ARTIFICIAL IGNORANCE
How To Implement Artificial Ignorance Log Analysis
Marcus Ranum

By request, here's a quick how-to on log scanning via artificial ignorance. :) It assumes UNIX and the presence of a good grep - you could use other stuff if you wanted to but this is just an example.

Setting up a filter is a process of constant tuning. First you build a file of common strings that aren't interesting, and, as new uninteresting things happen, you add them to the file.

I start with a shell command like this:

```
cd /var/log

cat * | \
        sed -e 's/^.*demo//' -e 's/\[[0-9]*\]//' | \
        sort | uniq -c | \
        sort -r -n > /tmp/xx
```

In this example "demo" is my laptop's name, and I use it in the sed command to strip out the leading lines of syslog messages so that I lose the date/timestamps. This means that the overall variation in the text is reduced considerably. The next argument to sed strips out the PID from the daemon, another source of text variation. we then sort it, collapse duplicates into a count, then sort the count numerically.

This yields a file of the frequency with which something shows up in syslog (more or less):

```
 297  cron: (root) CMD (/usr/bin/at)
 167  sendmail: alias database /etc/aliases.db out of date
 120  ftpd: PORT
  61  lpd: restarted
  48  kernel: wdpi0: transfer size=2048 intr cmd DRQ
 ... etc
```

In the example on "demo" this reduced 3982 lines of syslog records to 889.

Then what you want to do is trim from BOTH ends of the file and build an "ignore this" list. In this example, I don't care that cron ran "at" OK so I'd add a regexp like:

```
cron.*: (root) CMD (/usr/bin/at)
```

That's a pretty precise one. :)

At the bottom of my file there were about 200 entries that looked like:

```
  1  ftpd: RETR pic9.jpg
  1  ftpd: RETR pic8.jpg
  1  ftpd: RETR pic7.jpg
```

1 ftpd: RETR pic6.jpg

Clearly these are highly unique events but also not interesting. So I add patterns that look like:

ftpd.*: RETR
ftpd.*: STOR
ftpd.*: CWD
ftpd.*: USER
ftpd.*: FTP LOGIN FROM

Now, you apply your stop-list as follows:

```
cat * | grep -v -f stoplist | \
        sort, etc --
```

This time I get 744 lines. Putting a pattern in that matches: **sendmail.*: .*to=**

Drops it down to 120 lines. Just keep doing this and pretty soon you'll have a set of patterns that make your whole syslog output disappear. You'll notice that in the early example I had a warning from sendmail because the aliases database was out of date. Rather than putting a pattern for that, I simply ran newalias. Next time my aliases database is out of date, my log scanner will tell me.

System reboots are cool, too. My log shows:

48 kernel: wdc2 at pcmcia0: PCCARD IDE disk controller
48 kernel: wdc1 at pcmcia0: PCCARD IDE disk controller
48 kernel: wdc0 at isa0 iobase 0x1f0 irq 14: disk controller
48 kernel: wd0 at wdc0 drive 0: sec/int=4 2818368*512
...

Those will be pretty much static. So I add those exact lines. Now they won't show up whenever the system boots. BUT I'll get a notification if a new SCSI drive is added, or (I did this deliberately!):

kernel: fd0c: hard error writing fsbn 1 of 1-19 (fd0 bn 1; cn
kernel: fd0: write protected

Oooh! Some bad boy trying to step on my tripwire file!

Or:

kernel: changing root device to wd1a

..interesting. My pattern was for wd0a!

I used to run this kind of stuff on a firewall that I used to manage. One day its hard disk burned up and my log scan cheerfully found these new messages about bad block replacement and sent them to me. :) The advantage of this approach is that it's dumb, it's cheap -- and it catches stuff you don't know about already.

Once you've got your pattern file tuned, put it in cron or whatever, so it runs often. The TIS Gauntlet has a hack I wrote called "retail" which I can't unfortunately release the code for, but is easy to implement. Basically, it was like tail but it remembered the offset in the file from the previous run, and the inode of the file (so it'd detect file shifts) - the trick is to keep one fd open to the file and seek within it, then stat it every so often to see if the file has grown or changed inode. If it has, read to EOF, open the new file, and start again. That way you can chop the end of the log file through a filter every couple seconds with minimal expense in CPU and disk I/O.

I'm sure there are lots of fun ways this simple trick can be enhanced -- but just in its naive form I've found it quite useful. I wish I had a program that helped me statistically build my noise filters, but in general I find it's about a 2 hour job, tops, and it's one you do once and forget about.

Enjoy!
mjr.

Marcus J. Ranum, CEO, Network Flight Recorder, Inc