binary and manpages. To configure the daemon, create the `/usr/local/etc/syslog-ng` directory and then create a *syslog-ng.conf* to put in it. To start off with, you

can use one of the sample configuration files in the *doc* directory of the *syslog-ng* distribution.

There are five types of configuration file entries for *syslog-ng*, each of which begins with a specific keyword. The *options* entry allows you to tweak the behavior of the daemon, such as how often the daemon will sync the logs to the disk, whether the daemon will create directories automatically, and host-name expansion behavior. *source* entries tell *syslog-ng* where to collect log entries from. A source can include Unix sockets, TCP or UDP sockets, files, or pipes. *destination* entries allow you to specify possible places for *syslog-ng* to send logs to. You can specify files, pipes, Unix sockets, TCP or UDP sockets, TTYs, or programs. Sources and destinations are then combined with filters (using the *filter* keyword), which let you select syslog facilities and log levels. Finally, these are all used together in a *log* entry to define precisely where the information is logged. Thus you can arbitrarily combine any source, select what syslog facilities and levels you want from it, and then route it to any destination. This is what makes *syslog-ng* an incredibly powerful and flexible tool.

To set up *syslog-ng* on the remote end so that it can replace the *syslogd* on the system and send traffic to a remote *syslog-ng*, you'll first need to translate your *syslog.conf* to equivalent source, destination, and log entries.

Here's the *syslog.conf* for a FreeBSD system:

```
*.err;kern.debug;auth.notice;mail.crit      /dev/console
*.notice;kern.debug;lpr.info;mail.crit;news.err /var/log/messages
security.*                                   /var/log/security
auth.info;authpriv.info                     /var/log/auth.log
mail.info                                    /var/log/maillog
lpr.info                                     /var/log/lpd-errs
cron.*                                       /var/log/cron
*.emerg                                      *
```

First you'll need to configure a source. Under FreeBSD, */dev/log* is a link to */var/run/log*. The following source entry tells *syslog-ng* to read entries from this file:

```
source src { unix-dgram("/var/run/log"); internal(); };
```

If you were using Linux, you would specify *unix-stream* and */dev/log* like this:

```
source src { unix-stream("/dev/log"); internal(); };
```

The *internal()* entry is for messages generated by *syslog-ng* itself. Notice that you can include multiple sources in a source entry. Next, include destination entries for each of the actual log files:

```
destination console { file("/dev/console"); };
```

Aggregate Logs from Remote Sites

```
destination messages { file("/var/log/messages"); };
destination security { file("/var/log/security"); };
destination authlog { file("/var/log/auth.log"); };
destination maillog { file("/var/log/maillog"); };
destination lpd-errs { file("/var/log/lpd-errs"); };
destination cron { file("/var/log/cron"); };
destination slip { file("/var/log/slip.log"); };
destination ppp { file("/var/log/ppp.log"); };
destination allusers { usertty("*"); };
```

In addition to these destinations, you'll also want to specify one for remote logging to another *syslog-ng* process. This can be done with a line similar to this:

```
destination loghost { tcp("192.168.0.2" port(5140)); };
```

The port number can be any available TCP port.

Defining the filters is straightforward. You can simply create one for each *syslog* facility and log level, or you can create them according to those used in your *syslog.conf*. If you do the latter, you will only have to specify one filter in each log statement, but it will still take some work to create your filters.

Here are example filters for the *syslog* facilities:

```
filter f_auth { facility(auth); };
filter f_authpriv { facility(authpriv); };
filter f_console { facility(console); };
filter f_cron { facility(cron); };
filter f_daemon { facility(daemon); };
filter f_ftp { facility(ftp); };
filter f_kern { facility(kern); };
filter f_lpr { facility(lpr); };
filter f_mail { facility(mail); };
filter f_news { facility(news); };
filter f_security { facility(security); };
filter f_user { facility(user); };
filter f_uucp { facility(uucp); };
```

and examples for the log levels:

```
filter f_emerg { level(emerg); };
filter f_alert { level(alert..emerg); };
filter f_crit { level(crit..emerg); };
filter f_err { level(err..emerg); };
filter f_warning { level(warning..emerg); };
filter f_notice { level(notice..emerg); };
filter f_info { level(info..emerg); };
filter f_debug { level(debug..emerg); };
```

Now you can combine the source with the proper filter and destination within the log entries:

```
# *.err;kern.debug;auth.notice;mail.crit /dev/console
log { source(src); filter(f_err); destination(console); };
```

```

log { source(src); filter(f_kern); filter(f_debug); destination(console); };
log { source(src); filter(f_auth); filter(f_notice); destination(console); };
log { source(src); filter(f_mail); filter(f_crit); destination(console); };

# *.notice;kern.debug;lpr.info;mail.crit;news.err      /var/log/messages
log { source(src); filter(f_notice); destination(messages); };
log { source(src); filter(f_kern); filter(f_debug); destination(messages); };
log { source(src); filter(f_lpr); filter(f_info); destination(messages); };
log { source(src); filter(f_mail); filter(f_crit); destination(messages); };
log { source(src); filter(f_news); filter(f_err); destination(messages); };

# security.*                                          /var/log/security
log { source(src); filter(f_security); destination(security); };

# auth.info;authpriv.info                            /var/log/auth.log
log { source(src); filter(f_auth); filter(f_info); destination(authlog); };
log { source(src); filter(f_authpriv); filter(f_info); destination(authlog); };

# mail.info                                          /var/log/maillog
log { source(src); filter(f_mail); filter(f_info); destination(maillog); };

# lpr.info                                           /var/log/lpd-errs
log { source(src); filter(f_lpr); filter(f_info); destination(lpd-errs); };

# cron.*                                             /var/log/cron
log { source(src); filter(f_cron); destination(cron); };

# *.emerg                                           *
log { source(src); filter(f_emerg); destination(allusers); };

```

You can set up the machine that will be receiving the logs in much the same way as if you were replacing the currently used *syslogd*.

To configure *syslog-ng* to receive messages from a remote host, you must specify a source entry:

```
source r_src { tcp(ip("192.168.0.2") port(5140)); };
```

Alternatively, you can dump all the logs from the remote machines into the same destinations that you use for your local log entries. This is not really recommended, because it can be a nightmare to manage, but can be done by including multiple source drivers in the source entry that you use for your local logs:

```
source src {
    unix-dgram("/var/run/log");
    tcp(ip("192.168.0.2") port(5140));
    internal();
};
```

Now logs gathered from remote hosts will appear in any of the destinations that were combined with this source.

Aggregate Logs from Remote Sites

If you would like all logs from remote hosts to go into a separate file named for each host in `/var/log`, you could use a destination like this:

```
destination r_all { file("/var/log/$HOST"); };
```

`syslog-ng` will expand the `$HOST` macro to the hostname of the system sending it logs and create a file named after it in `/var/log`. An appropriate log entry to use with this would be:

```
log { source(r_src); destination(r_all); };
```

However, an even better method is to recreate all of the remote `syslog-ng` log files on your central log server. For instance, a destination for a remote machine's messages file would look like this:

```
destination fbsd_messages { file("/var/log/$HOST/messages"); };
```

Notice here that the `$HOST` macro is used in place of a directory name. If you are using a destination entry like this, be sure to create the directory beforehand, or use the `create_dirs()` option:

```
options { create_dirs(yes); };
```

`syslog-ng`'s macros are a very powerful feature. For instance, if you wanted to separate logs by hostname and day, you could use a destination like this:

```
destination fbsd_messages {  
    file("/var/log/$HOST/$YEAR.$MONTH.$DAY/messages");  
};
```

You can combine the remote source with the appropriate destinations for the logs coming in from the network just as you did when configuring `syslog-ng` for local logging—just specify the remote source with the proper destination and filters.

Another neat thing you can do with `syslog-ng` is collect logs from a number of remote hosts and then send all of those to yet another `syslog-ng` daemon. You can do this by combining a remote source and a remote destination with a log entry:

```
log { source(r_src); destination(loghost); };
```

Since `syslog-ng` is now using TCP ports, you can use any encrypting tunnel you like to secure the traffic between your `syslog-ng` daemons. You can use [SSH port forwarding \[Hack #72\]](#) or [stunnel \[Hack #76\]](#) to create an encrypted channel between each of your servers. By limiting connections on the listening port to include only localhost (using firewall rules, as in [“Firewall with Net-filter” \[Hack #33\]](#) or [“Firewall with OpenBSD’s PacketFilter” \[Hack #34\]](#)), you can eliminate the possibility of bogus log entries or denial-of-service attacks.

Server logs are among the most critical information that a system administrator needs to do her job. Using new tools and strong encryption, you can keep your valuable log data safe from prying eyes.

```

sh                andrew  ??          0.03 secs Mon Dec 15 05:44
gunzip            andrew  ??          0.00 secs Mon Dec 15 05:44

# lastcomm bash
bash              F    andrew  ??          0.00 secs Mon Dec 15 06:44
bash              F    root    stdout     0.01 secs Mon Dec 15 05:20
bash              F    root    stdout     0.00 secs Mon Dec 15 05:20
bash              F    andrew  ??          0.00 secs Mon Dec 15 05:19

```

To summarize the accounting information, you can use the `sa` command. By default it will list all the commands found in the accounting logs and print the number of times that each one has been executed:

```

# sa
 14      0.04re      0.03cp      0avio      1297k      troff
   7      0.03re      0.03cp      0avio      422k      lastcomm
   2     63.90re      0.01cp      0avio      983k      info
  14     34.02re      0.01cp      0avio      959k      less
  14      0.03re      0.01cp      0avio     1132k      grotty
  44      0.02re      0.01cp      0avio      432k      gunzip

```

You can also use the `-u` flag to output per-user statistics:

```

# sa -u
root      0.01 cpu      344k mem      0 io which
root      0.00 cpu     1094k mem     0 io bash
root      0.07 cpu     1434k mem     0 io rpmq
andrew    0.02 cpu      342k mem     0 io id
andrew    0.00 cpu      526k mem     0 io bash
andrew    0.01 cpu      526k mem     0 io bash
andrew    0.03 cpu      378k mem     0 io grep
andrew    0.01 cpu      354k mem     0 io id
andrew    0.01 cpu      526k mem     0 io bash
andrew    0.00 cpu      340k mem     0 io hostname

```

You can peruse the output of these commands every so often to look for suspicious activity, such as increases in CPU usage or commands that are known to be used for mischief.