Towards Structured Log Analysis

Dileepa Jayathilake
99X Research
99X Technology
Colombo, Sri Lanka
dilj220@gmail.com

Abstract— **Value of software log file analysis has been constantly increasing with the value of information to organizations. Log management tools still have a lot to deliver in order to empower their customers with the true strength of log information. In addition to the traditional uses such as testing software functional conformance, troubleshooting and performance benchmarking, log analysis has proven its capabilities in fields like intrusion detection and compliance evaluation. This is verified by the emphasis on log analysis in regulations like PCI DSS, FISMA, HIPAA and frameworks such as ISO 27001 and COBIT.**

In this paper we present an in depth analysis into current log analysis domains and common problems. A practical guide to the use of few popular log analysis tools is also included. Lack of proper support for structured analysis is identified as one major flaw in existing tools. After that, we describe a framework we developed for structured log analysis with the view of providing a solution to open problems in the domain. The core strength of the framework is its ability to handle many log file formats that are not well served by existing tools and providing sophisticated infrastructure for automating recurring log analysis procedures. We prove the usefulness of the framework with a simple experiment.

Keywords-log analysis; structured logs; log data extraction; log management; mind map

I. INTRODUCTION

Software log files can be huge assets to an organization if utilized properly. They contain vital information pertaining to most user and system actions. The challenge is how can an organization make sure that proper logging is implemented in required places and the logs are collected and analyzed to derive information that can add value. More often than not, the information dwelled within the huge collection of logs cannot be found anywhere else. This speaks for the importance of implementing proper discipline within an organization with regards to log analysis.

One major use of log files is in software troubleshooting [1]. Importance of having elegant logging is highlighted when applications malfunction in production because the log can contain many inner details on application operation that can be of immense value during troubleshooting. However, a typical log file contains vast amounts of information and digging into the information of interest is no simple job. This requires analysts with acquaintance in the log format. Even with familiarity, log analysis requires a significant time, which ultimately converts into cost. One additional drawback with completely manual analysis is the inability to take full advantage of recurrence in certain analysis procedures. A platform providing tools for automating these can save a lot of money.

There are quite a number of commercial tools that attempt to automate a part of the log analysis procedure. Most of them come as enterprise log management solutions including log collection, remote log access, log archiving, indexing, searching, reporting, alerting, dashboards, etc. Log analysis constitutes a single feature in these solutions. Almost all the existing tools consider logs as sources of unstructured information and most of them assume line logs. However, in reality, there are many log files that contain information chunks spanning well over one line. XML logs are one example. Extracting meaningful information from them requires being aware of their structure. On the other hand, line logs too exhibit a structure and their analysis can benefit from a structure-oriented methodology. Despite of its benefits, structured log analysis encounters many challenges too. Changes in the log file format over time, log file corruptions, inconsistencies and performance overhead are some of them.

This paper presents a thorough study on the log analysis domain followed by an insight into popular commercial tools. After that we detail the current state of an ongoing work on building a framework for structured log analysis. In Section II we discuss the popular uses of log analysis. Section III describes a study into the domains where log files are mostly used and research being done so far. This is followed by an insight into common problems in log analysis as a practice in Section IV. In Section V we state the requirements from a log analysis system according to our understanding on the subject. Section VI is a brief on few widely used log analysis tools. In Section VII we stress the importance of structured log analysis. Section VIII contains details about the new framework, which is followed by the description of an experiment we did with the framework to prove its value in Section IX. Section X concludes the work and Section XI presents future work.

II. LOG ANALYSIS IN USE

Software application logs are extensively used in troubleshooting. When something goes wrong in production, the most informative source a developer gets for investigating, more often than not, is the application log file. Usual practice is to manually analyze the log, whereas few organizations invest in developing tools to automate a part of the analysis process. Testers use application logs to verify the functional conformance of software with the specification [2]. They compare the logged program output with predetermined reference output for a given input. Time logs are used in some cases for software profiling and benchmarking.

System administrators have been among the heaviest users of log files. They utilize logs to monitor system health and to timely detect any undesired behavior in the system. Because most system and user actions are recorded in log files they become an important component in IT accountability. Furthermore, system breaches can be more easily identified by using security warning logs. Program and system warnings are generally used to recognize power failures, low memory conditions, etc. Some organizations use logs as a verification tool for compliance with certain regulations too.

One different domain for log file use is analyzing data from a collection of log files statistically. The results can be used to determine usage patterns of software, application bottlenecks, hardware requirements, etc. Behavioral patterns are used for determining reliability and performance while identification of causality trends help data mining.

III. LOG ANALYSIS DOMAINS AND PAST RESEARCH

Web server logs have been the most popular log file type for analysis. Usage spreads over a wide spectrum of aspects including access patterns, traffic patterns, client platforms, user agents, client technologies, attacks, HTTP errors, etc. Network logs are another type of log files that is subjected to analysis mainly for performance analysis and network activity monitoring. Next are the security logs that are often used for intrusion detection or for analyzing a compromised honey pot. It is proven that log files serve a critical purpose in an investigation into the cause and origins of an attack [3]. System error logs provide vital information for predicting failures and faults. For example, the work described in [4] is able to predict 93.7% hardware failures while [5] presents a framework for assessing dependability of supercomputers by analyzing error logs.

In addition to these special log file types most applications generate at least one log. Usually any software system spits a console log. Some organizations develop tools that can assist in analyzing their logs. One popular use for application log analysis is anomaly detection. For example, an electronic trading system can utilize log files to identify market manipulations. The work presented in [6] uses machine learning to detect operational problems in applications after processing its text log file. It uses application source code to detect the structure of the log file. [7] describes a method to track anomalies by finding less frequent patterns using principle component analysis. The graphical analysis method proposed in [8] reduces the analysis time by 80% for the 25,000 lab runs per year of the 5ESS system of AT&T.

IV. LOG ANALYSIS CHALLENGES

Lack of a widely adopted standard for logging is one of the burning problems in log analysis. Because of this, every log file has its own proprietary format. An IETF standard [9] ‘Universal format for Logger Messages’ was introduced few years back, but has expired without a natural successor. An XML based approach has been suggested as the next step. However, no draft is available as yet [1]. Syslog [10] emerged as a standard for system event logging. However, it is not suitable for certain logs such as HTTP logs and network logs like Cisco Netflow logs because of the overhead it generates. It is also not rich enough for application logs that need structural recording of information.

Another problem is with the consistency issues with the format of flat text logs. Added to this is the corruption of log files due to various reasons such as bugs in applications, a rouge thread overwriting the log, problems with file locks, memory corruptions and disk failures. Corruptions result in erasing parts of a log file, mixing up multiple log entries, presence of log entries in wrong order and garbage in the middle of log files.

Often, amount of information logged by the application confronts analysts with few challenges. Sometimes log entries do not contain sufficient amount of details for the analyst whereas some other less useful log entries that highly repeat clutter the log file. Apparently the problem stems from incorrect judgments of developers regarding the importance of log entries. Some trivial events may get logged as critical errors while severe issues being logged as non-critical, hindering the log analysis from getting the benefit of filtering log entries according to the severity.

More often than not, the format of log files and the content being logged evolves with products. This brings an additional challenge to log analysis tools. Furthermore, the huge size of log files also has been causing problems. The unstructured nature of text logs makes an analysis procedure difficult because of the need for a separate pass every time some new information is required. Due to these performance-related issues, online log analysis systems are pushed for compromising detection accuracy in favor of less computational complexity. Requirement of high expert involvement for rule based log analyzers and the need for native tools to analyze binary logs are two other problems.

V. EXPECTED FEATURES FROM A LOG ANALYSIS SYSTEM

After studying usage scenarios and problems associated with log analysis we can compile a list of features that a good automatic log analysis system should provide. First, it should be capable of handling log files with various structure and syntax. It should recognize common constructs that appear in log files such as timestamps, IP addresses, port numbers, common error and warning numbers, etc. At the same time it should possess intelligence to detect and bypass intermittent corruptions in a log file without causing the analysis to terminate prematurely. An analyst will also expect the system to be able to read from various log file sources such as the local file system, network streams, remote locations and databases. The system should recognize flat text log files, XML log files and binary logs. Ability to generate friendly user interfaces containing extracted and processed log data will also be a decisive factor. Furthermore, it is highly important to provide a mechanism for analysts to automate recurring analysis patterns. Such a system will grow a rich knowledgebase over time with reusable automatic analysis routines. The system should either contain or provide an interface to add knowledge on intrusion detection and standard compliance. Another feature with practical significance is to store extracted log data after a single pass so that subsequent queries for data can be executed faster.

VI. COMMERCIAL LOG ANALYSIS TOOLS

This section provides a list (non-exhaustive) of available log analysis tools along with a brief description highlighting their key features.

Splunk – This is one of the most popular commercial log analysis tools [11]. It comes as a native application for each of the popular platforms. It provides strong search capabilities within log files. Log files from many different sources can be integrated into an analysis. Splunk is capable of identifying common constructs appearing in logs such as timestamps. In addition to indexing logs based on automatically detected log entries it provides functionality for users to create custom indexes too. Indexed log files can be saved as templates so that the index can be used for a similar log file later. Analysis results are displayed in a dashboard with many feature-rich user interface controls. Although Splunk can handle any kind of text log file, it is appropriate for analyzing line logs. It comes with a free version (without expiration) with an upper limit to the total size of log files analyzed in a day.

LogRhythm – This is another widely used commercial tool for log analysis [12]. Its' important features are the ability to analyze a huge number of logs at once, automatic detection of interesting log entries, risk-based prioritization of log events, customizable rules, alerts, real-time log monitoring, normalization between different time zones, configurable charting, ability to save investigation data and file integrity monitoring. It has built in capabilities to evaluate log compliance with a number of standards. In addition it has strong intrusion detection capabilities too. In summary, LogRhythm is a sophisticated, enterprise solution.

ArcSight Logger – This is a tool for event log collection and reporting [13]. Being a commercial tool ArcSight Logger has the capability to handle event log messages from many different client platforms. The messages can be sent in a variety of protocols. The tool can handle terabytes of log data efficiently. It classifies log events so that different syntax used across platforms for same kind of log data is made transparent to the user. Searching is possible using plain text, regular expressions or indexed text. ArcSight Logger provides strong reporting capabilities too. Reports can be exported to various formats before saving. Alerts can be defined based on reports. The tool comes with a free evaluation version.

loggly – This provides a cloud based log management system [14]. Log files from various sources can be collected to a central place in cloud for analysis. Log entries can be searched and be viewed in a dashboard. Historic data can also be viewed. The tool supports alerting. A free trial version is provided.

loglogic – This is another log management infrastructure tool with the capability to collect logs from either enterprise or cloud and provide analysis [15]. Main features include ability to handle data in ranges of petabytes, advanced searching capabilities, dynamic dashboard, detailed reports, alerts, forensics engine, log retention management and compliance reporting.

AWStats – This is a free tool that can analyze logs generated by web servers like Apache web server, Internet Information Server, WebStar and some other proxy, wap, ftp, streaming and mail servers [16]. It is a command line tool that uses Perl scripts. It provides usage statistics, user origin information, popularity of pages, HTTP errors, number of favorites on the site, worm attacks detection, etc.

SecureVue – This is a situational awareness platform that utilizes logs from various types of assets in an organization such as hosts, network and security devices, applications and databases for capturing important security information [17]. It provides compliance with many security standards.

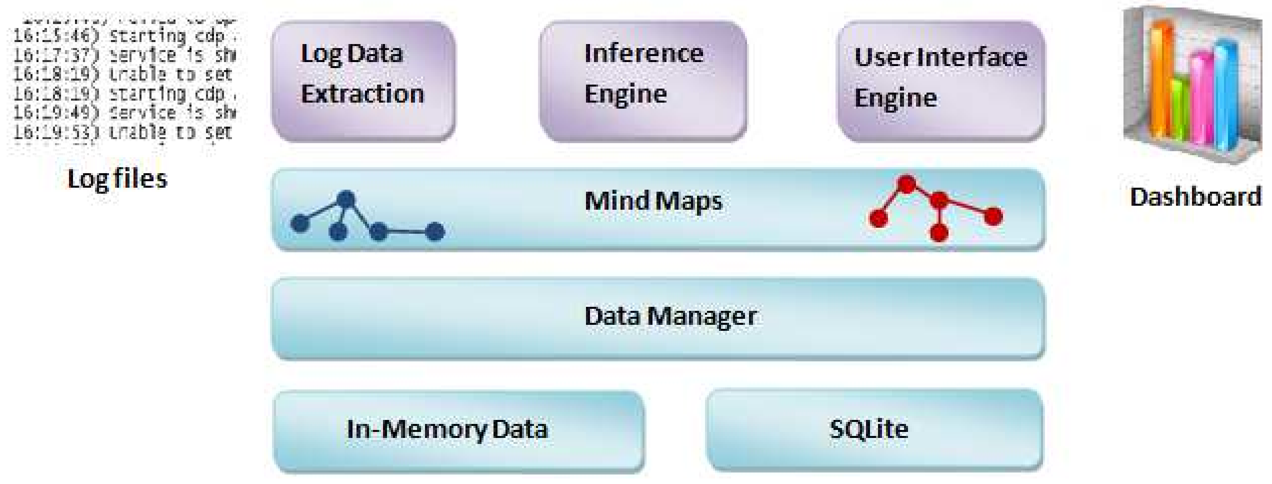

```
16:15:46) starting cdp
16:17:37) service is shu
16:18:19) unable to set
16:18:19) starting cdp
16:19:49) service is shu
16:19:53) unable to set
```

Figure 1. Major framework components and input / output

VII. IMPORTANCE OF STRUCTURED LOG ANALYSIS

Almost all commercial log analyzers we studied assume line logs and do not pay much attention to the structure of a log file. Most of them provide a search facility on unstructured log data. However, in practice, lot of software generates XML logs that are well structured and some others create non-XML logs that still have a prominent structure. An example for this second type is the Electronic Trading System log we present in section VIII. Extracting useful content from this type of logs requires accounting for their structure because data organization is significantly different from line logs. Data semantics highly depends on relations between neighboring data elements. On the other hand, line logs too have a format that can be defined to a certain degree of accuracy. A data extraction mechanism that is aware of this format will be able to pick more meaningful information from them than one that relies on mere matching. If a mechanism can be devised to express the formats of all log files involved during an analysis procedure, an analyst can ensure that all relevant data along with relationships between them are captured from the correct regions in log files. It provides both completeness and contextual correctness.

Search options provided by unstructured log analysis tools return a result set without a reference to the context of each result. This incurs the need for manually inspecting the result set in order to do a cherry picking on relevant results for further analysis. In contrast, the context-aware result set generated by structured analysis enables analysts to automate the procedure by implementing the result picking logic utilizing the context information. Therefore structured analysis favors automation.

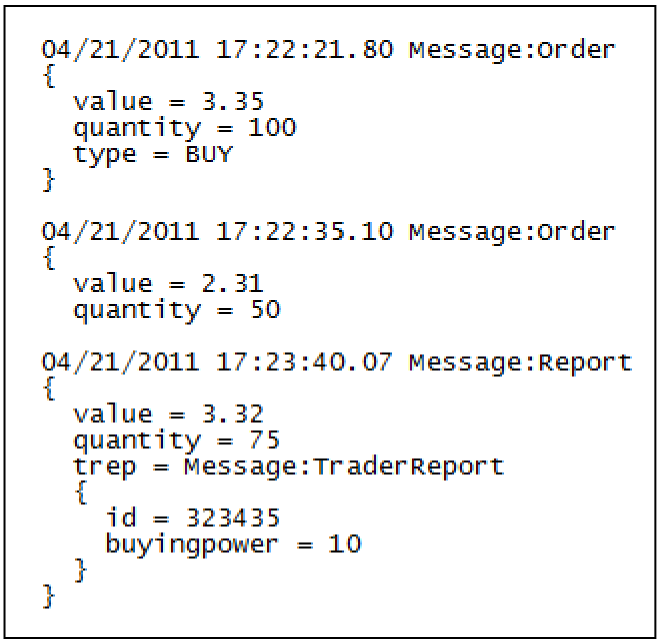
VIII. STRUCTURED LOG ANALYSIS FRAMEWORK

In our previous work [18] we presented a mind map based framework for structured log file analysis. It came up with a scripting language for log data extraction, log data processing and presenting results. After that, in another work [19] we presented a specification for a declarative language, which is more appropriate for the log data extraction phase. Here we describe an improved version of the same framework. Architecture of the framework is depicted in Figure 1.

Log Data Extraction module uses a log data format declaration language, which is an implementation for the specification we presented earlier [19]. It is compliant with Simple Declarative Language [20]. We call the new language “Log Data Extraction Language” (LDEL). The language is capable of expressing all log file formats we have found after analyzing 20 different log file types. The parser is flexible so that even the formats that appear to be inconsistent can be captured. The language is also resilient to most log file corruptions. When the parser encounters a corrupt area in a log file it has the ability to recover parsing operation from the next correct log entry. Before starting the analysis procedure, there should be a declaration for the format of each log file type that is going to get involved. Parsing a log file with respect to its format declaration results in a tree data structure containing extracted log data.

Inference Engine uses the tree structures generated by the Log Data Extraction module as its input. The main technology used inside the Inference Engine is the mind map based procedural language we presented in [18]. We call it “Mind Map Language” (MML). Analysts are supposed to write scripts in MML for deriving conclusions from log data. The language is optimized for this purpose. Data is manipulated within MML as mind maps (trees). The language has configurable syntax so that it can be localized. Numerous convenience routines are provided for extracting data from and processing trees. The output of the Inference Engine module is a set of trees containing inferences made. User Interface Engine uses these trees and presents them in a dashboard. Projecting tree data into the dashboard is also done using MML. There should be another set of MML scripts for doing this.

As mentioned earlier, the core data structure used in each module in the framework is tree. Data Manager module is responsible for keeping these data within the system. A generic data interface is provided in this module so that different implementations for data management can be plugged in. Each implementation will have its own positives and negatives. Depending on the requirements of a particular analysis procedure, appropriate data management mechanism can be selected. At this stage, two data management mechanisms are provided in the framework. In-memory data handling was what we had in the version we presented in [18]. After that we developed a relational database model that uses SQLite. SQLite is selected because it can be used in almost all popular platforms including mobile operation systems. In addition is a lightweight, embedded database that incurs less management overhead. In-memory data structures are fast, but are limited in size by the availability of memory. They are volatile. Relational database, on the other hand, provides persistence so



```
04/21/2011 17:22:21.80 Message:Order
{
  value = 3.35
  quantity = 100
  type = BUY
}

04/21/2011 17:22:35.10 Message:Order
{
  value = 2.31
  quantity = 50

04/21/2011 17:23:40.07 Message:Report
{
  value = 3.32
  quantity = 75
  trep = Message:TraderReport
  {
    id = 323435
    buyingpower = 10
  }
}
```

Figure 2. Trading system log file used in the experiment. It contains messages that contain attribute-value pairs. The second message is corrupted.

that the framework can contain historical data.

The framework also includes few convenience tools to assist analysts when using it. One tool is the Application Event Logger. It is a simple tool that writes any string to the log file along with the time. Analyst can use it to mark important events during the testing phase. For example, it can be used in a troubleshooting procedure where the analyst wants to relate an unintentional behavior in the application to the exact event that causes it. The analyst can run the application under required monitoring tools that record information regarding the target aspects in the application (e.g. disk access, low level operating system call failures, memory corruptions). Parallel to this he can run the Application Event Logger and record strings describing start and end of important application events. At the end the analyst has a time log generated by the Application Event Logger and logs generated by monitoring tools that contain details on failures and warnings along with the time they occurred. Then he can correlate these logs using MML scripts to figure out the event(s) that cause a particular malfunctioning.

Next convenience tool is the Log Pattern Highlighter. It is capable of identifying common constructs in log files such as timestamps, IP addresses, port numbers, common error and warning numbers and recurring strings. This comes very handy when trying to identify and declare the format of a new log file type. Author of the LDEL script that declares the format of a log file can use this tool to initially locate structural parts in the file. It can also be helpful during the analysis for identifying log entry patterns.

IX. EXPERIMENT AND RESULTS

We conducted an experiment log analysis procedure on a single log file that contains structured data. An obfuscated part of the log file is shown in Figure 2. It is generated by a server in an automatic trading system and contains all the input and output messages for the process. Some of these messages are requests received by the server from other components in the system. Responses to those requests are again messages. Request and response can be mapped using the unique request id, which is an integer. Then we can use the fields in the response messages to determine whether the request was successful. After that we plot the number of successes and failures with respect to time. This graph can be used to judge how well the server is doing. We were able to achieve this by using a LDEL declaration for the log format with message as the failure recovery point and one MML script to output a tree containing a sub tree for each one-minute interval, which in turn holds a child node for each request made within that minute along with a flag denoting whether it succeeded. We used another MML script to use this tree to generate a graph in HTML. SQLite data manager was used because of the high volume of data being processed (more than 10,000 messages). The operation was completed within 2 minutes in a 2.4 GHz Quad Core machine and the virtual memory usage stayed within 100 MB.

We did input the same log file into Splunk [11], which is a leader in commercial log analysis tools. Because it assumes a line log, the tool did not provide much useful information other than identifying the timestamps within the file. No mechanism was found within the tool to automate the above analysis procedure because of the unstructured handling of log data.

X. CONCLUSIONS

Existing log data analyzers solve a subset of the problems encountered by log analysts. They do a good job in collecting log files from different sources, indexing log data, identifying common constructs in log files, searching logs and deriving conclusions (standard compliance, intrusion detection) with known log formats. However, they mostly ignore the power of structured data extraction and processing. This hinders expert analysts from automating recurring analysis procedures. It also reduces the level of visibility into log data during an analysis.

The new framework provides a simple way for structural extraction of log data by expressing the format of a log file in a declarative language. It provides a high degree of resilience to log file corruptions. It is capable of expressing XML logs, other structured text logs and line logs too. Mind map based scripting language included in the framework is proven to be a handy tool for processing extracted log data. It creates a convenient infrastructure for analysts to automate log analysis procedures. Flexibility of the data management module empowers analysts with the freedom to select the appropriate data handling mechanism depending on the problem.

Though MML proves to be a strong platform for log data processing, it is not a good fit for the User Interface Engine. Generating a simple HTML graph required a long and complicated MML script. Dashboard generation is apparently the weakest aspect in the framework. Relational databases do not seem to be appropriate for handling logs that have log entries with highly variant attributes. For example, in the message log mentioned in Section VIII there are hundreds of message types where each message has its own set of fields. Some messages have only a few fields whereas some others contain thousands. Nested messages (messages containing other messages as field values) further add to this complexity. Fixed database schema used in relational databases is inefficient in handling this. What seems to be appropriate is a NoSQL database since NoSQL accepts varying table schema.

XI. FUTURE WORK

An important future work is to add more log management capabilities to the framework so that it can collect logs from various sources before analyzing. Real time analysis of logs is another desired feature. Furthermore, the framework requires more format declarations for commonly used log file types for it to be a tool with significant practical use.

Addressing the inhomogeneity problem in the database is also another important future enhancement. Since the nodes in data trees can have vastly different sets of attributes, the framework should support a complying database model for such cases. Our suggestion is to use a NoSQL database model such as BigTable (each data entity can have its own schema) or document-oriented model (database can have documents that contain key-value pairs).

A future enhancement with paramount importance is to develop a new model for User Interface Engine. The model should be able to create user interfaces that meaningfully represent information kept in trees inside the framework. We suggest a HTML 5 based implementation because of platform neutrality.

REFERENCES

[1] J. Valdman. Log file analysis. Technical Report DCSE/TR-2001-04, Department of Computer Science and Engineering (FAV UWB), 2001.

[2] J. H. Andrews. Theory and practice of log file analysis. Technical Report 524, Department of Computer Science, University of Western Ontario, May 1998.

[3] A. Neville, IDS Logs in Forensics Investigations: An Analysis of a Compromised Honeypot. March, 2003.

[4] T. T. Y. Lin and D. P. Siewiorek, “Error log analysis: statistical modeling and heuristic trend analysis,” Reliability, IEEE Transactions on, vol. 39, no. 4, pp. 419–432, 1990.

[5] C. Di Martino, D. Cotroneo, Z. Kalbarczyk, and R. K. Iyer, “A Framework for Assessing the Dependability of Supercomputers via Automatic Log Analysis.”

[6] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, “Detecting large-scale system problems by mining console logs,” in Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, 2009, pp. 117–132.

[7] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, “Online system problem detection by mining patterns of console logs,” in Data Mining, 2009. ICDM’09. Ninth IEEE International Conference on, 2009, pp. 588–597.

[8] S. G. Eick, M. C. Nelson, and J. D. Schmidt, “Graphical analysis of computer log files,” Communications of the ACM, vol. 37, no. 12, pp. 50–56, 1994.

[9] “draft-abela-ulm-05 - Universal Format for Logger Messages.” [Online]. Available: http://tools.ietf.org/html/draft-abela-ulm-05. [Accessed: 26-Feb-2012].

[10] "Security Issues in Network Event Logging (syslog)." [Online]. Available: http://datatracker.ietf.org/wg/syslog/charter. [Accessed: 28-Feb-2012].

[11] "Splunk for Application Management." [Online]. Available: http://www.splunk.com/web_assets/pdfs/secure/Splunk_for_Application_Management.pdf. [Accessed: 26-Feb-2012].

[12] "Log Analysis." [Online]. Available: http://www.logrhythm.com/Products/LogandEventManagement/LogAnalysis.aspx. [Accessed: 26-Feb-2012].

[13] "ArcSight Logger Review." [Online]. Available: http://review.techworld.com/security/3234293/arcsight-logger-review. [Accessed: 26-Feb-2012].

[14] "Application Intelligence." [Online]. Available: http://loggly.com. [Accessed: 26-Feb-2012].

[15] "Log Management Infrastructure." [Online]. Available: http://www.loglogic.com/products/log-management/log-management-infrastructure. [Accessed: 25-Feb-2012].

[16] "AWStats official web site: Free realtime log file analyzer to get advanced statistics." [Online]. Available: http://awstats.sourceforge.net. [Accessed: 25-Feb-2012].

[17] "Using SecureVue^{TM} from elQnetworks for log management." [Online]. Available: http://www.eiqnetworks.com/resources/eIQnetworks_SolutionBrief_LogManagement.pdf. [Accessed: 20-Jan-2012].

[18] D. Jayathilake, “A mind map based framework for automated software log file analysis,” in International Conference on Software and Computer Applications, Kathmandu, 2011, pp. 1-6.

[19] D. Jayathilake, “A novel mind map based approach for log data extraction,” in 6th IEEE International Conference on Industrial and Information Systems, Peradeniya, 2011, pp. 130-135.

[20] "Simple Declarative Language: A simple universal language for describing typed data in lists, maps and trees." [Online]. Available: http://sdl.ikayzo.org/display/SDL/Home. [Accessed: 15-Jan-2012].