Real-time Log File Analysis Using the Simple Event Correlator (SEC)

John P. Rouillard
rouilj@cs.umb.edu

Computer Science Department
University of Massachusetts at Boston

Presented Nov 18, 2004 at the LISA 2004 Conference: Atlanta, GA

Who cares about log events?

Nobody cares about the events.

Everybody cares about the problems that cause the events.

Why not log analysis?

- Too much information
- Patterns difficult to see
- Need to respond to problems indicated by logs
- ...

How is info in logs presented?

- Data in the event itself
- Spread across multiple log entries
- Absence of an event
- Relationships between events

- Full info in the event itself

  e.g. (disk full, login info, numeric data)

- Spread across multiple log entries Save event type
  - rate of arrival, time based thresholding
  - rate of success 70% of last events 7/10.
  - trend analysis problem getting better/worse based of data in log

  We want username and failure, but the logs give use one log entry with pid and username, another entry has pid and failure event.

  Also look at trends in the log messages. Is problem getting worse or better?

- Missing events

  e.g. Events that should occur don't. Failure in process. Difficult to determine since analysis is event driven. How do you detect a missing event?

- Relationships between events

e.g. Failure origin/development shown in time series. Server can't resolve hostnames is secondary to DNS server down.

Multiple events and the tie problem

Need information to **tie** or connect other events into a thread. May be generated externally and injected into event stream, or may occur in the event stream, but not in the correct order.

May be generated externally and injected into event stream, or may occur in the event stream, but not in the correct order.

This leads to a reordering problem.

Event relationships

- Before vs After
- Sequences
- Coincident within window (order unimportant)
- Reordering issues

- Before vs After

  Different problem cause if event A occurs before event B vs event B occurring before event A.

- Sequences

  Ultimate version of strict ordering.

- Coincident within window (order unimportant)

  Exact order depends on circumstances other than cause.

- Reordering issues

  Events may not be generated in correct order for correlation to proceed since tie event occurs after events to be placed into thread.

Detecting missing events

Until now mostly talking about receiving and event and analyzing it and its relationship to other events. Missing events break this "Event Driven" model.

Event detection is well event driven.

How to detect something that is not present?

How do we know to look?/Maintenance of data for missing events.

Missing event detection: Is the logging subsystem working? (1)

How do you know if your logging subsystem is working?

This example uses heartbeat detection to verify proper operation of log infrastructure. Use logger(1) from cron to generate heartbeat.

Would like to use built in syslog mark mechanism, but its not forwarded between machines. Hence we fall back to logger.

logger -p daemon.notice – '– HEARTBEAT –'

Priming the pump.

Auto add/discovery.

Is the logging subsystem working? (2)

Get around the missing event issue by generating seed events.

```
type=single
desc=play seed file for timestamps
ptype=regexp
pattern=SEC_STARTUP|SEC_RESTART
context=SEC_INTERNAL_EVENT
action = spawn (/bin/cat timestamp.seed; echo "TIMESTAMP_SEED_DONE");\
         create seeding_timestamps
```

Play a seed file with events to start timers for all hosts.

Removed:

```
create seeding_timestamps 30 shellcmd \
/usr/bin/mailx -s "seeding of timestamps failed to complete \
in 30 seconds on ‘hostname‘" admin Also
```

to generate an event to allow detection of end of seed file. This seed file is played when a special internal SEC event is received that indicates a start or restart operation. Limit the amount of time spent processing the seed file to 30 seconds. Report a problem if not done in that time.

Is the logging subsystem working? (3)

When the seed file is done, process the `TIMESTAMP_SEED_DONE` event and destroy the context that will report a problem.

```
type=single
desc=Handle end of timestamp seeds
ptype=regexp
pattern=^TIMESTAMP_SEED_DONE$
context = seeding_timestamps
action = delete seeding_timestamps
```

delete doesn't trigger the command associated with the `seeding_timestamps` context.

Is the logging subsystem working? (4)

Maintain the seed file recording any new events.

```
type=single
desc=detect new timestamps from unseen hosts
ptype=regexp
pattern=([A-z0-9._-]+) rouilj: \[.*\] -- HEARTBEAT --$
context=!seen_timestamp_from_$1 && ! seeding_timestamps
action = write timestamp.seed $0; create seen_timestamp_from_$1
continue=takenext
```

Check to see if a timestamp context exists for this host. If so, ignore it. If not append the message to the end of the timestamp.seed file so its existence will be remembered.

We don't do this while we are seeding with the timestamp file since the entry already exists.

Since continue=takenext, we use the HEARTBEAT event to update the per host contexts.

Another rule similar to this creates `seen_timestamps_from_$1` while `seeding_timestamps`.

Is the logging subsystem working? (5)

On every HEARTBEAT event, create a context for 21 minutes (1260 seconds) that will be renewed every 20 minutes by arriving events. If the event fails to arrive, report a problem.

```
type=single
desc=Detect missing timestamps
ptype=regexp
pattern=([A-z0-9._-]+) rouilj: \[.*\] -- HEARTBEAT --$
action= create timestamp_for_$1_active 1260 shellcmd \
   /usr/bin/mailx -s "Missing timestamp heartbeat for $1" admin
```

Create the context that will issue a warning on its expiration. timestamps arrive every 20 minutes by default, so time out in 21 minutes.

Shows:

Priming the pump.

Auto add/discovery.

PROBLEM: no rearm after context fires. Fix by creating rearm event since it is not possible to recursively create context with itself as trigger.

In read delployment, need to monitor every event facility at lowest passing level to make sure changes to syslog.conf don't lose log info. Modify host to include host.facility in contexts.

Combining multiple events: openssh (1)

When using openssh, errors that occur due to failing tunnels are an indicator of a user having a problem. However, the error is not logged with the username.

Combining multiple events: openssh (2)

Because of the privilege separation security method, two different processes are responsible for recording username and error information.

This leads to a difficult correlation problem.

Need a tie event, but ssh doesn't provide one. Doesn't matter if its a bow tie or regular tie, but one must be synthezed. Nice to hear a laugh. It proves that at least one person in the audience isn't alseep, even if that person needs serious psychological help.

Combining multiple events: openssh (3)

There are three steps in this correlation:

1. record the user login reported by process 1

2. handle a log event that ties process 1 and process 2 together

3. record the error information from process 2 with the user info from process 1.

One problem is that the tie event comes after events from process 2 are reported. So we have to buffer all events from process 2 and handle them after the tie event somehow.

Real order can be 1,3,2.

Record the user login reported by process 1

```
type=pairwithwindow
ptype=regexp
pattern=([\w._-]+) sshd\[(\d+)\]: \[ID 800047 auth.info\] \
        Connection from ([\d.]*) port
desc=Look for valid login from $3 to ssh daemon on $1 pid $2.
action=report ssh_$1_$2_log /bin/cat > unfound_login
desc2=Found valid login from %3 to ssh daemon on %1 pid %2
continue2=takenext
ptype2=regexp
pattern2=$1 sshd\[$2\]:.*Accepted (?:password|publickey) \
        for (\w+) from $3
action2= add ssh_%1_%2_log $0
window=60
```

Handle log event tying process 1 and process 2 together

Use alias to link parent's log context with that of the child.

```
type=pairwithwindow
desc=trigger parent $2 reunion with child on $1 for user $3
ptype=regexp
pattern=([\w._-]+) sshd\[(\d+)\]:.*Accepted \
        (?:password|publickey) for (\w+) from
action=report ssh_$1_$2_log /bin/cat > no_child_found
desc2=It's a parent and child reunion on $1 for %3
ptype2=regexp
pattern2=$1 $3\[.*SSHD child process (\d+) spawned by $2
action2=alias ssh_%1_%2_log ssh_%1_$1_log
window=20
```

The tie event is generated by the /etc/sshrc file this is run upon sucessful login.

Record error information from process 2 with the user info from process 1

By writing to the child log (`ssh_$1_$2_log`), it adds the error to the log that has recorded the login information.

```
type=single
desc=Report a problem
ptype=regexp
pattern=([\w._-]+) sshd\[([0-9]+)\]: \[ID 800047 auth.error\] \
       error: connect_to .* Network is unreachable
action= add ssh_$1_$2_log $0; report ssh_$1_$2_log email_problem
```

The wrap up

Successful log analysis requires the ability to identify log messages that provide all relevant information needed to address problems. This includes the ability to detect missing log entries as well as the ability to connect different log messages together using temporal information as well as information located within the log message.