

Secure Remote Log Servers Using SCP

Kristy Westphal 2001-02-14

Secure Remote Log Servers Using SCP

by *Kristy Westphal*

last updated Feb. 14, 2001

Introduction

A few months ago, a problem was presented. It became a necessity to implement a centralized system log server that would securely store logs. The design needed to provide a level of security that would prevent tampering or mischief, while preserving integrity. . It was necessary to find a solution that fit into my company's tight budget that would also be a) secure, b) affordable and c) easy to run, especially on a Solaris system. While these constraints made it difficult to discover a viable solution, I was nevertheless able to do so. This article will discuss a solution that meets these criteria and will work well in other environments as well. It should be noted that since I implemented the solution I have in place now, I have discovered some other options.

Why a remote logging solution?

You may ask yourself the question: why would I need a remote logging solution? Centralizing system log files can have several important advantages. Some advantages include: centralized management of log files, maximized disk space usage, easier access for auditing purposes and a more secure method of retention. As well, adding encryption and checksumming on top of a remote logging server adds yet another layer of security on these log files, which is always an advantage.

However, as with many things in life, there are disadvantages to centralizing your log files, as well. As I shall explain later in this article, the set-up of such a system can be rather time-consuming and somewhat difficult. (The good news about this is that once you have the system in place, it pretty much manages itself!) Another disadvantage of this type of system is that you may end up backing up files that have already been tampered with. This system can also be quite bulky, requiring a large partition of disk to store the files. Finally, if you set up the remote logging system without additional security, such as encryption and checksumming mentioned in the preceding paragraph, data will be transferred in plaintext and may be easily intercepted.

The secure logging solution that I will discuss in this article is simple in nature, using OpenSSH's version of SCP (secure copy), the md5 message digest algorithm, a few homegrown scripts, cron, and a little planning. This logging system can be a secure storage facility for your critical server logs files. What cannot be is an online server that gathers log entries from other servers in realtime. In this remote logging system that I am proposing, the current logs are still kept on the servers from which they

originate, they are then moved through SCP to the central server at certain times throughout the day.

Designing your remote log system

The first step in implementing your remote log system is to design the layout of the servers. It is necessary to select a secure server to store these files on. You may think that storing these logs files in a secure fashion on the server is sufficient to secure your files, even if your server is compromised. However, a key idea behind this concept is to use a server that has been already been locked down, thereby adding another layer of security to thwart any attempt to tamper with your log files. In my case, I had to either pick a server that had been locked down and wasn't being used for any other services, or designate one to be locked down, starting with reinstalling the OS.

The next step is to decide where you want the log files to reside. There are various directory structures that I could have gone with, but I wanted to make it easy to find what I wanted when I wanted it. In my system, I set up a partition called "logs". Within the logs partition, I have a separate directory for each of my servers. Using the scripts that I created, I made a subdirectory under the server directory with the current date. The log files for that day are then copied into this directory by name.

The next issue that I considered was how often I actually wanted to have the files sent by SCP to my log server. This was tough to decide, but I looked to my previously-established security policy to determine the appropriate schedule. It should be noted that the rotation schedule may also vary according to the server: for instance, if a server is heavily used, then you may want to copy files over more frequently. If you do this, you may want to create subdirectories under the date directories with hour timestamps on them. I decided that it would be adequate to have files copied over once per day. The permissions on all of these directories should be 700, owned by root.

Once I had the basics set up, I had to take another hard look at the security on this server. This server would end up with an intricate trust relationship with the other servers on my network, so it needed to be secured in such a fashion that only the logging mechanism, and nothing else, would be able to run. In my case, I chose an existing server, with up-to-date patches. It was then a matter of combing through the `/etc/inetd.conf` (and the corresponding `rc` directories for their startup scripts) to shut down all services not needed on the server.

In order to ensure the security of the server, I even went so far as to run my Nessus vulnerability scanner on it to see if there were other holes I needed to look for (please note: one vulnerability scanner does not ensure the security of a server; however, it does help to point you in directions that perhaps you hadn't thought of. A best practice might be to run more than one scanner against your servers. Unfortunately, this topic is beyond the scope of this article.) There are numerous well-written articles on how to secure Solaris servers that I recommend you refer to in order to fully secure your log server.

Setting up the remote log server

With the log server well secured, and the directory structure set up, I had to set up SSH on each server for which I would be getting logs. This was a good time to get it installed on all the servers as a replacement for telnet. However, prior to installing SSH, there are two additional packages you will need to install: [zlib](#) and [OpenSSL](#). I found that compiling these were fairly straightforward, as long as I installed them in this particular order. Once this is done, it is necessary to download the latest version of [OpenSSH](#). (latest release, at time of publication of this article, is 2.3.0.) The ported version of [OpenSSH](#) is the software package necessary for implementation on a Solaris system. The standard distribution of OpenSSH is written specifically for OpenBSD, and will not compile on Solaris systems. Two options that I found useful while running the SSH configure script were:

1. with-tcp-wrappers (as I was using wrappers) and;
2. with-SSL-dir=DIR (to point OpenSSH in the right direction to find OpenSSL).

Other options can be found in the installation file.

A closer look at SSH

Upon installation, SSH-keygen is executed. SSH-keygen builds both the public and private keys for the host, type of key dependent upon version of SSH. The 1.x series of SSH generates RSA Keys, while the all others use DSA. Recently, there have been problems with the SSH protocol discussed in various public forums. Man-in-the-middle attacks, as well as the release of dsniff and other vulnerabilities in the SSH 1.x protocol have introduced new risks that must be taken seriously.

This type of attack can occur easily in a SSH environment, if you don't take some precautions. The first step would be to utilize SSH 2.0, which offers enhanced security, and reconciles the shortcomings of version 1 of the protocol. The second solution would be to avoid exchanging public keys with your servers or people over the network for the initial negotiation. [1] The best way to do this would be to copy the key created in `~/.SSH/id_dsa.pub` to tape or floppy, then walk to the actual log server and copy it to its `~/.SSH/authorized_keys2` file (remember to append, not overwrite!). Sneakernet may be the oldest way in the world to deliver information, but sometimes, it is still the safest! Finally, you should continually audit your `known_hosts` file, double-checking public keys for accuracy. [2]

Having said this, I copied root's `id_dsa.pub` key into all the participating server's `~/.SSH/authorized_keys2` file (by hand!). Then I changed permissions for the file to be 600. Due to this trust relationship that I had established, my loghost could then log into each of these servers without the use of a password. Notice that you do not want the opposite affect: there is no need for your main log server to have the public keys of your participating servers. This will defeat the purpose of your secure server!

Back to the setup

Now that I had my SSH environment set up properly, I needed one more tool on all of my servers: the [md5 message digest algorithm](#). MD5 will help to verify that any data that was sent by the server actually made it to our log server without interference by putting a checksum on the file before and after SCPing it over. (For those who do not know, SCP is the replacement for FTP and RCP so that files can be sent securely over a network wire.)

Now with md5 on all my servers, the scripting shall begin. But wait! The scripts are actually the easy part. On each individual server, I run a script that makes an md5 signature for all the files I will be copying to the log server (in my case, all files in /var/log). I put this file in my already locked down /.SSH directory, with 600 permissions. Back on the log server, I created one script to perform the following tasks:

1. create the server directories,
2. SCP the log files to the loghost,
3. compress the log files; and,
4. run an md5 checksum against each, putting the results in a separate file that resides in the root of the /logs directory.

The following is an example of what your script might look like:

```
#!/bin/sh
#
# This script will create some directories to throw logs into.
#
dirdate=`date "+%m-%d-%y" `
md5=/usr/local/bin/md5
gzip=/usr/local/bin/gzip
awk=/usr/bin/awk

cd /logs
touch $dirdate
chmod 600 $dirdate
cd /logs/myserver
mkdir $dirdate
cd /logs/myserver2
mkdir $dirdate

sleep 30

[snip]

#
```

```

# This part grabs the log files

/usr/local/bin/SCP root@myserver:/var/log/authlog
/logs/myserver/$dirdate/authlog

/usr/local/bin/SCP root@myserver:/var/log/syslog
/logs/myserver/$dirdate/syslog

/usr/local/bin/SCP root@myserver:/var/adm/messages
/logs/myserver/$dirdate/messages

/usr/local/bin/SCP root@myserver:/var/adm/lastlog
/logs/myserver/$dirdate/lastlog

/usr/local/bin/SCP root@myserver:/var/adm/wtmpx /logs/myserver/$dirdate/wtmpx

/usr/local/bin/SCP root@myserver:/var/adm/utmpx /logs/myserver/$dirdate/utmpx

/usr/local/bin/SCP root@myserver:/var/adm/loginlog
/logs/myserver/$dirdate/loginlog

sleep 30

[snip]

# This part will compress and md5 our secure logs

cd /logs/myserver/$dirdate
$gzip *
echo '*****myserver*****' >> /logs/$dirdate
for i in `ls -l | $awk '{print $9}'`
do
$md5 $i >> /logs/$dirdate
done

exit

```

The last step was to cron this script to run at the predetermined intervals. And lastly, I needed to test the script to make sure that it does everything it is supposed to!

Archiving and retention policies

The final step that I needed to take in this process, was to decide how long I wanted to keep the log files, and where I would store files that I no longer needed on disk. Again, referring to the existing security policy helped a great deal. In my case, I decided to keep the files on-line for one month, then archive to tapes that are stored in a secured place.

Other resources and options

As I mentioned in the introductory section of this discussion, the solution that I implemented is not the only solution for setting up a secure logging server. There are a few other possibilities on the horizon. For those of you who are brave enough to replace your standard version of syslogd: take a look at [syslog-ng](#), the next generation. This is a drop-in replacement for syslogd (works on many platforms besides Solaris), created by BalaBit IT Ltd. It is intended to be more easily configurable and tunable than regular syslogd, and promises encrypted files in a future release.

Another promising replacement for syslogd is [nsyslog](#) written by Darren Reed. It runs on TCP and over SSL when sending data over the network. My implementation was completed prior to discovery of this software package. This package has not been evaluated. Therefore, I can currently offer no insight on its features and performance.

Another possibility, that does not require replacing your syslogd daemon, is to set up a separate network, with all your servers having at least two NICs so that one can be on the private logging network, and one on the public/intranet network. This option is pretty pricey, and may not be easily implementable in some situations.

Finally, on the more distant horizon, lies the [Security Issues in Network Event Logging](#) working group, a part of Internet Engineering Task Force (IETF) committee. The objective of this group is to:

"document and address the security and integrity problems of the existing Syslog mechanism. In order to accomplish this task [they] will document the existing protocol. The working group will also explore and develop a standard to address the security problems." [3]

They are developing three interesting projects - Syslog-Auth, Syslog-Sign and Syslog-reliable protocols - that may make syslog inherently more secure.

Conclusion

Currently, there are not many nice, neat solutions to making a secure syslog server. However, if you put some time and thought into developing something that is stable and secure, your auditing, forensic and security policies will be a lot more easily enforced than if you use syslog in its current default implementation. In addition, the more you can do to secure accurate log files, the more easily and effectively you will be able to respond to any sort of server compromise.

References

[1] ["The End of SSL and SSH?"](#), by Kurt Seifried, SecurityPortal.com, December 18, 2000.

[2] "[dsniff and SSH: Reports of My Demise are Greatly Exaggerated](http://sysadmin.oreilly.com/news/silverman_1200.html)", by Richard E. Silverman, http://sysadmin.oreilly.com/news/silverman_1200.html, December 22, 2000

[3] [IETF Syslog Working Group Home Page](http://www.ietf.org/html.charters/syslog-charter.html), <http://www.ietf.org/html.charters/syslog-charter.html>

Kristy Westphal is a versatile network administrator, skilled in troubleshooting and process analysis. Her experience in the IS field have allowed her to become knowledgeable in several flavors of UNIX and NT, as well as various aspects of information security and disaster recovery planning.

Relevant Links

[The End of SSL and SSH?](#)

Kurt Seifried, SecurityPortal.com

[Installing Solaris](#)

SecurityFocus.com

[Securing Solaris](#)

SecurityFocus.com

[Hardening Solaris: Creating a Diamond in the Rough Pt. I](#)

Hal Flynn, SecurityFocus.com

[Hardening Solaris: Creating a Diamond in the Rough Pt. II](#)

Hal Flynn, SecurityFocus.com

[Privacy Statement](#)

Copyright 2006, SecurityFocus