Making sure your logs work

SECURE LOGGING

Might as well do it properly – rsyslog. **BY KURT SEIFRIED**

I was looking at my server backups the other day and suddenly realized I had no backups of the logfiles – you know, all the stuff in */var/log/* that you ignore until something breaks. Also, I realized I had no idea whether logging was actually working, so maybe I should go about fixing this.

Why Logging Locally Isn't Secure or Reliable

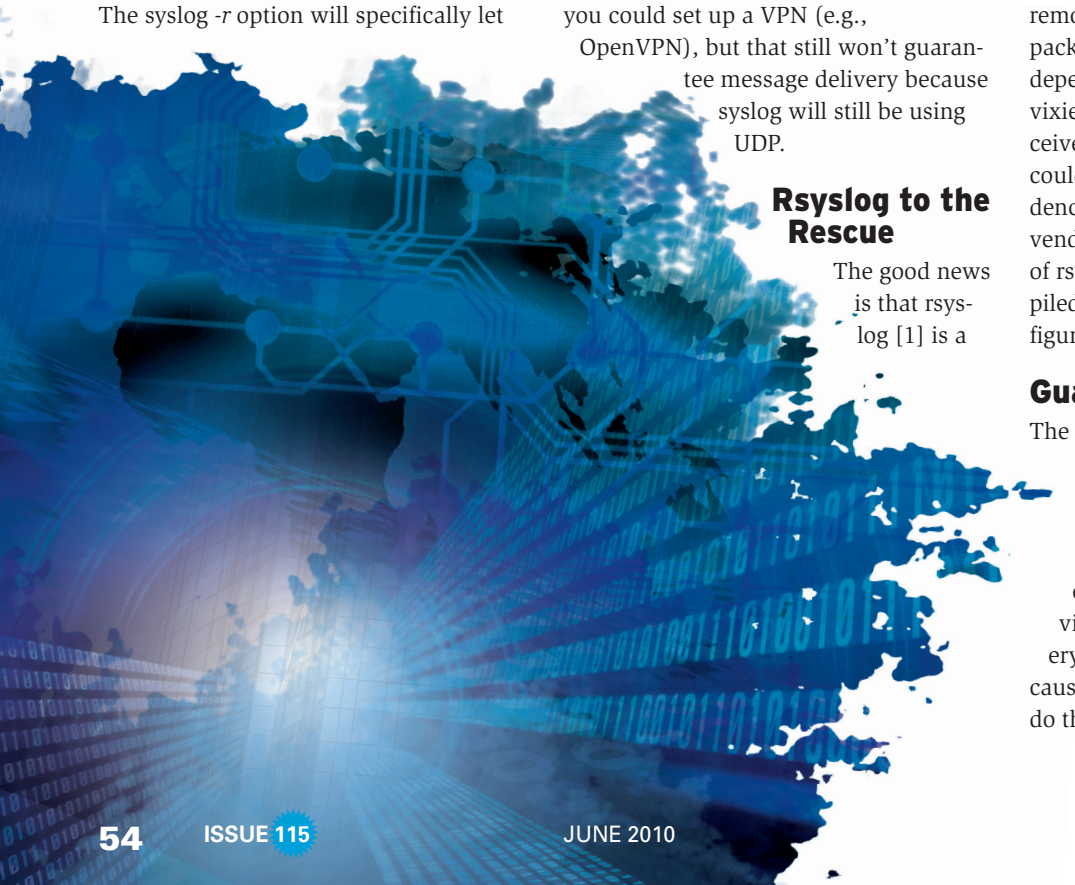
By definition, attackers who gain access to your system can monkey around with the logfiles, modifying them or simply deleting them altogether if they're not worried about being quiet. This can leave you with no log of how they broke in or what they did, or even that you have suffered a break in. Also, your logging can simply break and stop writing logs, and, unless you're checking those logs, you'll never notice.

This old problem has a simple solution: You just log to a remote host that is designed to hold the logfiles securely. The syslog *-r* option will specifically let syslog listen to the network and accept syslog messages from remote hosts. Problem solved, right?

Not really. The main problem is that syslog uses UDP for message transport, and UDP doesn't guarantee delivery. So, for example, if you accidentally firewall syslog, the messages could simply be dropped without any warning. If you want to send logs across networks you don't trust (e.g., the Internet), you can't be sure that an attacker hasn't injected log messages, because spoofing UDP is much easier than TCP in that no three-way handshake or sequence ID is needed. Also, if an attacker is really on the ball, he or she can alter messages in transit, without being detected (the messages are not protected in any way).

Why Not SSL Wrap It?

Trick question. Applications like Stunnel only play nicely with TCP-based services. If you're utterly determined not to run rsyslog for some reason, I suppose you could set up a VPN (e.g., OpenVPN), but that still won't guarantee message delivery because syslog will still be using UDP.



Rsyslog to the Rescue

The good news is that rsyslog [1] is a drop-in replacement for sysklogd, and if you're running a recent Debian, Ubuntu, or Fedora, it's the default logging package. The bad news is that you will most likely need to upgrade it if you want the more advanced features. (Fedora ships version 2.0.6; the latest stable release is 5.2.0.)

For those of you on a different flavor of Linux, chances are your vendor ships rsyslog, and you can simply install it with yum or whatever package manager you use (e.g., *emerge -va rsyslog* on Gentoo Linux). Downloading the source and compiling it is not hard; however, you should make sure that you add support for:

```
./configure --enable-gnutls ^^
   --enable-mysql
make
make install
```

Of course, you will need the gnutls-devel and mysql-devel packages installed on your system for this to work. Once compiled, you need to install rsyslog. This can be tricky because you will need to remove the existing rsyslog or sysklogd package forcibly and ignore the resulting dependency complaints (initscripts, vixie-cron, cronie, etc.) that you will receive. Future updates of these packages could also be an issue because a dependency is missing. Unfortunately, most vendors are shipping very old versions of rsyslog. Once you have rsyslog compiled and installed, you will need to configure it securely.

Guaranteed Delivery

The first thing rsyslog does right is use TCP, which is a much more reliable transport than UDP. The second thing rsyslog does is provide application-level acknowledgment of messages. Thus, it provides a guarantee of message delivery, so even strange TCP errors won't cause messages to disappear silently. To do this, it supports RELP (Reliable Event

Logging Protocol); configuration is trivial on a client:

```
*.* :omrelp:10.1.2.3:2514
```

So, the first goal of ensuring that remote logging actually results in remote logging taking place is accomplished. Another advantage is that a central rsyslog server will correctly report the origin of a message even if the sender is behind a NAT machine with other rsyslog clients (in other words, you will be able to tell them apart, which doesn't work so well with other logging packages).

Secure Delivery

Secure delivery is essential; otherwise, an attacker can modify messages in transit or inject fake messages and cause all sorts of problems (like hard disk write error warnings every day at 3am that cause your pager to go off). Also, you don't want an attacker eavesdropping on messages. For example, if a user accidentally enters her password instead of her username, that password will be logged (and sent to whatever remote systems your logs go to), potentially exposing the password to an attacker with access to your network traffic. To address this, rsyslog supports TLS (Transport Layer Security) natively; all you need to do is edit *rsyslog.conf* [2].

This brings up about the only flaw I can find with rsyslog: the documentation. Generally speaking, the documentation is good, but the examples are not always the best. In this case, the primary example shows the *$InputTCPServerStreamDriverAuthMode* variable set to "*anon*", meaning no client authentication is taking place, which sort of defeats the whole point of using TLS to secure communications. If you look in the gtls Network Stream Driver documentation file, you will see that the mode you want to use is *x509/name*. This setting will cause rsyslog to validate the certificate and the name before allowing the client to communicate. Also, you should configure this on the clients to ensure that an attacker can't execute a man-in-the-middle attack and impersonate a server (and thus harvest potentially sensitive log messages).

Note that support for the *x509/name* configuration directive only appeared in version 3.19.4 and later, so you'll need to update Fedora (rsyslog 2.0.6), Debian (rsyslog 3.18.6), and Ubuntu 9.0.4 (rsyslog 3.18.6), but not Ubuntu 9.10 (rsyslog 4.2.0), to name a few versions of Linux.

Listing 1: Off-Peak Message Delivery

```
01 # reliably transmit messages
02 # during off-peak hours (10p to 4a)
03 $ModLoad omrelp
04 $WorkDirectory /rsyslog/work #
   where to place the spool files?
05 $ActionQueueType LinkedList
06 $ActionQueueDequeueTimeBegin 22
07 $ActionQueueDequeueTimeEnd 4
08 $ActionQueueFileName relpact
09 $ActionQueueSaveOnShutdown on
10 *.* :omrelp:10.1.2.3:2514
```

Off-Peak Message Delivery

Real-time remote logging has one problem, which is that network traffic will be steady and in some cases quite heavy (someone scans servers for weak accounts, a sudden slew of errors, etc.). If you have branch offices or remote locations that you want to tie into a central logging system, you could find yourself taking up a significant portion of upload bandwidth during business hours (not all the world has high-speed Internet like Japan and Norway yet). Fortunately, rsyslog addresses this with off-peak message delivery (Listing 1).

This will send log data to 10.1.2.3 between 10pm and 4am; otherwise, it will spool the logfile locally for later transmissions. The *ActionQueueSaveOnShutdown* is important; without it, you will lose data if you shut down rsyslog, because it will not write data in memory to the spool. The other benefit of off-peak message delivery is that you can stagger delivery times for servers so your central logging server doesn't get flooded by clients [3].

Transitioning to Rsyslog

What if you want to start using rsyslog, but you have older syslog clients that you can't yet upgrade? That's easy; run a central rsyslog server with support for UDP messages and upgrade the clients when you can. On the central rsyslog server, you simply:

```
@ModLoad imudp
$InputUDPServerRun 514
```

Because rsyslog is a module and supports multiple inputs (and outputs), you can easily run it with support for multiple client types (UDP, TCP, RELP, etc.).

I'm Stuck with an Older Rsyslog

If you are truly stuck with an older rsyslog, the good news is that you can at least use Stunnel to SSL wrap it. Red Hat has a knowledge base article with detailed instructions [4]. The process is pretty much like wrapping any other service: You set up Stunnel on the server to accept connections, and on the client, you connect to the server and configure a local port that is forwarded to the server. Rsyslog on the client machine connects to the local port that sends the data off to the server.

In Conclusion

The rsyslog package is a lot more reliable and secure than syslog or sysklogd. Additionally, you can log to a database, send SNMP alerts, and browse events with a nice web interface [5] [6]. ■

INFO

[1] Rsyslog: *http://www.rsyslog.com/*

[2] Encrypting Syslog Traffic with TLS (SSL): *http://www.rsyslog.com/doc-rsyslog_tls.html*

[3] Delivery during off-peak hours: *http://wiki.rsyslog.com/index.php/OffPeakHours*

[4] Wrapping rsyslog with Stunnel: *http://kbase.redhat.com/faq/docs/DOC-18564*

[5] "The sys admin's daily grind: RSyslog" by Charly Kühnast, *Linux Magazine* June 2008, pg. 63, *http://www.linux-magazine.com/Issues/2008/91/WHERE-TO-NEXT*

[6] "The sys admin's daily grind: phpLogCon" by Charly Kühnast, *Linux Magazine*, July 2008, pg. 69, *http://www.linux-magazine.com/Issues/2008/92/MILKING-MACHINE-2.0*

THE AUTHOR

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.