Event log messages as a human interface, or, "Do you pine for the days when men were men and wrote their own device drivers?"

A Technical Report submitted to the School of Engineering and Computer Science,
Victoria University of Wellington, New Zealand, Dec 2010
Technical Report: ECSTR10-24

Paul Radford, Andy Linton and Ian Welch
Network Engineering Research Group
School of Engineering and Computer Science
Victoria University of Wellington
P.O. Box 600, Wellington 6140, New Zealand
{paul.radford, andy.linton, ian.welch}@ecs.vuw.ac.nz

ABSTRACT

Computer systems administrators, as a part of their job function, must monitor event logs generated by their systems for signs of failure, impending failure, or security breaches. Many of these systems produce well-defined output that can be easily filtered for important events. Many others, however, are inordinately complex, a situation increasingly common with the advent of multi-tier systems aimed at Internet commerce. Event logs are very often the only system-level output produced by servers, and thus represent the only common denominator across vendors and solutions.

This paper will establish the position that event log messages have shortfalls as an interface for effectively managing such systems, and that a fundamentally different approach is required to improve the situation.

Author Keywords

Event log messages, monitoring, systems administration.

INTRODUCTION

Real-world computer systems administrators are professionals mostly dealing with two broad categories of tasks: they have to be involved with the implementation of new systems; or maintenance of existing systems. This paper focuses on the maintenance effort, in particular the awareness an individual administrator possesses in regards to system state – such an awareness should include a working knowledge of the system components, as well as past, current, and potential future threats to the operational status of those parts. The only existing mechanism by which they can exercise such vigilance, other than point-in-time manual checks of resource usage (such as the disk space remaining, processor load, etc.) is through monitoring of system events via event logs, or their synthesised, visual brethren in the form of 'dashboards' - as portrayed by Few (2006). Event log messages are typically added by programmers for the purposes of accounting, debugging and alerting. 'Syslog' (Request-For-Comments 5424) is by far the most popular application protocol for sending these messages to an aggregation point.

An example of a widespread method of monitoring system event logs is filtering them through regular expressions categorised into varying grades of black- or white-lists. The focus in this work is on methods used in practice rather than approaches such as statistical analysis proposed by academia (e.g. Li et al. (2004); Yamanishi & Maruyama (2005)). The LogWatch (www.logwatch.org) and Logcheck (logcheck.org) projects typify the filtering approach. On the commercial side, NMS (www.nimsoft.com) and Splunk (www.splunk.com), among many others, also provide such functionality as part of their offerings.

The quality of input to any automated system (such as those mentioned above) is critical to the quality of its output, for it is here that the established log-monitoring solutions hit their first problem. In reality, systems administrators are called upon to upgrade software on a daily basis, even if only to implement fixes for identified security risks, and such changes typically reduce the effectiveness of an automated filter. For example, an automated filter is likely to 'break' if a software or operating system upgrade changes the format of any log messages issued by that software.

The evolutionary past of event log development makes interpreting them hard. The key problems here are: incrementally-added features focussed on debugging; a lack of foresight by programmers as to how the logs would be used; and the almost-imperceptible fusing of human assumptions with genuine information in the output (Buckley & Siewiorek, 1995). The overwhelming number of messages alone is an intimidating challenge in many cases, even when visualisation is considered (Tricaud & Saadé, 2010). This is without the additional difficulties of parsing their hugely diverse range of message formats, types and schema (Hammoud, 2009) and performing the semantic reconciliation that is therefore necessary (Caragea & Honavar, 2008). All these factors are detrimental to an administrator's efforts towards understanding the current state of their systems.

LITERATURE REVIEW

The amount of literature on the specific topic of event log system design is scant at best; only two papers could be located which pertain directly to the problem at hand, and both lament this same scarcity of relevant work (Buckley &

```
Feb  3 12:35:56 gateway openvpn-server[2080]: 123.234.147.159:39936 TLS Error: TLS handshake failed
Feb  3 12:36:16 gateway openvpn-server[2080]: 123.234.147.159:40014 Authenticate/Decrypt packet error: packet HMAC authentication failed
Feb  3 12:36:16 gateway openvpn-server[2080]: 123.234.147.159:40014 TLS Error: incoming packet authentication failed from
123.234.147.159:40014
Feb  3 14:38:25 gateway kernel: ATM dev 0: error -110 fetching device status
Feb  3 14:38:44 gateway kernel: ATM dev 0: usbatm_complete: urb 0xdef9f2a0 failed (-84)!
Feb  3 15:12:32 server1 krb5kdc[4196]: TGS_REQ (3 etypes {16 1 3}) 192.168.1.100: PROCESS_TGS: authtime 0,  <unknown client> for
host/server1.example.com@EXAMPLE.COM, Request is a replay
Feb  4 13:06:42 gateway named[23870]: clients-per-query decreased to 10
```

Figure 1. Sample of events that passed through Logcheck's default filter set

Siewiorek, 1995; Etalle et al., 2007). Buckley & Siewiorek performed an in-depth analysis of the event logs produced by VAX/VMS systems and found that even a tightly-controlled product managed by one entity (Digital Equipment Corporation, in this case) suffered from inconsistencies and generally lower-than-expected quality in the event logs, despite being one of the best examples of its day. Etalle et al. took a security/privacy-oriented look at the landscape; their framework allows one to formally ascertain the qualifications of an actual system, weighing up audit integrity and privacy.

The vast majority of literature surrounding the topic of event logging accepts the current state of poor and inconsistent input, yet attempts to enhance the quality of the output via the well-established academic doctrine of incremental, iterative improvement (e.g. Flack & Atallah (2000); Xu et al. (2009)). What appears to be lacking in the literature is any examination of event logging from an HCI perspective, other than when the topic was brushed against by Buckley & Siewiorek (1995) in their discussion on quality.

WHAT DO WE CARE ABOUT?

What do systems administrators, i.e. the audience for any deployed event log system, actually consider to be important? The content emerging from the efforts above is subjected to only the most simplistic of tests (if any), such as a small number of 'domain experts' that offer their opinions on how 'good' or 'useful' the output is to them (e.g. Wallin et al. (2009)), or the subjective opinions of one or two anonymous people (e.g. Saneifar et al. (2009)).

Bearing these points in mind, it can be simplistically deduced, using a behaviouralist mindset, that systems administrators are interested in messages that are *relevant* to their job function. Those messages should be *salient*, in that they contain minimal redundancy and assist in identifying (potential) issues of genuine concern, as highlighted by Flack & Atallah (2000). They argue that messages contain *meaning* that is embedded in the grammatical form of a message, and that it can be imperilled even by the first step of any solution that requires parsing, effectively leading to a 'lossy' input.

A common thread amongst the academic literature is the difficulty involved in interpreting natural-language portions of event log messages. Not only is there the immense challenge of parsing a language itself, but the complications arising from non-standard formats, non-native users of any given language, inconsistencies across and within products, and even spelling mistakes, are far from trivial. Approaches which rely on the "rare equals bad" assumption, result in the abandonment of the actual interpretation of *meaning*. This is not reflected in other well-established disciplines (e.g. industrial chemical plant monitoring, such as Bunch et al. (2004), or failure monitoring in aviation, such as Molloy & Parasuraman (1996)) that tend to focus on nominal states, rates of change and threshold trips. It is important to note that these points are often easily machine-readable. No work could be found which addressed this issue of automatically filtering event log messages on an explicit basis of salience to their human audience.

HOW DOES THIS ISSUE MANIFEST ITSELF?

The role of systems administrator can be severely weighed down with log audits (Wallin et al., 2009). An anonymised sample of real events which passed through a default Logcheck filter set is shown in Figure 1. Logcheck is the most popular log analysis tool (as shown by popcon.debian.org on 23 June, 2010) installed on Debian Linux v5.0; a common server operating system. These events wouldn't be such a problem if the sample represented a daily load, but many of today's systems can easily produce upwards of 1GB of event logs, every single day. This volume can rapidly dull the vigilance of any systems administrator.

Research on filtering has not yet produced any real-world benefit for the vast majority of administrators, as the vast majority of deployed filtering solutions are still based on simplistic regular expression white-/black-lists. The fundamental difficulty lies in the interpretation of the natural-language messages, a rapid and yet unpredictable rate of change, and dealing with spurious messages (i.e. noise) (Yamanishi & Maruyama, 2005). It is our argument that a re-think of the event log paradigm is urgently needed.

DEFINING THE PROBLEM

The environment in which the modern event log system resides cannot be tied down to a tight formal definition. Its input represents a continuous flow that is chaotic, incomplete and inconsistent (Buckley & Siewiorek, 1995). This reflects a natural variation of programming approaches, values, priorities, and the lack of a standard enforced with the proverbial iron fist. The lines in Figure 1, for example, came from four different projects and are all dissimilar. The entire task of re-working an interface to cope with this environment is obviously an endeavour beyond this short paper.

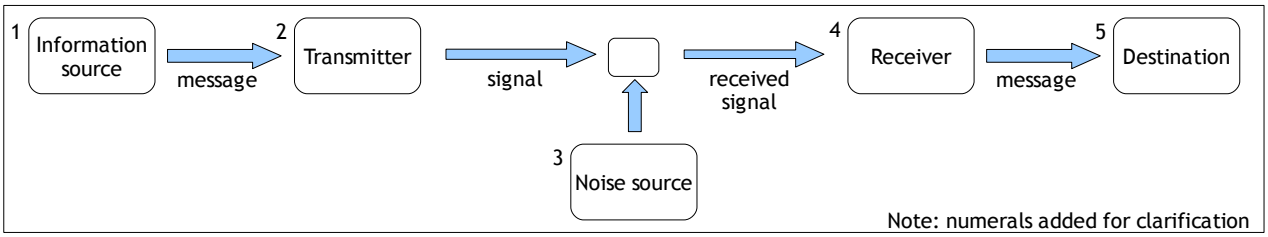

Figure 2: Shannon & Weaver's communications model (Cobley & Jansz, 1999; Shannon, 1949; Weaver, 1949)

Nevertheless, a contribution can be made by examining the rationale and purpose of the system. The automated filtering stage sits between humans and the event log message-generating systems. Humans are poorly adapted to deal with the flood of messages that even one busy system can produce, so our natural coping strategy is to avoid overload by selectively excluding most or all of the surrounding contextual information on offer. We currently do this with filtering solutions such as Logcheck. The resulting lack of context makes discerning the salience of any given message more difficult, while a key contextual area currently being neglected is that of the software code itself. No known tool can extract log-message salience from the contextually-richest source available to us in an open-source environment: the code itself.

EVENT LOGS AS A COMMUNICATION MEDIUM

To clarify our thinking on communicating via log messages, and to begin to tackle the issue of determining salience, we hark back to Shannon & Weaver's seminal communications model. Claude Shannon conceived the model (Shannon, 1949) in Figure 2 to deal with the challenges of radio communication (Cobley & Jansz, 1999). Warren Weaver subsequently used it in the context of semiotics and human communication (Weaver, 1949), helping to formalise the notion that information was encoded in a lossy manner every time it was communicated (Cobley & Jansz, 1999). Here we will attempt to further adapt the model, this time into the semiotic context of event logging.

1. "Information source": The original intention/thought of the programmer and/or program designer; those individuals editing source code.
Limitations: human competence, memory, decision-making.
Information state: rich, informal human thought processes.
Example: VPN project: "Issue a message about the encryption key being corrupted by unknown means."

2. "Transmitter": The encoded thought process of the programmer, i.e. source code which is then (most often) losslessly compiled into binary machine code. Such information is a subset of the original intention of the 'information source', having been formalised and revised into a Turing-complete grammar, following absolute rules of logic and with conditionals that are machine-testable.
Limitations: grammar expressiveness, human language ability (i.e. a native speaker of the applicable natural language, the individual's level of education).
Information state: formalised, testable grammar, probably in a binary form.
Example: VPN source code: "if (key_corrupt) then { ..."

3. "Noise source": The sources of other disparate messages and unrelated messages from our own binary, as well as any packet loss related to the commonplace use of the User Datagram Protocol (UDP) with logging daemons.
Limitations imposed: incomplete information which has possibly been deliberately manipulated or omitted, spurious information (also possibly injected as a distraction tactic).
Information state: pertinent content buried in distractions.
Example: "`Authenticate/Decrypt packet error: packet HMAC authentication failed`", embedded in 100,000 other messages.

4. "Receiver": The system processes which both receive messages, and perform filtering on them. The filtering process may appear out of place, but is similar in its intention to Weaver's "semantic receiver" which he imagined sitting between the receiver and the destination (Weaver, 1949), the difference being that the semantic receiver was to cater to the varying characteristics of each potential receiver device.
Limitations: static or trained filtering tools that must assess salience based on trends, probabilistic factors in neural networks, and/or signatures.
Information state: out-of-context messages, littered with potential false-positives and -negatives, and most importantly, devoid of any encoded indicator of salience.
Example: For the purposes of this paper, the example is the same as the prior stage, except that filtering has taken place.

5. "Destination": The systems administrator. This role is filled by a human who must make critical decisions on a daily basis, often using the information presented to them by the various automated monitoring processes.
Limitations: busy human with other priorities competing for their attention, experience level and language ability of said administrator, uncertainty about the work situation.
Information state: mental impressions formed by information of questionable-salience that has survived filtering.
Example: any action taken (Weaver, 1949) as a result of the VPN message.

Summary and reasoning

The programmer is the person best informed about the situations in which the event log message will be issued – their mental model of the problem domain is the most in-depth of the actors in this context – and originally creates the sequence of conditionals leading to such an issuing.

Thus the human thought processes transition from poorly-defined but context-rich, through testable and context-poor (in the code), to incomprehensible, context-less and buried by the time they reach the systems administrator. That role requires vigilance – a difficult and error-prone task when there might be only one failure indicator amid the logs, and the administrator has other business tasks to complete (Molloy & Parasuraman, 1996).

It is clear that a communication deficit develops between the source and the destination. We would argue this deficit is creating a risk of what James Reason terms "latent failures" (Reason, 1990). Merely learning the nominal state of systems becomes an exercise in futility, as can be seen in many 'dashboards' that seek to visually portray a current state (Few, 2006) – and through such a systemic lack of awareness, failures frequently result (Molloy & Parasuraman, 1996). Much media and public perception focuses on so-called 'active failures' (Reason, 1990), blaming operators or administrators for committing errors. Buckley & Siewiorek (1995) called for higher-quality log files that would provide a more consistent basis for analysis. We would agree with that sentiment, and would add that visualisation tools such as dashboards will not reach their full potential until suitably instrumented outputs and thresholds are available for them to portray.

CONCLUSION

This paper has presented a rationale for improving the quality of issued event log messages. In particular; the need to encode the information in such a way that it can survive the transition from the programmer's intention to its understanding by the systems administrator. Such instrumented outputs can be part of an effort to bring computing into line with other industries, by allowing the notion of a 'nominal state' to be formed, and presenting salience in a machine-readable form. Machines cannot yet assess salience themselves, but we should be able to encode it for them, and provide a basis for valid visualisation at the same time.

The situation needs to improve because log messages are overwhelming systems administrators. Technology designed for an era prior to the popularisation of the Internet can no longer cope with the message volume associated with, potentially, billions of users. Trends are difficult to spot in all this noise – the case that it requires a higher data input quality is a strong one. Improvement will only come when event log messages are considered to be an important feature of each software product and not merely a debugging tool for the programmer. From a human-computer interaction viewpoint, event log messages are currently languishing in the era of the early DOS systems, with arcane syntax and a lack of conclusive documentation – we can do better. Adding metadata is the first step.

REFERENCES

Buckley, M. F. & Siewiorek, D. P.. VAX/VMS Event Monitoring and Analysis. In: FTCS '95, CDP (1995), 414-423.

Bunch, L.; Breedy, M.; Bradshaw, J. M.; Carvalho, M.; Suri, N.; Uszok, A.; Hansen, J.; Pechoucek, M. & Marik, V.. Software agents for process monitoring and notification. In: Proc. 2004 ACM Symposium on Applied Computing, ACM New York, NY, USA (2004), 94-100.

Caragea, D. & Honavar, V.. Learning classifiers from distributed data sources. Encyclopedia of Database Technologies and Applications, Idea Group (2008).

Cobley, P. & Jansz, L.. Introducing Semiotics. Cambridge, England: Icon Books Ltd. (1999).

Etalle, S.; Massacci, F. & Yautsiukhin, A.. The Meaning of Logs. In: Lambrinoudakis, C.; Pernul, G. & Tjoa, A. M. (Ed.), TrustBus 2007, volume 4657 of LNCS, Springer (2007), 145-154.

Few, S.. Information Dashboard Design: The Effective Visual Communication of Data. Sebastopol, CA, USA: O'Reilly Media Inc (2006).

Flack, C. & Atallah, M. J.. Better Logging through Formality. In: Debar, H.; Mé, L. & Wu, F. (Ed.), Proc. RAID, volume 1907 of LNCS, Springer (2000), 1-16.

Hammoud, N.. Decentralized log event correlation architecture. In: Proc. MEDES '09, ACM New York, NY, USA (2009), 480-482.

Internet Engineering Task Force. RFC 5424: The Syslog Protocol (2009).

Li, Z.; Taylor, J.; Partridge, E.; Zhou, Y.; Yurcik, W.; Abad, C.; Barlow, J. J. & Rosendale, J.. UCLog: A unified, correlated logging architecture for intrusion detection. In: Proc. 12th ICTSM (2004).

Molloy, R. & Parasuraman, R.. Monitoring an automated system for a single failure: vigilance and task complexity effects, Human Factors, 38, 2 (1996), 311-322.

Reason, J.. Human Error. New York, USA: Cambridge University Press (1990).

Saneifar, H.; Bonniol, S.; Laurent, A.; Poncelet, P. & Roche, M.. Terminology extraction from log files. In: Bhowmick, S. S.; Küng, J. & Wagner, R. (Ed.), Proc. DEXA 2009, volume 5690 of LNCS, Springer (2009), 769-776.

Shannon, C. E.. Communication in the presence of noise, Proc. IRE, 37, 1 (1949), 10-21.

Tricaud, S. & Saadé, P.. Applied parallel coordinates for logs and network traffic attack analysis, Journal in Computer Virology, 6, 1 (2010), 1-29.

Wallin, S.; Leijon, V. & Landen, L.. Statistical analysis and prioritisation of alarms in mobile networks, International Journal of Business Intelligence and Data Mining, 4, 1 (2009), 4-21.

Weaver, W.. Recent contributions to the mathematical theory of communication, The Mathematical Theory of Communication, 1 (1949), 93-117.

Xu, W.; Huang, L.; Fox, A.; Patterson, D. & Jordan, M.. Online System Problem Detection by Mining Patterns of Console Logs. In: Proc. ICDM '09, IEEE Computer Society (2009), 588-597.

Yamanishi, K. & Maruyama, Y.. Dynamic syslog mining for network failure monitoring. In: Proc. 11th ACM SIGKDD, ACM New York, NY, USA (2005), 499 - 508.