Log Analysis and Event Correlation Using Variable Temporal Event Correlator (VTEC)

Paul Krizak - Advanced Micro Devices, Inc.
paul.krizak@amd.com

ABSTRACT

System administrators have utilized log analysis for decades to monitor and automate their environments. As compute environments grow, and the scope and volume of the logs increase, it becomes more difficult to get timely, useful data and appropriate triggers for enabling automation using traditional tools like swatch. Cloud computing is intensifying this problem as the number of systems in datacenters increases dramatically. Cloud computing also begs for more complex automation techniques in system maintenance and service operation. Modern solutions such as SEC and Splunk have done a great job at scaling to large log volumes and making complex correlations possible, but they have drawbacks. This paper presents an alternative solution we developed at AMD called the Variable Temporal Event Correlator, or "VTEC."

VTEC is unique in that AMD designed it with multi-core and multi-system scalability in mind. Virtually every component is multi-process and/or multi-threaded to take advantage of every available CPU cycle on the system. If needed, each component can be isolated on its own machine to distribute load. VTEC also introduces a novel method for representing temporal event data in a way that is immediately useful for enabling autonomic computing. Finally, it includes a built-in job scheduler that allows for categorization, scheduling, and prioritization of actions generated in response to events.

The VTEC system consists of five modules. Each module has a specific, well-defined task to perform, and communicates with the other modules in a well-defined language over TCP sockets.

- **Streamer -** A piece of utility software that can tail log files or accept input from STDIN. Its most useful feature is the ability to insert "heartbeat" messages into the log stream so the *absence* of event data can trigger actions.
- **Syslog-ng -** VTEC uses the powerful syslog-ng system logger as the log router; it is the only non-Perl component. Its purpose is to accept log streams from syslog, TCP/UDP sockets, and Streamers. It reformats the messages, filters the data, and routes messages to the appropriate rule engines.
- **Rule engines -** These are composed of any executable code that accepts data on STDIN. In practice, Perl scripts that include reusable interface modules are used for this purpose. The rule engines accept filtered log data from syslog-ng and interact with the temporal variable server and the action server (see following bullets). The rule engines are where the event correlation tasks occur. Since the rule engines are open-ended editable custom scripts, a rule engine can do *anything* your language of choice can do.
- **Temporal variable server -** VTEC hosts all the temporal variables in a separate server daemon. This frees the rule engines from the drudgery of maintaining state across reboots/restarts and allows rule engines to share data easily. The server is also able to inject log messages into the log stream when temporal variables exceed thresholds that the rule engines define, freeing the rule engines from having to implement complex timeout conditions to query variable values.
- **Action Server -** When rule engines need to take some sort of action, they have the option of running that task locally (which is not advisable, since this can block the rule engine from processing incoming data) or queuing a job in the action server. The action server has a number of job queues with varying priorities.

Tags: security, case study, syslog, log analysis, event correlation, temporal variables

Users can schedule jobs to run immediately or at a specific time (e.g., alert a sysadmin about this event, but not until 10 a.m., when they are awake). VTEC processes queues with higher priority first, allowing an emergency page to go out immediately despite a backlog of less urgent repair jobs in lower-priority queues.

The most interesting and novel aspect of VTEC is the temporal variable server and the temporal variables it hosts. There are three data types in the temporal variable server:

- **Scalar -** A scalar consists of two pieces of information: a piece of scalar *data* (such as a string or a number) and a *timeout*. The timeout is set when the scalar is created, and defines the amount of time the data is valid. When the scalar is queried, the timeout is checked. If the current time has not exceeded the timeout, the data value is returned. Otherwise, a zero is returned. Scalars are useful for setting alarms, preventing e-mail floods, and storing temporary data.
- **Incrementer -** The incrementer data type builds on the scalar. It is an organized collection of scalars, each with a data value of 1. When an incrementer is instantiated, it is given a timeout value. Every time the incrementer is set (called a *hit* in VTEC), a new scalar is added to the collection with the predetermined timeout and a data value of 1. When queried, the incrementer returns the sum of the values of its constituent scalars. Incrementers are useful for calculating and observing the *rate* of events over time.
- **List -** This data type builds on the incrementer. A list is a collection of incrementers that are each referenced by a *key* - in short, a Perl hash of incrementer objects. Lists have the unique property that they can be queried in three different ways:
    1. The value of a particular key (the current value of that incrementer);
    2. The sum of the current values of all keys; or,
    3. The number of non-zero keys.

  Lists are useful because they can aggregate event rate data and organize it (i.e., by hostname), then present immediately useful data about that collection of rate data (e.g., the current number of hosts reporting an event, or the total number of those events across the entire environment).

In addition to these data types, the temporal variable server can inject special messages into the log stream when certain threshold conditions are met. Rule engines can watch for these messages to take action without being burdened with constantly querying the state of variables. We provide many examples of how simple and intuitive it is to use these three data types to extract useful information from complex events in streaming log data, and how queued actions can be used to heal the environment without human intervention.

AMD has used VTEC since 2006 to monitor and automate maintenance activities on its large compute grids. Log volumes range up to 10 GB/day with VTEC running smoothly on modest 2-4 core virtual and physical machines. VTEC tracks hardware problems such as disk, CPU, and RAM failures and takes appropriate actions (e.g., shut down/close the broken system and create a ticket). VTEC can monitor the environment for trends that indicate events (e.g., *n* systems are unable to contact *m* NFS filers, so there must be a network problem). Most importantly, VTEC enables autonomic computing by allowing intelligent dispatch of repair jobs in response to detected problems. If these repair jobs fail to work, VTEC can notify humans to take over.

In summary, VTEC is a powerful tool for automating log analysis and event correlation. While there are many other tools that perform similar tasks, VTEC's approach to the problem presents a complete, scalable, and intuitive solution that is able to grow and adjust to virtually any workload.

Outline of Final Paper

1. Introduction
    1.1. The problems facing administrators of large compute environments.
    1.2. Autonomic computing based on event detection.
    1.3. How do we detect events?
    1.4. What types of actions should we take in response to events?
2. Background
    2.1. Existing log analysis and event correlation systems:
        2.1.1. Logwatch
        2.1.2. Swatch
        2.1.3. SEC
        2.1.4. Splunk
    2.2. Problem context at AMD (ca. 2006):
        2.2.1. Increasing number of systems to maintain
        2.2.2. How available tools fell short
    2.3. Problem statement and design goals
3. Architecture
    3.1. Syslog-ng (log routing)
    3.2. Streamer
    3.3. Action server
    3.4. Temporal variable server
        3.4.1. Scalar
        3.4.2. Incrementer
        3.4.3. List
    3.5. Rule engines
4. Rule engine implementation examples
5. Performance
    5.1. Syslog-ng log routing performance and potential bottlenecks
    5.2. Temporal variable server performance and scaling
    5.3. Action server scheduling performance
6. Challenges
7. Conclusion
8. References