Global Information Assurance Certification Paper

GCUX Practical Assignment (v.1.9) (April 2002)

Syslog Hunting and Gathering

Dale V. Clark, Sep. 20, 2002

abstract

The syslog protocol has been implemented across most modern computing platforms, where it is widely used to log and route operating system and application event messages. The ability to configure syslog to route messages from remote hosts to a secure central loghost can provide important security advantages, but the resultant voluminous store of messages from diverse hosts and messagers makes prolematic the extraction of messages of particular importance originating from a subset of logging hosts. This paper describes a means for improving the gathering of syslog entries from only those hosts of interest and for then hunting among these sheperded entries for only those entries of particular importance.

introduction

The generation, collection, examination and archiving of syslogs is an important part of system administration and security, and at my site a central secured loghost has been established for this purpose. Hosts with syslog capabilities not only generate local syslogs, but are also configured to relay all syslog entries to this loghost, which host collects these entries and manages their archiving through a suite of home-grown archiving tools.

The collection of syslog entries from a multitude of diverse architectures on a central loghost spares the Information System Security Officer (ISSO) from having to visit each machine under his purview for syslog examination, and this relaying capability is an important feature of syslog. Aside from this convenience, the real time forwarding of syslog messages to a secure central store makes it far more difficult for an intruder to cover his tracks by altering syslogs on a compromised host, and is an important security feature.

This arrangement is not without its drawbacks, however. Consider, for instance, the multitude of applications that use syslog. For the developer, the syslog facility provides a standard, portable, and convenient logging facility, eliminating the need for platform-dependent proprietary logging systems. For the administrator, it provides a highly configurable, centralized log management tool, which in its simplest configuration can enable the routing of multitudinous log messages to a few central log files. For these reasons, syslog is widely used by most modern operating systems, by tools bundled with these operating systems, and by increasing numbers of third party applications.

Consider also how the typical application uses syslog. Messages may be logged at various priorities by these applications, ranging from top priority emergency messages, to the lowest priority debug messages, intended for use only during code development or installation. Like a pyramid, high priority messages are few and far between, while at the lowest level debugging messages may be emitted by seemingly every non-trivial line of code. Some applications, like named(1M), spew out enormous quantities of informational messages of low signal to noise ratio, of only occasional interest.

Finally, at my site we have a hundred or so machines relaying these messages to our central loghost. Due to storage considerations, each remote machine is configured to retain locally only those log entries deemed of particular interest. For security reasons, however, _all_ syslog entries of whatever priority and for all facilities (*.debug) are forwarded to loghost. Arriving at loghost, they are then postpended to one of only six logfiles, five of which collect messages generated by specific facilities (e.g., mail) with the sixth gathering the remainder.

The problem then, for an ISSO, is to extract from these voluminous logs only those entries pertaining to security, and from these only those originating from those platforms under his purview.

The objective of this paper is to develop a strategy for dealing with this problem, and to implement a solution. First, however, a quick look at the

underlying syslog tool.

syslog(3)

The syslog protocol was developed by Eric Allman at UCB as a general purpose logging facility, and is used widely by most modern operating systems and applications. Following is a brief exposition of the use of this tool insofar as it pertains to the subject of this paper. Man pages are available for the syslog daemon, syslogd(1M), the syslog configuration file, syslog.conf(4), and the various syslog library routines, syslog(3). Details of the underlying protocol are the subject of RFC3164, available at www.ietf.org/rfc/rfc3164.txt

Applications invoke syslog through the syslog(3) library routine, supplying (1) the text of the message, and (2) an encoding of the message priority and the type of facility generating the message. With this information, syslog(3) prepares a packet (sent to UDP port 514) consisting of three parts: the encoding of the priority/facility; message origin data (originating host and timestamp, inserted by syslog); and the message itself. Listening to port 514 is syslogd(1M), which routes this message according to syslog.conf(4).

Of these three principal parts, only syslog.conf is of particular importance to the problem at hand. Basically, this file consists of lines specifying actions for messages arriving with differing facility/encoding encodings. Actions of interest for the purposes of this paper are (1) appending to a file, and (2) relaying to a remote host. Facilities and priorities are defined in syslog.h, but are commonly given as follows (shown along with their Irix 6.5.12 one-character encodings):

priorities		facilities			
emerg	0	kern	A	<reserved>	M
alert	1	user	B	<reserved>	N
crit	2	mail	C	<reserved>	O
error	3	daemon	D	cron	P
warning	4	auth	E	local0	Q
notice	5	syslog	F	local1	R
info	6	lpr	G	local2	S
debug	7	news	H	local3	T
		uucp	I	local4	U
		<reserved>	J	local5	V
		<reserved>	K	local6	W
		<reserved>	L	local7	X

A quick example, showing two lines from a syslog.conf file similar to that in use at my site:

```
*.debug                                 @loghost.mysite.edu
auth.notice                             /var/adm/SYSLOG.auth
```

See the man page for syslog.conf(4) for details, but the essence of the above two lines is that ALL messages are to be forwarded to 'loghost' (the facility portion of the selector field is '*', signifying all facilities, and the priority portion of the selector field is 'debug', signifying all priorities equal to or greater than debug - they all are - are to be relayed to loghost), and that all facility=auth messages of priority 'notice' or greater are to be appended to the SYSLOG.auth file.

problem specifics

the gathering problem

In what follows we look at the specific problem of gathering at mysite's SGI loghost syslog messages generated by all eighteen Sun workstations at my site with these four requirements:

(1) Sun syslog messages are to be separated from messages originating from other hosts and routed to Sun-specific log files.
(2) Sun syslog messages generated by facilities of interest are to be further

separated from messages generated by other facilities and routed to facility-specific Sun log files.
(3) All such sheperded messages must retain the five parts of the original syslog packet:

- facility
- priority
- host name
- time stamp
- message

(4) No message to retain artifacts of the sheperding process.

meeting requirement (1)

The selector field of syslog.conf lines permits only the specification of facility.priority for routing syslog messages; no mechanism intrinsic to the syslog protocol provides for routing messages according to other criteria. There are, however, some facility values provided specifically for local use, and the approach taken to meeting this first requirement is to specify that messages arriving at loghost with facility=local3 will be recognized as having originated at a Sun, and then be routed to a Sun-only syslog on loghost.

A line such as the following in loghost's syslog.conf will accomplish this on the loghost end:

```
local3.debug                            /var/adm/SYSLOG.sun
```

The problem now becomes how, on the originating host end. to route all Sun syslog messages to loghost with the local facility changed to local3. Note that the choice of local3 was somewhat arbitrary, it just being one of the local[0-7] facilities not already in use.

The solution chosen was to re-syslog on the originating host all syslog messages in the following manner:

```
*.debug;local3.none                     /usr/local/adm/pkg/syslog/relay/relay
local3.debug                            @loghost.mysite.edu
*.notice;local3.none                    /var/adm/SYSLOG
```

The first line routes all messages (*.debug) to 'relay', a named pipe (FIFO) behind which sits a program which simply re-syslogs the received message with the facility changed to local3. The second line then forwards the message to loghost with the local3 facility, signaling that it originated on a Sun. The use of local3.none in the first line is necessary to prevent a logging loop. That is, a message "send help!" sent to the relay pipe would be resent endlessly to the pipe unless specifically excluded. The use of local3.none in the third line prevents the same message from being written twice to the local syslog; once with the original facility, then again with the message resent with the local3 facility.

Note that this solution to requirement (1) works also when a remote Sun must relay its messages through an intermediary host due to network topology. For example, consider a Sun console for a Cray supercomputer has a network interface to that Cray only. The above second line would be:

```
local3.debug                            @cray.mysite.edu
```

while the Cray syslog.conf would have a corresponding line:

```
local3.debug                            @loghost.mysite.edu
```

There could of course be a chain of intermediate but connecting hosts.

Pipe and reader are started/stopped by a startup script in /etc/init.d, invoked by hard links to it in the various /etc/rc?.d directories.

The relayer code is straightforward C, and is included in Appendix A. The startup script script is in Appendix B.

meeting requirement (2)

The solution to requirement (1) has the drawback that all Sun syslog messages arrive at loghost with the facility reset to local3; how then to route it by its originating facility to an appropriate file? For example, a message originating with auth.crit would wind up in the same (large) file as a named.info message - not a desirable situation, especially for security purposes, where auth messages should merit special attention, not be buried amidst the named infos.

As we have seen, the syslog selector field permits only the specification of facility.priority, so we must again rely on a helper application. This time we choose Perl, for its text handling capabilities, and have it create another named pipe (FIFO) which it then monitors. The revised enabling line in loghost's syslog.conf then becomes:

```
local3.debug                            /usr/local/adm/pkg/syslog/relay/unwrap
```

where 'unwrap' is the FIFO. The Perl unwrapper script then has the job of determining the original facility.priority and on this basis route the message to the appropriate Sun syslog on loghost.

The problem now becomes how to preserve this facility.priority information so that the unwrapper script can extract it.

On some systems, such as Irix, this information is incorporated into the message part of the syslog packet, making extraction easy. Consider the following syslog message, generated by loghost's syslogd itself:

```
Sep 19 01:40:11 6F:loghost syslogd: restart
```

Here we see the facility.priority (syslog.info) encoded as '6F' and placed directly in front of the originating host name, separated by a colon.

With Solaris 7, however, by default the priority.facility is not incorporated into the actual message:

```
Jun 17 03:10:00 nettest syslogd: logs sync'ed
```

Fortunately, however, an option to include this information exists. The file /kernel/drv/log.conf configures the STREAMS kernel logging device driver interface (man log(7D)). As shipped, the operative line in this file is:

```
name="log" parent="pseudo" instance=0;
```

Simply change this line to read:

```
msgid=1 name="log" parent="pseudo" instance=0;
```

and reboot, in order to have a message ID of the following form inserted into every syslog message:

```
Sep 15 09:57:09 ns named[147]: [ID 8605 local5.debug] no IPv6 interfaces found
```

This new message ID is within brackets and contains the message ID number and the originating facility and priority, which makes it easy for Perl to extract this info for proper routing. So, we can now have on loghost files such as

```
SUNLOG
SUNLOG.auth
SUNLOG.mail
SUNLOG.named
SUNLOG.tcpd
```

File names are arbitrary and set by the unwrapper script, giving us the capability to group by facility our Sun-only syslog entries to files of our choice.

Pipe and reader are toggled by a startup script script in /etc/init.d, invoked by hard links to it in the various /etc/rc?.d directories.

The unwrapper script is in Appendix C.

The startup script script is in Appendix D.

meeting requirement (3)

In meeting requirement (2) we have in essence solved requirement (3); the entry now incorporates all five elements: original facility, priority, host name, time stamp, and actual message. However, we can condense the presentation of the facility and priority message in the same way chosen by Irix: a two-character encoding prepended to the host name and separated from it by a colon. We discard the message ID number, as it is not generally useful. The unwrapper script thus transforms this message:

```
Sep 15 09:57:09 ns named[147]: [ID 8605 local5.debug] no IPv6 interfaces found
```

into:

```
Sep 15 09:57:09 7V:ns named[147]: no IPv6 interfaces found
```

Note that the priority here is given before the facility, in Irix fashion. This makes for a shorter log entry, and is more easily recognized by log processing scripts.

A notable failing of syslog time stamps is its lack of year data. Eric Allman has said that when he created syslog he never imagined that anyone would want to keep log files around for very long (!). The interested reader can readily modify the unwrapper script to insert the year into the time stamp field, if desired. We have not done so at my site, as our archived log files have full date strings postpended to the file name.

meeting requirement (4)

A consequence of the re-syslog'ing performed by the relay code in meeting requirement (1) is that artifacts useful only for encapsulation are prepended to the original log message. Consider the following message, created on sws with the command 'logger -p auth.error "test ms 01"' before relogging it:

```
Sep 19 15:39:32 sws clark: [ID 702911 auth.error] test ms 01
```

Here we see nothing very remarkable; just a contrived message with time stamp, host name, and the message ID we caused to have inserted. Here is the same message as it arrives at the unwrapper script, after having been relogged and relayed to loghost through a Cray (line broken for clarity):

```
Sep 19 15:39:32 7T:cray.mysite.edu relayer[150]: [ID 677204 local3.debug] \
Sep 19 15:39:32 sws clark: [ID 702911 auth.error] test ms 01
```

Notice that the original message has some new, not very useful, artifacts of the relaying process prepended to the original message:

By sws:

- New originating process (just the name and ID of the relaying process).
- New message ID.

By cray:

- Nothing; message and headers are properly relayed as received to loghost.

By loghost:

- Prepends the originating host name as it sees it (cray.mysite.edu). According to the RFC, it should use just the (base) host name (sws) supplied in the packet header (and never use domain names). This apparently nonconformant behavior does not matter for our purposes, as this relaying artifact will be deleted later.
- Next, prepends the facility and priority contained in the packet PRI header, which in this case is that supplied by the relayer code (local3.debug), encoded into two characters and inserted before the host

name, separated from it by a colon.
- Next, prepends the time stamp. This might be the time stamp found in the packet PRI header, or its own view of the time; it does not use the originating host name in the header packet, so may not use the packet header date stamp either. But, as this whole artifact will be deleted later, it does not matter. Besides, the time stamp we are interested in is the original time stamp, which is retained in the message body.

After removing the above sheperding artifacts, the unwrapper script transforms the message ID into an Irix-compatible two-character code, inserted as before, resulting in a more compact message correctly and succinctly incorporating all the original message data:

```
Sep 19 15:39:32 3E:sws clark: test ms 01
```

As implemented, this ms would now be appended to loghost:/var/adm/SUNLOG.auth. This then completes an implementation of a gathering process satisfying our four specified requirements.

the hunting problem

Now that we have gathered together Sun- and facility-specific log entries in these files:

```
SUNLOG
SUNLOG.auth
SUNLOG.mail
SUNLOG.named
SUNLOG.tcpd
```

it remains to hunt down those entries of interest from among the considerable numbers of entries within these files of no interest whatever.

This is not at all a new problem, and there exist well-known tools developed specifically for this task. Ideal solutions to this problem, however, would require:

(1) An on-demand tool that may be run against multiple arbitrary log files.
(2) A real time tool for streaming messages of interest from multiple arbitrary log files as they arrive.

In addition, mechanisms must be provided for easy specification of patterns to include and exclude in the search space. The exclusion of patterns of no interest is a better approach than the inclusion of just those patterns of known interest, as the message space cannot reliably be predicted, but the highlighting and special handling of include patterns of great interest (e.g., "access denied") is also useful. Extant tools commonly combine include and exclude patterns: excluding known entries of no interest, including all the rest by default, with special include patterns to trigger action for those entries of great interest.

An example of a well-known tool meeting requirement (1) is Craig Rowland's logcheck. An example of a well-known tool meeting requirement (2) is Todd Atkin's swatch.

One drawback of swatch, however, is that only one file may be tail'd at a time; this does not meet requirement (2), although of course multiple instances of swatch could be run. A drawback of logcheck is its use of egrep regular expression syntax; much more powerful regex syntaxes are available, particularly that provided by Perl.

There are other minor difficulties in installing and adapting these somewhat dated tools, and the problem in general is not a difficult one, so that the approach taken here was to roll our own. This affords the opportunity of building into these tools some post-processing capabilities lacking in both logcheck and swatch.

meeting requirement (1)

A tool was developed, written in Perl, that allows:

```
  - The specification of multiple arbitrary log files to search. This can
    include archived log files, which may be specified as a date range.
  - The specification of multiple arbitrary configuration files. These files
    contain the patterns to ignore, and may be variously combined according to
    the files being searched, the degree of filtering desired, etc.
  - The specification of multiple arbitrary regular expression files. The use
    of named regular expressions considerably simplifies their use and is less
    error-prone.
  - Post-processing of those patterns not excluded, grouping them first by
    originating process, then sorting these process-specific entries by
    priority level, and finally listing only those importantly unique entries
    with a count of their appearance.

Brief sample output (lines broken for clarity):
================================================================================
/usr/local/bin/sudo
===================
  level 6
  -------
  28    sysmon : TTY=unknown ; PWD=/usr/local/adm/home/sysmon ; USER=root ; \
        COMMAND=/usr/local/adm/bin/fcat -etb
  15    sysmon : TTY=unknown ; PWD=/usr/local/adm/home/sysmon ; USER=root ; \
        COMMAND=/usr/local/adm/bin/ftpcheck
   2    sysmon : TTY=unknown ; PWD=/usr/local/adm/home/sysmon ; USER=root ; \
        COMMAND=/usr/local/adm/sbin/findww /tmp
   2    sysmon : TTY=unknown ; PWD=/usr/local/adm/home/sysmon ; USER=root ; \
        COMMAND=/usr/local/adm/sbin/findww /u2
   1    sysmon : TTY=unknown ; PWD=/usr/local/adm/home/sysmon ; USER=root ; \
        COMMAND=/usr/local/adm/sbin/findww /allsys/u1

================================================================================

This particular tool operates by exclusion of known patterns of no interest
only, and is named syslog.grepv as its primary action is to "grep -v" logs.

  syslog.grepv     is included in Appendix E.
  syslog.grepv.cfg is included in Appendix F.
  syslog.group     is included in Appendix G.
  syslog.regex     is included in Appendix H

meeting requirement (2)
-----------------------
  A tool was developed, written in Perl, that allows:

 - As with syslog.grepv, the specification of multiple arbitrary log files,
   configuration files, and regular expression files.
 - Non-excluded entries are printed to the screen in real time.
 - Included entries of interest may have arbitrary actions associated with
   them, such as printing the entry in inverse video, accompanying the
   printing of the message with a bell, sending a message to someone, etc.

Brief sample output (lines broken for clarity, and IPs obscured):
================================================================================
SUNLOG.auth:    Sep 20 16:05:03 6E:kdc sshd-adm[4811]: Accepted publickey \
  for sysmon from 222.222.22.22 port 33331 ssh2
SUNLOG.named:   Sep 20 16:05:06 6V:ns named[152]: client 222.222.22.22#1037:\
  query: vizsolutions.safeserver.com IN A
SUNLOG:         Sep 20 16:05:17 6B:kdc ACESERVER: (14001) AUTHENTICATION : \
  Primary Requesting From Replica Changes for Table Token From Replica \
  (222.222.22.22)  CrossRealm ACM_OK 0. [mscomm.c.2951.16086]
================================================================================

This particular tool operates by emulating a tail -f on specified log files,
and is consequently named syslog.tailf.
```

syslog.tailf is included in Appendix I.
syslog.tailf.cfg is included in Appendix J.

conclusion

Log files tend to fill quickly with mostly repetitive messages of little interest, yet are also the receptacles of error and event messages of great importance, such as system anomalies, failures, and security breaches. Collecting and archiving these messages in a secure central location, reviewing them both offline and in real time, and extracting those of interest are important system administration tasks. This paper has looked at some of the problems involved with these tasks and provided some possible solutions.

references

http://www.ietf.org/rfc/rfc3164.txt # Syslog protocol.