Automated System Monitoring and Notification With Swatch

Stephen E. Hansen & E. Todd Atkins
Stanford University

Automated System Monitoring and Notification With Swatch

Stephen E. Hansen & E. Todd Atkins – Stanford University

ABSTRACT

This paper describes an approach to monitoring events on a large number of servers and workstations. While modern UNIX systems are capable of logging a variety of information concerning the health and status of their hardware and operating system software, they are generally not configured to do so. Even when this information is logged, it is often hidden in places that are either not monitored regularly or are susceptible to deletion or modification by a successful intruder. Also, a system administrator must often monitor several, perhaps dozens, of systems. To address these problems, our approach begins with the modification of certain system programs to enhance their logging capabilities. In addition, our approach calls for the logging facilities on each of these systems to be configured in such a way as to send a copy of the critical system and security related information to a dependable, secure, central logging host system. As one might expect, this central log can see a megabyte or more of data in a single day. To keep a system administrator from being overwhelmed by a large quantity of data we have developed an easily configurable log file filter/monitor, called *swatch.* Swatch monitors log files and acts to filter out unwanted data and take one or more user specified actions (ring bell, send mail, execute a script, etc.) based upon patterns in the log.

The Problem

It is an unfortunate fact that most UNIX systems, as delivered, do little to ease the job of the system administrator when it comes to keeping tabs on the health of those systems. Often, the first inkling of a problem occurs when keystrokes stop being echoed or the phone rings.

What every good system administrator tries to do is keep an eye on the health of each of the systems in his or her care. The health of a system should be reflected in the log messages generated by the kernel and the various daemons and utilities. In addition, these messages should also include information relevant to system security. However, with most systems we have seen, the system's log information is not generally made available to the system administrator in a way that is either secure or convenient, rather it is often hidden in places that are either not monitored regularly or are susceptible to corruption or destruction by system failure or a successful intruder. The assumption seems to be that system log files are only to be consulted after the fact, to help with postmortem rather than prevention. What this means is that the UNIX *syslog*(3) facility, regardless of the original intent, is generally used as more of a debugging aid than as a tool for system management.

Improved Security Logging

For purposes of monitoring systems security, standard UNIX logging features often prove to be inadequate and/or inconvenient. To address this problem, our approach begins with the modification of certain system utilities to enhance the reporting done, particularly with regard to possible security related activities. Table 1 lists some of the utilities modified and the changes made to their logging capabilities.

Program	Logging Enhancements
fingerd	Reports the originating host and the *finger* target(s) to syslog.
ftpd	Reports originating host to syslog. Reports file transfers to a local log file along with the local user name and, if the user is "anonymous", the password.
ruserok	Used by *rshd* and *login* when called by *rlogind.* Disallows and reports to syslog any attempts to use a */etc/hosts.equiv* or *~/.rhosts* file that contains a '+'.
rshd	Reports the access status, local user, remote user and host, and the command issued to a local log file.
login:	Reduced number of tries to three. Reports to syslog on 'Incomplete Login Attempt', 'Repeated Login Attempt', and 'Root Login Refused'. Includes the account names attempted and the originating host.

Table 1: List of logging enhancements made to several system programs.

At our site we were fortunate enough to have access to the vendor's source code for all our utilities. While this is not possible for everyone, each of the utilities listed in Table 1 are available from various network archive sites. In a few cases it might be preferable to use the public version instead to improve portability. Another source of security related information is from the tcp wrapper code written by Weitse Venema[1]. Besides providing access control for those network services run out of *inetd*, it generates information via syslog about the connections it mediates.

One important utility not listed in Table 1 is *sendmail*(8). Even without modification sendmail can be configured to generate a plethora of status information. Unfortunately, *sendmail* isn't very discriminating in what it reports, assigning every status message the same priority.

Centralized Logging with syslog

When we have added to the logging capabilities of the various utilities, we have, for the most part, made use of the syslog library functions. Besides providing a consistent and relatively standard logging interface, syslog directs logging messages to different files or hosts based upon the source of the message and its level of importance.

The way our facility is set up, each server system keeps its own copy of most of the syslog messages in the file */var/log/syslog*. These syslog files are rotated on a daily basis, compressed, and kept online for about a week. Log messages that might reflect a system's health or potential security problems are also forwarded to a central log host, the LOGMASTER. In practice this means that almost everything except sendmail status messages are sent to the LOGMASTER. Leaving the sendmail status messages on the servers cuts down on the network traffic due to syslog without significantly affecting our ability to monitor. On our systems the sendmail messages can account for as much as 90% of a host's log messages, although 50% is more common. Appendix A shows the syslog configuration file (*/etc/syslog.conf*) for a host being monitored. The last three lines in the file are responsible for sending data to the LOGMASTER.

Copying the syslog information to a central site is done for several reasons. First, it provides redundancy and security. If the log files on the originating host are destroyed or modified, either accidently or by malicious intent, those on the more secure central site will be left intact. Second, it simplifies the monitoring of all the log information. By collecting information from a number of systems in a single time ordered file, problems may be found that would be missed if viewed in isolation, such as network or security related problems. For example, a single failed log in attempt on one system might be attributed to a typing error. The same failed log in attempt occurring on several systems in sequence could indicate an intruder trying to break in. Collecting information from several different system utilities as well as from more than one system can provide information indicating a pattern of attack. Several *fingers* followed by a failed *login* or *rsh* command is a common pattern revealed by this type of monitoring.

Winnowing the Chaff: An Introduction to Swatch

Our facility manages about a dozen file and CPU servers which have over 50 client machines. The server systems receive an enormous amount of log information through the syslog daemon. Even after filtering out the sendmail information messages the LOGMASTER sees about a megabyte of syslog messages per day. As one can imagine, sorting through that much information on a daily basis can be very time consuming. We also found that some important log entries tend to get lost among all of the less important entries when examining the log files.

One solution to this problem would be to search for certain types of information, which can be done by using the *egrep*(1) program with some complex command line arguments. Even with this solution one still has the problem of having to constantly monitor the output so that the urgent information is seen when it comes in. Some of this information needs to be acted on soon after it is received. For example, if the system on a file server machine locks up then somebody needs to be alerted so the machine can be brought back up as quickly as possible. For us the most desirable solution was to have a more complex program sift through the log and do a few simple tasks when certain types of information were found. We decided to call this program *swatch*, which stands for *S*imple *WATCH*er.

Swatch Design Goals

There were four goals that were set when designing swatch.

1. Configure the program in such a way that it would only take a few minutes to teach any systems administrator how to use it.
2. Have a simple set of actions that could be performed after receiving certain types of information.
3. Allow swatch's users to define their own actions if they like, and allow them to use parts of the input as arguments to the action.
4. Once swatch is running it should be reconfigurable on demand or after a specified interval without having to stop and restart the program manually.

Using Swatch

Swatch may be run three different ways: make a single pass through a file; look at messages that are being appended to a file as that file is being updated; or examine the standard output of a program. A complete description of swatch's command line options can be found in Appendix B.

Swatch's most powerful function is in examining information as it is being appended to a log file. We use swatch to look at messages as they are being added to the syslog file, alerting us immediately to serious system problems as they occur. Using a *tail*(1) of */var/log/syslog* as input is the default action for swatch but another file can be "tailed" by using the -t command line option as in

```
swatch -t /var/log/authlog
```

Receiving timely notification of certain types of probes or attacks often enables us to find out which users are logged on to the originating system. Finding out such information can help identify hackers or compromised accounts.

By using the -f option, swatch can be made to read in and process a file from beginning to end. This single pass feature can be used to examine old syslog or other text files.

```
swatch -f /var/log/syslog.0
```

This option can be used to catch up on the contents of log files after being away from the computer for a while (like after vacationing in Hawaii for a week). This feature is often used to filter through several megabytes of old syslog files to look for evidence of suspected system and network related problems as well as system probes and break-in attempts.

Having swatch examine the output from a program is also useful. For example, one might want to sort through process accounting or other audit information that is not kept in a plain text file and requires special processing to read.

```
swatch -c swatchrc.acct -p lastcomm
```

Implementation

Swatch relies heavily on expression matching. For this reason the Perl[2] language was used because of its Awk and C like characteristics, as well as its increasing familiarity among systems administrators.

Swatch has three basic parts: a configuration file, a library of actions, and a controlling program.

Configuration File

Each non-comment line in a swatch configuration file consists of four tab separated fields: a pattern expression, a set of actions to be done if the expression is matched, an optional time interval, and the location of a time stamp, if any. As shown in Figure 1, a line's pattern field consists of one or more comma separated expressions while the action field may contain one or more comma separated actions.

The patterns must be regular expressions which Perl will accept, which are very similar to those used by the UNIX egrep program. Each string to be matched is compared, in order, with the expressions in the configuration file and if a match is found the corresponding actions are taken. A copy of the UNIX manual page for swatch's configuration file is listed in Appendix C.

The time interval can be used to help eliminate redundant messages. For example, on our systems `"file system full"` messages tend to come at the rate of several dozen per minute. We specify an interval of five minutes which will usually eliminate hundreds of redundant notifications.

The time stamp location information is optional and can only be used when a time interval is specified. Swatch uses it to strip away the time stamp in order to compare it to other messages which are stored in its internal history list.

Lines beginning with the '#' character are treated as comment lines and are ignored.

Actions

Swatch understands the following actions: echo, bell, ignore, write, mail, pipe, and exec.

- The **echo** action causes the line to be echoed to swatch's controlling terminal. An optional mode argument causes the text to be shown in normal, bold, underscore, blinking, or inverse mode. Normal mode is the default.
- The **bell** action sends a bell signal (^G) to the controlling terminal. An optional argument specifies the number of bell signals to send, with one being the default.
- The **ignore** action causes swatch to ignore the current line of input and proceed to the next one. The ignore action is mainly useful early on in the configuration file to filter out specific unimportant information that would otherwise match a more general expression found later in the configuration file.
- The **write** and **mail** actions can be used to send a copy of the line to a user list via the write and mail commands.
- The **pipe** and **exec** actions were added to provide some flexibility. The pipe action allows the user to use matched lines as input to a particular command on the system. The exec action allows the user to run a command on the system with the option of using selected fields from the matched line as arguments for the command. A $N will be replaced by the

/pattern/[,/pattern/,...]   action[,action,...]   [[[HH:] MM:] SS   [start:length]]

Figure 1: Format of pattern action line for a Swatch configuration file.

Nth field in the matched line. A $0 or a $* will be replaced by the entire line.

See Appendix C for more details on the actions and their arguments.

Controlling Program

The controlling program is swatch, but the real work is done by a watcher process. Swatch's first task is to translate the configuration file into a Perl script. After creating the watcher script, swatch forks and executes it as the watcher process. The watcher script also contains two signal handlers. Upon receiving an alarm (SIGALRM) or hang-up (SIGHUP) signal swatch will terminate the watcher process, re-read the configuration file, and start a new watcher process. If a quit (SIGQUIT) terminate (SIGTERM) or interrupt (SIGINT) signal is received, swatch will attempt to cleanup after itself then exit.

Examples

We have previously described several ways in which swatch can be used. In this section we will illustrate the two most common ways in which swatch is used at out facility. First, we have a swatch job running continuously looking for failed login attempts and system crashes and reboots. The swatch configuration file we use for this purpose is shown in Figure 2.

Second it's common for each system administrator to have a customized swatch configuration file in his or her home directory, ~/.swatchrc, that contains pattern/action pairs that are personally interesting, or that pertain to his or her system responsibilities. A swatch job using this configuration file is generally run in a window while the administrator is logged in. The personal swatch configuration file of one of the authors is shown in Figure 3, while Figure 4 shows six hours of output generated by this configuration.

Example 1: Continuous Monitoring for High Priority Events

This swatch configuration (Figure 2) runs in the background and continuously looks for high priority events, such as "file system full" and "panic" messages.

```
#
# Swatch configuration file for constant system monitoring in the background
#
# Test the pager every once in a while
/test pager/                  exec="/etc/call_pager 5551234 123"
#
# Bad login attempts
/INVALID|REPEATED|INCOMPLETE/ exec="/etc/backfinger $0"
#
# EECF
/EE-CF.*(panic|halt)/         mail=action,exec="/etc/call_pager 5551212 0911"
05:00   0:16
/EE-CF.*reboot/               mail=action,exec="/etc/call_pager 5551212 0411"
05:00   0:16
/EE-CF.*SunOS Release/        mail=action,exec="/etc/call_pager 5551212 0411"
05:00   0:16
/EE-CF.*file system full/     mail=action,exec="/etc/call_pager 5551212 0611"
05:00   0:16
#
# Sierra
/Sierra.*WizMON/              mail=action,exec="/etc/call_pager 5551234 1666"
05:00   0:16
/Sierra.*(panic|halt)/        mail=action,exec="/etc/call_pager 5551234 1911"
05:00   0:16
/Sierra.*reboot/              mail=action,exec="/etc/call_pager 5551234 1411"
05:00   0:16
/Sierra.*SunOS Release/       mail=action,exec="/etc/call_pager 5551234 1411"
05:00   0:16
/Sierra.*file system full/    mail=action,exec="/etc/call_pager 5551234 1611"
05:00   0:16
```

Figure 2: Swatch configuration file for continuous monitoring

The first pattern/action line is used to test our pager number periodically to ensure that swatch, our dial out line, and our pager are all working. We run the *logger*(1) program periodically via *cron*(1) to send a message which contains the string "test pager." This causes swatch to attempt to page our on call systems administrator.

The second pattern/action line looks for a */bin/login* syslog message of the form

```
Jul 30 13:49:47 Sierra login:
  REPEATED LOGIN FAILURES ON
  ttyq0 FROM cert.cert.org:
  root, anonyme, anonyme
```

The string `REPEATED` matches the pattern and swatch executes a script to finger the host that initiated the failed login and store the information for later examination. We are occasionally able to detect compromised accounts with this information.

The rest of the file contains pattern/action lines that are grouped by specific names of machines. It watches out for kernel messages which indicate a potentially serious problem, such as a machine crash or an unexpected reboot. It also looks for messages from the room temperature monitor. When these types of messages are encountered, swatch sends a mail message to our systems administrator mailbox and executes a script to call a pager with a code indicating the system and message type.

Example 2: Individualized Swatch Configuration File

Individuals may design customized swatch configuration files that look for patterns and take appropriate actions depending on their personal preferences. The configuration file shown in Figure 3 is run in a workstation window whenever the system administrator is logged in. The output (a sample of which is shown in Figure 4) is generally ignored or

```
#
# Personal Swatch configuration file to be run in a window on a workstation
#
# These probes should be harmless, but who knows?
#
/fingerd.*(root|[Tt]ip|guest|atkins)/   echo,bell,exec="/bin/date >>
  /home/atkins/tmp/finger.log",exec="/usr/local/etc/backfinger @$6 >>
  /home/atkins/tmp/finger.log"
#
# This should never happen
/su: atkins/                            echo,bell
/su: .* failed/                         echo,bell=3

/[dD]enied/||/DENIED/                   echo=boldunderline,bell

# Alert me of bad login attempts and find out who is on that system
/INVALID|REPEATED|INCOMPLETE/           echo=underline,bell=3

# Important program errors
/LOGIN/                                 echo=boldunderline,bell=3
/passwd/                                echo=bold,bell=3
/ruserok/                               echo=bold,bell=3

# Ignore this stuff
/sendmail/,/nntp/,/xntp|ntpd/,/faxspooler/  ignore

# Report unusual tftp info
/tftpd.*(ncd|kfps|normal exit)/         ignore
/tftpd/                                 echo,bell=3

# Kernel problems
/(panic|halt|SunOS Release)/            echo=blink,bell        3:00    0:16
/file system full/                      echo=bold,bell=3       5:00    0:16

# Try to ignore uninteresting kernel messages
/vmunix.*(at|on)/                       ignore
/vmunix/                                echo,bell              1:00    0:16
```

Figure 3: Personalized swatch configuration file

only occasionally glanced at unless the bell alerts the administrator to a message of interest. Note that the tftpd pattern/action lines in Figure 3 ignore tftp requests from valid hosts and alert the user to invalid requests.

Other Useful Programs

We have written a few scripts which we have found useful when using the swatch package.

```
Sep 13 05:07:23 Sierra vmunix: ie0: no carrier
Sep 13 07:32:07 Sierra ftpd[17910]: FTP LOGIN FROM thermo-amy.Stanford.EDU [36.6
5.0.83], eaton
Sep 13 07:35:58 Sierra ftpd[18015]: FTP LOGIN FROM thermo-amy.Stanford.EDU [36.6
5.0.83], eaton
Sep 13 07:58:30 gloworm login: INCOMPLETE LOGIN ATTEMPT ON ttyp2 FROM deis17.cin
eca.it
Sep 13 08:15:35 loading-zone.Stanford.EDU vmunix: /loading-zone: file system ful
l
Sep 13 08:15:35 stjames vmunix: NFS write error: on host loading-zone remote fil
e system full
Sep 13 08:53:25 Sierra login: REPEATED LOGIN FAILURES ON ttypb FROM uwmfe.neep.w
isc.edu: help, newuser, d
Sep 13 09:26:59 Sierra su: 'su root' failed for quinn on /dev/ttyp9
Sep 13 09:45:04 espresso.Stanford.EDU login: ROOT LOGIN ttyp0 FROM coffee
Sep 13 10:04:50 Gordon-Biersch vmunix: pid 16100: killed due to swap problems in
exec: I/O error mapping pages
Sep 13 10:05:20 Sierra ftpd[21910]: FTP LOGIN FROM thermo-amy.Stanford.EDU [36.6
5.0.83], eaton
Sep 13 10:06:25 Gordon-Biersch vmunix: /tmp: file system full, anon reservation
exceeded
Sep 13 10:06:43 Gordon-Biersch vmunix: pid 16118: killed due to swap problems in
exec: I/O error mapping pages
Sep 13 10:07:02 Gordon-Biersch vmunix: pid 16124: killed due to swap problems in
exec: I/O error mapping pages
Sep 13 10:09:34 Sierra ftpd[22085]: FTP LOGIN FROM thermo-amy.Stanford.EDU [36.6
5.0.83], eaton
Sep 13 10:33:55 Gordon-Biersch fingerd[16484]: pudleys.Stanford.EDU (36.2.0.92.1
654) -> "atkins"
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: SunOS Release 4.1.1 (ISL_CLIENT) #
1: Mon Jan 13 08:58:58 PST 1992
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: Copyright (c) 1983-1990, Sun Micro
systems, Inc.
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: mem = 24576K (0x1800000)
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: avail mem = 22630400
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: Ethernet address = 8:0:20:b:67:21
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: cpu = Sun 4/40
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: sd0: <SUN0207 cyl 1254 alt 2 hd 9
sec 36>
Sep 13 11:35:13 espresso.Stanford.EDU vmunix: sd2: <Fujitsu M2624F cyl 1463 alt
2 hd 11 cyl 1463 alt 2 hd 11 sec 63>
Sep 13 11:54:22 espresso.Stanford.EDU vmunix: rebooting...
Sep 13 11:56:40 espresso.Stanford.EDU vmunix: NOT BLOCK: GOTO REQUESTLOOP
Sep 13 11:56:50 espresso.Stanford.EDU vmunix: zs3: silo overflow
Sep 13 12:06:05 Sierra ftpd[28258]: FTP LOGIN FROM vali.Stanford.EDU [36.59.0.32
], fanning
Sep 13 12:11:10 Sierra ftpd[29236]: FTP LOGIN FROM me-bradshaw.Stanford.EDU [36.
65.0.71], bradshaw
```

Figure 4: Output from swatch using the configuration file in Figure 3 over the course of 6 hours and more than 2300 lines of input

Reswatch

Reswatch was written to run out of cron periodically. It finds all instances of swatch that the user is running and sends a SIGHUP. This is useful if swatch is getting its input from an active log file, like syslog, that is moved and rendered inactive. Since we want to start getting our input from the new active log file, the old file handle needs to be closed and the new one opened. This effect is achieved when swatch aborts one script and starts a new one after receiving a SIGHUP.

Backfinger

Backfinger is used to finger the host that generated an unsuccessful login attempt. Output from this command is placed in its own log file. Backfinger uses safe_finger to filter out potentially dangerous output from remote finger servers. This is most useful when culprits fail to log in to a system using an unauthorized account, like root, guest, or anonymous. Some administrators might be surprised at how often this happens on their systems.

CallPager

For those who must carry a pager, this is very useful for receiving urgent information, such as serious system failures or possible security breaches. This is a simple script which uses the UNIX tip command to call a pager through a modem and leave a code number to indicate the type of message detected. Users can customize the codes so that they can tell exactly what type of message was detected, and the system from which it came.

Conclusions

Over the past year and a half swatch has proven to be a valuable tool for monitoring the health of a large collection of workstations and servers. On several occasions we have been able to detect intruders probing our systems who would probably have been missed without centralized logging and swatch. On a few occasions it prevented system meltdown when air conditioning units failed on a weekend or late at night. Its value has increased as we have gathered more experience in optimizing the swatch configuration file entries.

In the near term, we see a need to improve the logging capabilities of additional system utilities (i.e. sendmail, ntp, ypserv, xdm, xlogin). We plan to gather suggestion from other sites using the package before making substantial changes to swatch itself.

Availability

Swatch source and documentation along with its companion scripts are available via anonymous ftp from Sierra.Stanford.EDU, [36.2.0.98], in the pub/sources directory. Listserver access is available from listserver@Sierra.Stanford.EDU.

Author Information

Stephen E. Hansen received the B.S. and M.S. degrees in Electrical Engineering from Stanford University in 1976 and 1981 respectively. In 1975 he joined the Integrated Circuits Laboratory at Stanford University, first as Systems Programmer, and since 1978 as Senior Scientific Programmer. In 1983 he organized the Electrical Engineering Computer Facility at Stanford where he currently serves as its Director. Mr. Hansen can be reached via U.S. Mail at the Applied Electronics Laboratory 218, Stanford, CA 94305-4055 or via electronic mail at hansen@sierra.stanford.edu.

Todd Atkins received a B.S in Electrical Engineering from Stanford University in 1988. Since 1987 he has been with the Electrical Engineering Computer Facility as a Systems Administrator. Mr. Atkins can be reached via U.S. Mail at the Applied Electronics Laboratory, Room 113, Stanford, CA 94305-4055 or via electronic mail at Todd_Atkins@eecf.stanford.edu.

References

[1] W. Venema. "TCP WRAPPER, A Tool for Network Monitoring, Access Control, and for Setting Up Booby Traps", Proc. 1992 USENIX Security Symposium, USENIX Association, Sept. 1992.

[2] L. Wall and R. Schwatz. "Programming Perl", O'Reilly and Associates, Sebastopol, CA. 1991.

Appendix A: A Syslog Configuration File.

```
# syslog configuration file.
#
# Master syslog configuration file.
#
# This file is processed by m4 so be careful to quote (`') names
# that match m4 reserved words. Also, within ifdef's, arguments
# containing commas must be quoted.
#
# Note: Have to exclude user from most lines so that user.alert
#       and user.emerg are not included, because old sendmails
#       will generate them for debugging information. If you
#       have no 4.2BSD based systems doing network logging, you
#       can remove all the special cases for "user" logging.
#
*.err;kern.debug;auth.notice;user.none                     /dev/console
*.err;kern.debug;daemon,auth.notice;mail.crit;user.none   /var/adm/messages
lpr.debug                                                  /var/adm/lpd-errs

# You may want to add operator to the following if your operator
# is a traditional Unix style operator.
*.alert;kern.err;daemon.err;user.none                      root
*.emerg;user.none                                          *

ifdef(`LOGHOST',
# for loghost machines, to have authentication messages (su, login, etc.)
# logged to a file, un-comment out the following line and adjust the file
# name as appropriate.
auth.notice                                                /var/log/authlog
daemon.info;auth.notice;mail.debug;kern.debug             /var/log/syslog
*.err;daemon.none;mail.none;kern.none;auth.none;user.none /var/log/syslog
)

# following line for compatibility with old sendmails. they will send
# messages with no facility code, which will be turned into "user" messages
# by the local syslog daemon. only the "loghost" machine needs the following
# line, to cause these old sendmail log messages to be logged in the
# mail syslog file.
#
ifdef(`LOGHOST',
user.alert                                                 /var/log/syslog
)

#
# non-loghost machines will use the following lines to cause "user"
# log messages to be logged locally.
#
ifdef(`LOGHOST', ,
user.err                                                   /dev/console
user.alert                                                 root
)

# Send most everything to the LogMaster. If this is the logmaster,
comment out the following two lines
*.info;kern.none;mail.none                                 @logmaster
kern.debug;mail.err                                        @logmaster
```