

Advanced Log Processing

Anton Chuvakin 2002-08-01

Advanced Log Processing

by Anton Chuvakin, Ph.D, GCIA

last updated August 1, 2002

One of Murphy's laws advises to "only look for those problems that you know how to solve." In security, this means to only monitor for those attacks that you plan to respond to. It is well known that any intrusion detection system is only as good as the analyst watching its output. Thus, having nobody watching the IDS is equivalent to having no IDS at all. But what should an IDS administrator do if he or she is drowning in a flood of alerts, logs, messages and other attention grabbers?

Logs serve to assure that everything is working as it should be, and to help figure out what malfunctions or security incidents may have occurred. Having logs from multiple machines collected in one place simplifies both day-to-day maintenance and incident response. The advantages of a centralized log collection site include more effective auditing, secure storage of logs, and the increased opportunity for analysis across multiple platforms. In addition, secure and uniform log storage might be helpful in case an intruder is prosecuted based on log evidence, in which case, careful documentation of log handling procedure might be needed.

This article will offer a brief overview of log collection and analysis. It will discuss using logs in incident response and handling logs in day-to-day routine. Specifically, it will look at three fundamental problems: log transmission, log collection and log analysis. It will also briefly touch upon log storing and archival.

Log Transmission

First, let's look at log transmission. The tradition mechanism for UNIX log transfers is UDP, port 514 (see [RFC 3164](#) for more details) . Log messages are sent and received by a syslog daemon. Network security devices (not only UNIX based) also often use UDP for logging. What are the evident problems with this approach? Messages can be injected, quietly dropped (or replaced) or delayed in transit. Furthermore, there is no delivery confirmation or encryption. With these concerns in mind, it becomes clear that logging over UDP is unsuitable for high security environments, unless a separate LAN is used for collecting the logging information.

What are alternative channels for log transmission? First, there is a standard for a reliable syslog transmission ([RFC 3195](#)), but no major vendors currently implement it. The simplest approach is to accumulate logs locally and periodically copy them to an aggregation server via SSH-based secure copy (SCP). There are several methods available to automate the process, such as the following:

1. rotate the logs - logrotate
2. compress them - gzip
3. apply the checksum algorithm (e.g. md5sum) - md5sum
4. copy the logs from host to aggregation server - scp
5. run the md5sum again and compare
6. store the log files and the checksums in a secure place (maybe encrypted)

(For more details on this transfer method, see Kristy Westphal's SecurityFocus article [Secure Remote Log](#)

[Servers Using SCP](#).) However, the evident flaw of this method is the fact that it is time-delayed. Unlike the UDP-based syslog, this log copying method allows for a lag time between the log generation and safe storage and analysis. Sometimes it is critical to see the log files immediately.

One way to facilitate instant access to logs is tunneling. Some say this is inelegant but it works. Using [Netcat](#) one can tunnel UDP over Secure Shell by redirecting the syslog traffic to TCP tunnel, protected by Secure Shell. The directions can be found at <http://www.patoche.org/LTT/security/00000118.html> Make absolutely sure that the syslog is not receiving messages from other hosts, or message looping will occur.

In fact, by replacing Netcat with [Cryptcat](#), one can eliminate SSH from the equation. In this case the setup is as follows:

On log-generating host:

1. edit /etc/syslog.conf to have:

```
*.* @localhost
```

2. run command:

```
# nc -l -u -p 514 | cryptcat 10.2.1.1 9999
```

On log collecting host:

1. run syslog with remote reception (-r) flag (for Linux)

2. run command:

```
# cryptcat -l -p 9999 | nc -u localhost 514
```

Another option is that Stunnel SSL wrapper can be used in place of Cryptcat (this document from the [Centralized Syslog Loghost Project](#) has details.)

If one is not satisfied with makeshift tunneling solutions, or requires even more security (such as even higher delivery guarantee or cryptographic log integrity verification), it may be time to consider syslog replacements. We will look at two well-known replacements: [syslog-ng](#) by Balabit and [msyslog](#) by CORE SDI. (A third alternative, [nsyslog](#) by Darren Reed, does not appear to be actively updated anymore.) Their common features include TCP communication, more filtering options (in addition to SEVERITY and FACILITY of standard syslog), and log file integrity support.

msyslog

Let us look at configuring msyslog for production environment. Myslog-1.08a-1 is installed on the client machine (which produces logs) and the server machine (which collects logs) from RPM packages. Both client and server run RedHat Linux 7.2.

The TCP-mode setup that worked involves the following list of changing on the hosts:

On the client:

1. Modify /etc/syslog.conf to have

```
*.*                %tcp -a -h loghost -p 514 -m 30 -s 8192
```

IN PLACE OF

```
*.*                @loghost
```

2. Run msyslog as `msyslogd -i linux -i unix`. Just running `/etc/init.d/msyslog start` does the trick.

On the server:

1. Run msyslog as `msyslogd -i linux -i unix -i 'tcp -a -p 514'` This can be accomplished by modifying `/etc/syslog.conf/msyslog` to have:

```
IM_LINUX="-i linux"           # example: "-i linux"
IM_TCP="-t tcp -a -p 514"     # example: "-i tcp accepted.host.com 514"
IM_UNIX="-i unix"            # example: "-i unix"
```

The result is an unencrypted TCP connection (which can be verified by `tcpdump proto TCP and port 514`, one would see the log messages)

Let's add hashing to the mix. On the syslog aggregation server, where the messages are distributed between various log files, one needs to:

1. Add a line to `syslog.conf`:

```
*.info;mail.none;authpriv.none;cron.none %peo -l -k
/etc/.var.log.authlog.key %classic /var/log/messages
```

2. Stop the syslog daemon and rotate or erase the log files

3. Run the program to create the initial checksum (using the included "peochk" program):

```
# peochk -g -k /etc/.var.log.authlog.key
```

5. Start the syslog: `/etc/init.d/msyslog start`

To test the functionality run:

```
peochk -f /var/log/messages -k /etc/.var.log.authlog.key
```

to see:

```
(0) /var/log/messages file is ok
```

on the intact log file.

If one is to edit the "messages" file (say, by removing a line) and then retest, the output would be:

```
(1) /var/log/messages corrupted
```

The advantages of msyslog include that it uses the same /etc/syslog.conf file as regular syslog, full syslog interoperability in UDP mode (as client and server) and extensive regular expression support, that allows matching messages with various source, time and content fields. The version that was evaluated also uses easy-to-use RedHat-style /etc/sysconfig file to control daemon start-up options.

The options include various modules that handle I/O, such as receiver TCP/UDP, send TCP/UDP, send to database, record cryptographic has and others. Msyslog also features a debug mode that was very helpful during the setup.

The most secure setup will involve binding msyslog to a localhost address (127.0.0.1) and using SSH RSA/DSA authentication for access control, together with hash integrity checking on the logging server. Another important issue is buffering. Since TCP offers reliable delivery (unlike UDP), some measures should be taken to keep the log files in case the log server goes down. Msyslog offers configurable buffering option. In the above configuration: '-m 30 -s 8192' stand for retry limit (in seconds) and buffer size (in message lines). Buffers are also important for dealing with message bursts, that do happen, for example, when programs are starting up or when a "noisy" firewall is getting port scanned.

Syslog-ng

Syslog-ng (version 1.5.17-1) was installed from RPM packages on the same test systems. Syslog-ng supports TCP connections, filtering based on message contents, logging of complete chain of forwarding loghosts (unlike regular syslog which will only record the name of last step), and so on. Extensive documentation is available.

To make syslog-ng work, one has to use an included conversion tool to convert the /etc/syslog.conf to syslog-ng format file. The command:

```
# /usr/share/doc/syslog-ng-1.5.17/syslog2ng < /etc/syslog.conf > syslog-ng.conf
```

does the trick. The excerpt is shown below:

```
# global options
options { use_dns(yes);
          use_fqdn(no);
          use_time_recvd(no);
          chain_hostnames(no);
          mark(0);
          sync(0);
};

source s_local { internal();
                 unix-stream("/dev/log" keep-alive(yes) max-connections(10));
                 file("/proc/kmsg");
};
```

```

# *.*
destination d_2 {
    tcp("anton" port(514));
};

filter f_5 {
    level(debug...emerg);
};

log { source(s_local); filter(f_5); destination(d_2); };

```

It is easy to follow the logic, even though the file is different from regular syslog. However, writing files by hand and using advanced options will take some learning. For example, to enable TCP logging one has to replace UDP with TCP in the configuration file (done in the above example: see 'tcp("anton" port(514));'). Again, by default the communication is not encrypted.

Syslog-ng also features more granular access control and can use TCP wrapper to limit network access. The program can also redirect messages to custom programs for real-time processing. For example, to send every log message to the STDIN on the "correlate.sh" script, add to the config file:

```

log { source(s_local); destination(d_prg); };
destination d_prg { program("/home/bin/correlate.sh -a"); };

```

The first test performed was the interoperability test: syslog-ng client successfully sent messages over UDP and TCP to msyslog server.

Syslog-ng comes with a stress test tool (that calls the "/usr/bin/logger" command in a large loop). However, it is apparent that TCP transfers will be slower than UDP, even with no encryption. The syslog replacements should be stress tested (at well above normal message rates) before enterprise deployment. It should be noted, that for conventional syslog UDP transmission the failure mode will be losing messages, it is not clear how the TCP-enabled daemons will behave.

Overall, msyslog and syslog-ng are viable options where extra security is desired. However, a detailed testing is required before deployment.

A few words on covert logging: if one is running a honeypot (like we do) and is experiencing intense paranoia about attackers detecting your system and keyboard log transfers, some covert options are available. Encrypted spoofed UDP transfer mechanism has been proposed for this purpose. However, that discussion is beyond the scope of this paper.

Log Collection

The typical method of log collection is to have a dedicated logging host holding the log records from many machines in a single mammoth file that is rotated and compressed periodically. This method has been used since the early days of UNIX and it has few disadvantages.

Logging to a database brings us to the next level of log aggregation. Msyslog has native support for logging to a database (MySQL and Postgres). To configure, do the following on the log collecting server:

0. Install and start mysql

```
# /etc/init.d/mysql start
```

1. Create a database instance:

```
# echo "CREATE DATABASE msyslog;" | mysql -u root -p
```

2. Define tables for log storage:

```
# cat syslog-sql.sql | mysql msyslog
```

The file is shown below:

```
CREATE TABLE syslogTB (  
    facility char(10),  
    priority char(10),  
    date date,  
    time time,  
    host varchar(128),  
    message text,  
    seq int unsigned auto_increment primary key  
);
```

3. Edit syslog.conf to enable database-logging module:

```
*.*      %mysql -s localhost -u snort -d msyslog -t syslogTB
```

4. Grant access privileges for message insertion:

```
# echo "grant INSERT,SELECT on msyslog.* to snort@localhost;" | mysql -u root -p
```

5. Restart msyslog

```
# /etc/init.d/msyslogd start
```

The following image shows how the result looks in PHPAdmin.

Home

msyslog (1)

msyslog
syslogTB

Database *msyslog* - table *syslogTB* running on *localhost*

Showing rows 0 - 29 (69676 total)

SQL-query : [Edit]
SELECT * FROM `syslogTB` LIMIT 0, 30

Show : 30 rows starting from 30

in horizontal mode and repeat headers after 100 cells

←T→		facility	priority	date	time	host	message
Edit	Delete	NULL	NULL	2002-05-17	10:51:19	anton. [REDACTED]	syslogd: restart
Edit	Delete	NULL	NULL	2002-05-17	10:51:39	anton. [REDACTED]	syslogd: exiting on signal 1
Edit	Delete	NULL	NULL	2002-05-17	10:51:41	anton. [REDACTED]	syslogd: restart
Edit	Delete	NULL	NULL	2002-05-17	10:51:43	anton. [REDACTED]	syslogd: exiting on signal 1
Edit	Delete	NULL	NULL	2002-05-17	11:37:56	anton. [REDACTED]	syslogd: restart
Edit	Delete	NULL	NULL	2002-05-17	10:30:50	sans	sansdemo syslogd: restart
Edit	Delete	NULL	NULL	2002-05-17	10:30:50	sans	sansdemo msyslog: msyslogd startup succeeded
Edit	Delete	NULL	NULL	2002-05-17	10:30:31	sans	sansdemo msyslog: msyslogd shutdown succeeded

In this setup, message queuing is enabled on the client (which uses the configuration shown above). In some cases, it was necessary to restart the client after the server restart if using the TCP mode.

Other tools to collect syslog messages include [SQLSyslogd](#), which can be used with regular syslog daemon.

Collecting logs in the database presents several important advantages over plain text storage. Databases can be set to accept messages at much higher rates. In the tests, for simple messages msyslog-MySQL combination received and archived about 240 messages per second. A sustained rate of thousands of messages per seconds is not unheard of for a commercial log aggregation software, which can also analyze the resulting massive datasets. If proper analysis software is available, log file database can be used to analyze the data more effectively. For example, fine-grained searches can be performed (as will be shown below).

Log Analysis

Now let's turn to the analysis part. Log analysis often is defined as getting meaningful intrusion data and some historical trends from log file. In fact, a lot of good software is written to analyze plain text log files (a large resource list is available at [Counterpane's Log Analysis Resource](#)). Not having the space to review all the log analysis scripts, it makes sense to review the approaches and then formulate suggestions on log analysis.

Log analysis can be split into real-time and periodic analysis. Tools like "swatch" or "logsurfer" provide real-time log processing and "logcheck", "logwatch" and many others use the periodic approach.

Both approaches work, provided you know what to look for and can write some sophisticated regular expressions to tell the grains from the chaff. Some of the analysis scripts (such as Logcheck come with an extensive set of default regexes), while others (such as Swatch) make you create your own. The techniques are well covered in their documentations.

One tool deserves special mention for having a much more sophisticated real-time analysis and correlation engine. [SEC \(simple event correlator\)](#) by Risto Vaarandi offers not just regex matching on a line by line basis, it also offers an extensive list of sophisticated multi-event matches such as match an event A and wait for X seconds for the event B to arrive, then execute an action, match an event A, then count same events for X seconds and execute an action if threshold is exceeded, etc. In addition, events can be matched across multiple lines. Provided, that you know what to look for, the program can be an extremely powerful tool for log analysis.

But what if the logs are stored into a database? What techniques can one use to analyze those? Instead of running a script one can write a simple (or, not so simple, if desired) SQL file to look through events and establish relationships.

To analyze logs, one might want to run various SELECT queries, such as the following (the msyslog database described above is used for tests):

High-Level Overview Queries:

```
-----
| number of events                               | select count(*)
from syslogTB; |
-----
| number of hosts that sent messages in          | select count( distinct host)
from syslogTB; |
-----
| number of messages per host                    | select host, count(host)
from syslogTB |
|                                               | group by host order by
host desc; |
-----
```

Sample output from MySQL:

Host	Count(host)
box1	20
box2.example.com	3147

Drill-down detailed reports:

```
-----
| Search Type                                     |
SQL String |
-----
| search by hostname                             | select * from syslogTB where host like
"%box1%" |
-----
| search by message text                         | select * from syslogTB where message like "%
restart%"; |
-----
| search by combination                          | select * from syslogTB where message like "%
```

```

restart%" |
|           and host
like "box1";
|           | select * from syslogTB where message like "%
restart%" |
|           and time
like "10:51%";
|

```

Sample output from MySQL:

facility	priority	date	time	host	message	seq
NULL	NULL	2002-05-17	10:51:19	box2.example.com	syslogd: restart	1
NULL	NULL	2002-05-17	10:51:41	box2.example.com	syslogd: restart	3

count the number of events `select count(*) from syslogTB where message like "%restart%";` see all unique message types `select distinct message from syslogTB;`

One's imagination is the only limit, as SQL syntax is very flexible and allows extremely complicated queries to be built. Just keep in mind that with a loaded database the multi-message queries might take a while (however, still much faster than doing a 'grep' on a mammoth plain text file).

Conclusion

As a conclusion, several best practices for system logging should be followed:

- Use a dedicated logging host
- Make sure that tight access controls are enabled on all logging servers
- Encrypt logs if business case for this exists
- Try to log to more than one box to increase reliability
- Watch for overflowing log partitions/storage
- For more security store logs on WORM media (if business case exists) or transfer them to a non-networked computer

Clearly, not all of the above need to be implemented for all environments. The lists provides some of the things that will ensure that one will be able track an incident when it occurs.

Anton Chuvakin, Ph.D. is a Senior Security Analyst with [netForensics](#), a security information management company that provides real-time network security monitoring solutions.

[Privacy Statement](#)

Copyright 2006, SecurityFocus