John R. Reuning. Applying Term Weight Techniques to Event Log Analysis for Intrusion Detection. A Master's Paper for the M.S. in I.S. degree. July, 2004. 60 pages. Advisor: Robert Losee.

Strong similarities exist between intrusion detection and information retrieval. This paper explores the application of probabilistic information retrieval techniques to log analysis for host-based intrusion detection. Using information retrieval techniques may yield significant improvements to the performance of intrusion detection systems. This paper provides a brief review of current relevant research in intrusion detection and log analysis, introduces information retrieval methods appropriate for intrusion detection, and evaluates the effectiveness an experimental log analysis system using the 1999 DARPA Intrusion Detection Evaluation data sets. The system is based on Bayesian probability theory and uses a TF-IDF term weight measure to identify anomalies.

Headings:

Computer Security

Information Retrieval

APPLYING TERM WEIGHT TECHNIQUES TO EVENT LOG ANALYSIS FOR INTRUSION DETECTION

by
John R. Reuning

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science.

Chapel Hill, North Carolina

July 2004

Approved by

____

Robert Losee

Table of Contents

List of Tables

List of Figures

Introduction

In the field of information security, intrusion detection refers to the process of identifying unauthorized access to computer systems or electronic data. Several methods exist for detecting intrusions: manual inspection of a system, audit log processing, event log analysis, file integrity checking, host-based intrusion detection, and network intrusion detection. Each method has advantages and pitfalls, either in the amount of human attention required to set up and maintain the system, the accuracy of incident detection, or failure of the system to exclude false positives.

Intrusion detection systems differ on two axes: 1) they can be either network-based or host-based and 2) they can use rulesets or anomaly detection to identify events. A network intrusion detection system is either a dedicated server or network device that examines raw network traffic. A host-based intrusion detection system can run on a server or workstation to watch tasks or files for signs of unauthorized activity. For intrusion identification, systems use rulesets, patterns for matching signatures of known problems; anomaly detection, establishing a baseline for normal activity and flagging deviance; or a combination of these two methods.

Log analysis usually refers to report generation for human inspection but is sometimes used by host-based intrusion detection systems as a data source for identifying problems. System or event logs can be generated by almost any active software running on a computer and in Unix/Linux are frequently managed by an application named syslog

(Nemeth, Snyder, Seebass, & Hein, 2001). Log message output reflects the state of the operating system or application at a given time. Messages vary in their level of severity, depending on the software or application design. For example, syslog provides 8 categories ranging from “emerg” to “debug” for differing levels of event severity. Audit logs differ somewhat from event logs. Audit logs usually consist of transactions or process call traces instead of application error or informational messages. The data contained in process audit logs is generally more structured than that of event logs such as those produced by syslog.

Much research is being conducted on intrusion detection systems, especially ones that are network-based. In addition, many experimental systems operate on process audit logs. However, little attention is directed toward event log analysis for intrusion detection. Commercial and open source products exist that match known patterns in log messages and alert a system administrator when a match occurs. This method works well, but it requires human attention and expertise.

Applying a probabilistic information retrieval model (TF-IDF term weighting and relevance judgments) to host-based event log analysis could be an effective means of identifying security incidents. This paper addresses the following question: How effective are TF-IDF weight calculations on event log messages in identifying anomalies for intrusion detection? If a system can be developed to accurately highlight problems without regular pattern updates or specialist intervention, large and small IT organizations could save significant system administrator and security analyst resources.

Relevant Literature

Anomaly-based intrusion detection

Researchers agree on the basic goal of an intrusion detection system. Wagner and Soto (2002) state it thus: "The goal of an intrusion detection system (IDS) is like that of a watchful burglar alarm: if an attacker manages to penetrate somehow our security perimeter, the IDS should set off alarms so that a system administrator may take appropriate action" (p. 255). The intrusion detection system assists information security professionals in maintaining data integrity.

Intrusion detection systems have traditionally been based on matching patterns in raw network traffic. More recently, however, activity has focused on profiling activity of a system or network and detecting anomalies. Wagner and Soto (2002) are critical of traditional, pattern-matching intrusion detection systems. They state that "[s]ignature-based schemes are typically trivial to bypass simply by varying the attack slightly, much in the same way that polymorphic viruses evade virus checkers" (Wagner & Soto, p. 255). They contend that anomaly detection is more resistant to the evasion tactics of attackers (Wagner & Soto).

Anomaly detecting intrusion detection systems are not without their own failings.

> The current model-based approaches all share one common problem: a truly robust intrusion detection system must solve a special case of the machine learning problem, a classic AI problem. That is, to prevent false alarms, the IDS must be able to infer, from statistical data, whether the current execution of the

> system is valid or not. The false alarm rate of present systems is a major problem in practice. (Wagner & Dean, 2001, p. 1)

System administrators and security analysts want an IDS that requires little human maintenance, is accurate in identifying problems, and has a low rate of false positive generation.

Implementation techniques

Various machine learning and data-mining techniques have been used for baselining normal system activity and identifying abnormalities. Zanero and Savaresi (2004) tested a two-stage anomaly detection system. The first stage (the focus of their research) used unsupervised learning to cluster input data, reducing the information to a manageable size (Zanero & Savaresi). The second stage applied statistical and machine learning techniques on the stage-one results to identify anomalies (Zanero & Savaresi). Zanero and Savaresi compared three unsupervised learning techniques: K-means, S.O.M. (Self-Organizing Map), and PDDP (Participatory District Development Programme). In their experiments, they found that the S.O.M. algorithm performed the best for clustering of input data (Zanero & Savaresi).

Wagner and Soto (2002) examined a system that kept track of series of system calls. The system operated as a finite-state automaton, and like most host-based IDS's, it "learn[ed] the normal behavior of applications and recognize[d] possible attacks by looking for abnormalities" (Wagner & Soto, p. 256). A study by Sequeira and Zaki (2002) applied clustering algorithms to create an intrusion detection system. Their system achieved an 80% accuracy rate for detecting intrusions with a 15% false positive generation rate (Sequeira & Zaki).

The detection algorithms employed by Ye, Xu, and Emran (2000) and Liao and Vemuri (2002) are similar to the methods explored in this paper. Ye et al. designed and tested a system that used Bayesian networks with undirected links between nodes to identify anomalies in Unix process audit log data. The system based its classification of processes on the probability of co-occurrence of audit events in a sliding window of events (Ye et al.). The study concluded that the "Bayesian network has a promising performance in intrusion detection" (Ye et al., p. 178). While detailed information was provided on the Bayesian network design, the conclusions were somewhat vague (Ye et al.). Little data was given on which types of intrusions were correctly or falsely identified (Ye et al.).

Liao and Vemuri used term weights in their calculations. Their experiment treated system calls in BSM audit logs (e.g. open, close, and mmap) as terms and applied the k-Nearest Neighbor categorization technique to identify anomalous processes (Liao & Vemuri). In fact, Liao and Vemuri made use of a TF-IDF weight in their categorization system. Their strategy differs slightly from the one used in this paper, however. Their method grouped audit log entries by process, not as discrete documents (Liao & Vemuri). By focusing on a TF-IDF calculation and treating log entries as individual documents, this paper uses a different term weight method for identifying anomalous log entries[1]. In addition, this paper introduces the use of event log messages as a data source.

[1] Specific differences between the kNN method used by Liao and Vemuri and the TF-IDF weight method in this study will be addressed in the following section.

Event log analysis

Unlike anomaly-based intrusion detection, the topic of log analysis has received little research attention. Commercial and open source tools are used for generating log file reports (http://www.loganalysis.org), but these reports must still be examined by a specialist. In addition, many log analysis tools report at regular intervals, such as daily or hourly, not in real time as do intrusion detection systems. Muscat (2003), however, outlines a framework for building an intrusion prevention system based on patterns generated from processing event log data. He suggests that examining system log entries in conjunction with a finite state machine will yield effective patterns for a network-based intrusion detection system (Muscat). Muscat acknowledges the usefulness of log message data for intrusion detection, but specific research is needed in this area.

Two log analysis tools exist that differ from the traditional pattern matching approach, SL2 and SIDS. SL2 presents itself as an anomaly detecting system. The documentation describes it thus: "This script will scan the directory where your logfiles reside and report everything that it finds with the exception of the those expressions found in the ignore file, scanlog.ignore" (Fulton & Hoffman). While SL2 detects anomalies, it still uses predefined patterns to flag messages. This requires an expert to configure the system with patterns. The second log analysis tool, SIDS (Statistics-based Intrusion Detection System), goes beyond pattern matching. Its primary target is web transaction logs, and it uses a simple form of thresholds to identify anomalies. According to a presentation by the system's designer, though, it is prone to resource overutilization and a large number of false positives (Russell, n.d.).

Event log messages describe system or application state. These messages are written by application developers and are similar in style and in form to source code comments. The application of information retrieval techniques to computer source code relates closely to event log analysis. Ugurel, Krovetz, Giles, Pennock, Glover, and Zha (2002) experimented with automatic classification of source code archive documents using support vector machines. Their system tried to assign source code files to application categories and achieved an accuracy level as high as 86% with a relatively low frequency of false positives (Ugurel et al.). They included programmer comments in their tests, which are similar to the text found in log message output (Ugurel et al.). Both programmer comments and log message output text are created by software developers and tend to be concise and technically specific. Among the conclusions of the article is that term frequency might have improved their results (Ugurel et al.). The system proposed in this paper examines event log messages as documents, much as Ugurel et al. do with source code.

Input data

For conducting intrusion detection experiments, researchers used either a dedicated test environment or live systems to produce input data. A test environment is typically a localized network or group of servers that is isolated from the Internet or other publicly accessible networks. Data used for training an IDS simulates that which would be generated by normal use of a computer system or group of servers. The advantage of using test environment input data is that one knows exactly which attacks are performed against a test system and at what time. Uncontrolled factors are greatly reduced.

However, training or background network traffic and usage patterns in a test environment can only approximate those of a real world environment.

Both the 1998 and 1999 DARPA Intrusion Detection Evaluations used data generated in a test environment. According to Haines, Rossey, and Lippmann (2001), the "datasets supported the evaluation, and contained extensive examples of normal and attack traffic run on a realistic testbed network" (p. 1). The test data was generated using a tool developed by MIT called the Lincoln Adaptable Real-time Information Assurance Testbed. This tool was designed to simulate multiple network hosts performing a variety of functions (Haines et al.).

Outside of the participants of the DARPA evaluations, Ye et al., Liao and Vemuri, and Zanero and Savaresi conducted experiments using DARPA test datasets. Wagner and Dean and Wagner and Soto used different test environments. Wagner and Soto constructed a test environment that included "a fresh Linux RedHat 5.0 installation with a version 2.2.19 kernel" (p. 261). They simulated normal activity for wuftpd, an ftp server application, by conducting "hundreds of large file downloads over a period of two days" (Wagner & Soto, p. 261).

Other researchers drew test data from live systems. This type of input data offers the benefit of real world usage patterns, but it may contain unanticipated or unidentified events. Kruegal and Vigna (2003) used live web server logs from Google and two universities in their experiments. Maxion and Tan (2000) tested an experimental system on both simulated usage data and on BSM audit logs from a live system.

Performance measurement

All quantitative performance measures focus on a system's detection of attacks and the generation of false positives. According to Barbará and Jajodia's *Applications of Data Mining in Computer Security* (2002), the detection rate and false positive rate for evaluating test systems are defined thus:

> The detection rate is defined as the number of intrusion instances detected by the system divided by the total number of intrusion instances present in the test set. The false positive rate is defined as the total number of normal instances that were (incorrectly) classified as intrusions divided by the total number of normal instances. (p. 93)

To report these performance rates, researchers typically uses either generic percentages or generate ROC (Receiver Operating Characteristic) curves. ROC curves are similar to precision-recall graphs in information retrieval literature. The graphs show, for various internal parameter values in an experimental intrusion detection system, the false positive rate increase as the detection rate arrives at 100%. The percentage of false positives is normally plotted on the x-axis with the percentage of intrusions detected on the y-axis.

Of the studies referenced in this paper that provided system performance evaluations, Kruegal and Vigna, Liao and Vemuri, and Maxion and Tan, presented results using ROC curves. Sequeira and Zaki, Ye et al., and Zanero and Savaresi reported findings in terms of percentage rates. Wagner and Soto employed a qualitative evaluation method.

Applying Term Weights to Intrusion Detection

Information Retrieval meets Intrusion Detection

Intrusion detection aims to identify attempts, either successful or unsuccessful, to obtain unauthorized access to a system. To be effective, the Intrusion Detection System (IDS) needs to provide a high rate of successful identification with a low rate of false positives. If the IDS fails to identify intrusion attempts, it does not complete its task. Additionally, if the IDS identifies too many normal events as malicious, system administrators and security analysts waste time investigating innocuous leads.

Information (or document) retrieval systems are quite similar to intrusion detection systems. A typical information retrieval system takes a user query and matches that with documents, objects, etc. within the system. The basic goal is for the system to produce the item(s) that the user is trying to find. Consider the example of a Web search engine such as Google. When performing a search, the user types a query into a text box and submits it to the system. The query is generally a few words describing web pages that the user seeks. In response to the query, the search engine returns a ranked list of results. Information retrieval systems often use statistical or probabilistic methods to determine the likelihood that a web page matches up with what the user is looking for.

The field of information retrieval employs the concept of relevance (the germaneness of items to queries) and measures the success or performance of a system in terms of recall and precision. Relevance is used to describe a match between what is available and what a user is trying to find. In terms of a Web search engine, this might be finding Web sites on intrusion detection systems when typing “intrusion detection system” into Google. For an IDS, a highly relevant match would be the identification of an intrusion attempt, since that is what the system is looking for. Nonrelevant would describe Google returning www.4greyhounds.org for the abovementioned query or an IDS flagging legitimate user activity on a system.

Performance of a retrieval system includes measures of recall and precision. Recall refers to the percentage of the total number of relevant documents available to the system that is returned for a given query:

$$\text{Recall} = \frac{\text{number of relevant items retrieved}}{\text{total number of relevant items}}$$

(Harman, 1997, p. 251). A high recall value means that the system correctly identifies all of the desired items. Precision is the percentage of how many relevant documents are retrieved based on a given query:

$$\text{Precision} = \frac{\text{number of relevant items retrieved}}{\text{total number of items retrieved}}$$

(Harman, p. 252). A high precision measure means that few nonrelevant documents are identified, or a low rate of false positives is attained. In terms of recall and precision, a good IDS should have a high value in both. That would indicate that the IDS identifies almost all unauthorized activity and produces a low number of false positives.[2]

Most information retrieval systems use a ranking method for generating results. Google, for example, ranks web pages by how closely it thinks the document fits (or is relevant to) a user's query. The relevance of a document is determined by how closely the query terms match the terms in the document. One well-known ranking method is based on a TF-IDF (term frequency - inverse document frequency) weight (Spark Jones, 1997). Each word or term in an information retrieval system's knowledgebase is given a weight based on how many documents contain that term. When generating results, the retrieval system uses these weights to rank the items presented to the user.

This TF-IDF calculation can also be applied to log analysis. Each log message is treated as a separate entity, much like Google treats web pages. In information retrieval terminology, each log message is a document, and the elements of the message are document terms. Instead of matching a search query with a list of web pages, the log analysis tool would index a set of log messages and use the TF-IDF weighting calculation to determine whether a new log message deviates from the norm.

The "IDF" part of TF-IDF is inverse document frequency. The weight value of a term is inversely related to the number of documents that contain the term. Less frequently occurring terms are given a higher weight. This means that commonly occurring log messages have a lower weight than do those with infrequently occurring elements. For log analysis, the result is that log messages with only frequently occurring

[2] Spam filters, which have begun using IR techniques in recent years, can also be evaluated in terms of precision and recall. A successful spam filter identifies almost all spam email messages (high recall) and blocks very few legitimate messages (high precision). In fact, some spam filters use the same statistical and probabilistic methods described in this paper to identify unwanted email.

elements are given a low weight. Messages that do not occur frequently are given a high weight and can easily be flagged as anomalous. Since only a human expert can make a final determination, the weight calculations in this case represent a higher or lower probability that the log message is anomalous.

An advantage of the TF-IDF method of log analysis is that the weight calculations can be done very quickly. A list of terms and corresponding IDF weights can be stored as a hash, which generally yields fast results for searching. The TF, or term frequency, component merely involves counting the number of times each element occurs in an individual log message. Since log messages are typically quite terse, counting elements should take few system resources beyond parsing the message itself.

Experimental System

The previous section described how information retrieval methods can be applied to log analysis and intrusion detection. The following is a description of the experimental system designed for this study.

The experimental system bases identification of security incidents on event log messages that deviate from the norm on a given server or workstation. A two stage process is needed for data generation. First, a training process is required for the system to establish a norm. System training involves indexing log messages for a given time period (e.g. a few days or a few weeks) and is similar to that of a search engine indexing web pages. After training, the system is ready for active classification of log messages.

For identifying anomalous events, the experimental system employs the TF-IDF weight described by Sparck Jones. This method involves keeping a count of the total number of documents or log messages indexed. The IDF weight for each element or term is calculated as log(N/n), where N is the total number of log messages and n is the number of messages that contain the given term (Sparck Jones). Croft and Harper (1997) used a similar calculation, log(N-n)/n. However, they state that for large document collections, the difference between the two methods is negligible (Croft & Harper).

A hash table similar to an inverted document index is constructed, keeping track of how many times each term occurs in a document. Afterward, the IDF weight is calculated for each term. The hash table resembles Table 1:

Table 1: Sample Term Weight Hash Table

failed	5.734
sftp	3.781
sshd	1.278
unknown_user	8.529
user1	1.241
user2	2.003

As indicated in the previous section, less frequently occurring elements receive higher weights. The TF-IDF weighting assumes that terms will reoccur, so random or semi-random strings (e.g. TCP sequence numbers in firewall log messages) require exclusion.

Once the system has established a norm, new log messages can be processed and anomalies flagged. The steps for log message evaluation are as follows:

1. Parse the log message into elements with term frequency calculated. (The frequency is the number of occurrences of a given element in the parsed log message.)
2. Look up each term in the table and obtain its IDF weight. Unknown terms are assigned an IDF value of 5% over the IDF value of the most uncommon term in the index.
3. Calculate the TF-IDF weight for each term and a total score for the log message.

The total score for the message is the sum of the TF-IDF weights of the terms. Log messages with a total score above a given threshold are considered anomalous and are flagged as indicating a security incident or system failure.

The Liao and Vemuri study mentioned in the previous section employed the k-Nearest Neighbor technique to identify anomalous processes. The kNN method bases the classification of a new document on its relationship to all known documents. According to Barbará and Jajodia (2002), the kNN "score" or weight for a given document or point in a feature space is attained "by computing the sum of the […] distances to the k-nearest neighbors of the point (p. 87). The system used in this paper uses only precalculated IDF term weights to determine the weight of a new document (log message) and to classify it as normal or anomalous.

A notable implementation difference between the Liao and Vemuri approach and the method used in this paper focuses on the presence of an unknown term. The number of unique terms in process audit logs is significantly lower than that of event logs. Liao and Vemuri listed a total of fifty unique system function calls found in their training data.

Their test system classified any process with an unknown term as anomalous (Liao and Vemuri). Due to the limited number of available system function calls, this decision was reasonable. However, a Linux syslog message collection spanning only a twenty-four hour period can have unique terms numbering in the hundreds.[3] The high number of unique terms and the frequency of new terms test data requires the use of an approximate "anomalous" term weight. A general classification of anomalous, the approach used by Liao and Vemuri, would result in an unacceptably high number of false positives when analyzing event log data.

[3] This excludes fields such as the date/time stamp that change constantly. Including this information would make the term count even higher.

Methodology

Input Data

Evaluation of the test log analysis system was based on identification of intrusions in the 1999 DARPA Intrusion Detection Evaluation simulated test data. Most other research based on the DARPA evaluation data examined the process audit logs. Since this study focused on analyzing syslog messages, only log output in the file dump data was used. The file dump data included Solaris, SunOS, and Linux syslog files, Windows NT event logs, Apache logs, network device logs, and several other minor log file types.

To define the scope of the evaluation, only Unix and Linux syslog files were used in the experiment. Data for three hosts in the DARPA simulation network was available. All of these hosts were categorized as “victim” systems, and the Simulation Network Hosts document provided further details as shown in Table 2:

Table 2: Victim Unix and Linux Simulation Network Hosts (Simulation Network Hosts - 1999)

IP Address	Hostname	Operating System
172.16.112.50	pascal.eyrie.af.mil	Solaris 2.5.1
172.16.113.50	zeno.eyrie.af.mil	SunOS 4.1.4
172.16.114.50	marx.eyrie.af.mil	RedHat 4.2 (kermel 2.0.27)

The IP addresses for the hosts were used to identify which attacks were relevant to the experiment (discussed in the following section).

In addition to the syslog log files, the DARPA file dump data included logging service configuration information. The /etc/syslog.conf configuration file for each of the three hosts contained directives for each log file. Table 3 shows which log files were used as input data for the test system. For each log file, the configuration details indicate the type of information present in the file.

Table 3: Log Files

Hostname	**Log file**	**Syslog configuration**
marx	/var/log/maillog	mail.*
	/var/log/messages	*.info;mail.none;authpriv.none
	/var/log/secure	authpriv.*
pascal	/var/adm/messages[4]	*.err;kern.debug;mail.crit;user.none;user.err
	/var/log/authlog	auth.notice
	/var/log/syslog[5]	auth.notice;daemon.info;mail.debug; kern.debug;user.err;user.alert
zeno	/var/adm/auth_msgs	auth.emerg;auth.alert;auth.crit;auth.err; auth.warning
	/var/adm/messages	*.emerg;*.alert;*.crit;*.err;kern.warning; kern.notice;kern.info;kern.debug
	/var/log/syslog	mail.debug

The log files contained informational, warning, and error messages for kernel subsystems, email applications, authentication (login), and other applications running on the test hosts.

Training and Testing

As stated above, the input data for this study consisted of Unix and Linux syslog messages. This data was divided into training data and test data. The 1999 DARPA

[4] All /var/adm/messages variants were included (e.g. /var/adm/messages.0, /var/adm/messages.1, etc.).
[5] All /var/log/syslog variants were included (e.g. /var/log/syslog.0, /var/log/syslog.1, etc.).

Evaluation provided five weeks (Monday through Friday) of total data — three weeks of training data and two weeks of test data. Intrusions, or attempted intrusions, were generated during both weeks of test data gathering, as well as during the second week of training data. This study used all three weeks of training data and the first week of test data (1999 DARPA Intrusion Detection Evaluation Plan).

The DARPA Evaluation project supplied detailed information on the simulated attacks (intrusions and attempted intrusions). The information on each attack included source and destination, the start time and duration, the type and nature of the exploit, whether or not the attack left traces in system logs, and several other categorizations specific to the DARPA Evaluation project. Since this study focused on syslog data, only attacks that were 1) directed toward the three "victim" systems listed above and 2) were detectable from syslog data were included in the experiment. Activity directed at Windows NT hosts, network equipment, and Unix/Linux systems for which syslog data was not recorded were excluded.

According to the attack labeling information, 757 total individual attacks took place during the first week of test data generation. Of those, 88 attacks were viewable in syslog messages. Finally, 38 attacks existed for the three systems examined by this study (Detection Scoring Truth). Performance of the test log analysis system was based on how many log messages corresponding to these attacks were flagged as anomalous.

The test system used term weights to identify anomalies in the log file data. Log messages, such as those generated by firewalls and email server applications, contain fields that change frequently. Since the term weights are based on the frequency of occurrence, weights can be skewed when log files contain many unique terms. To avoid this problem, a list of stop patterns was used:

```
src[0-9]+
id[0-9]+
window[0-9]+
ttl[0-9]+
delay[0-9]+
msgid[0-9]+.*
```

Instead of stop words, which are common in information retrieval systems, regular expression patterns were employed to exclude log message terms. The DARPA log files contained frequently changing terms such as email message id numbers and timestamps.

Performance Evaluation

Results of the test system's ability to detect known attacks were provided as ROC (Receiver Operating Characteristic) graphs. The graphs show the relationship between successful attack detection and false positive generation as the message weight threshold was lowered. Lowering the threshold within the system decreased the message weight at which a log message is flagged as anomalous. A lower threshold value within the system increased the number of attacks correctly identified but also increased the number of false positives.

The ROC graphs were formatted with the false positive rate on the x-axis and the detection rate on the y-axis. The following formulas were used to determine the measurement rates:

$$\text{False positive rate} = \frac{\text{number of log messages outside of attack timeframe flagged as anomalous}}{\text{total number of log messages analyzed}}$$

and

$$\text{Detection rate} = \frac{\text{number of log messages in attack timeframe flagged as anomalous}}{\text{total number of log messages within attack periods}} .$$

In addition to the basic detection rate vs. false positive rate comparison, several other factors were varied as part of the study. Three different training time periods were used. Results with ROC graphs were generated from test system indexes of one day, one week, and the complete three weeks of training data. Further, the raw message weight was compared to a weight normalized by log message term length. The normalized weight was calculated by dividing the raw message weight by the number of terms in the given log message.

Results

This chapter presents results from the experimental system in two areas: training and testing. The training section discusses characteristics of the log file indexing for the three hosts. The testing section covers the effectiveness of the test system's method for identifying anomalous events.

Training

The 1999 DARPA Intrusion Detection Evaluation data sets contained three weeks of training data. A total of fifteen days of log files were present for each of the three simulation "victim" hosts. The following graphs demonstrate the progression of documents (individual log messages) indexed, the number of unique terms, and the maximum term weight for each host's index during system training. The date values along the x-axis are a combination of week and day. They correspond to the five days in each of the three weeks of training data.

Figure 1: Document Count Progression

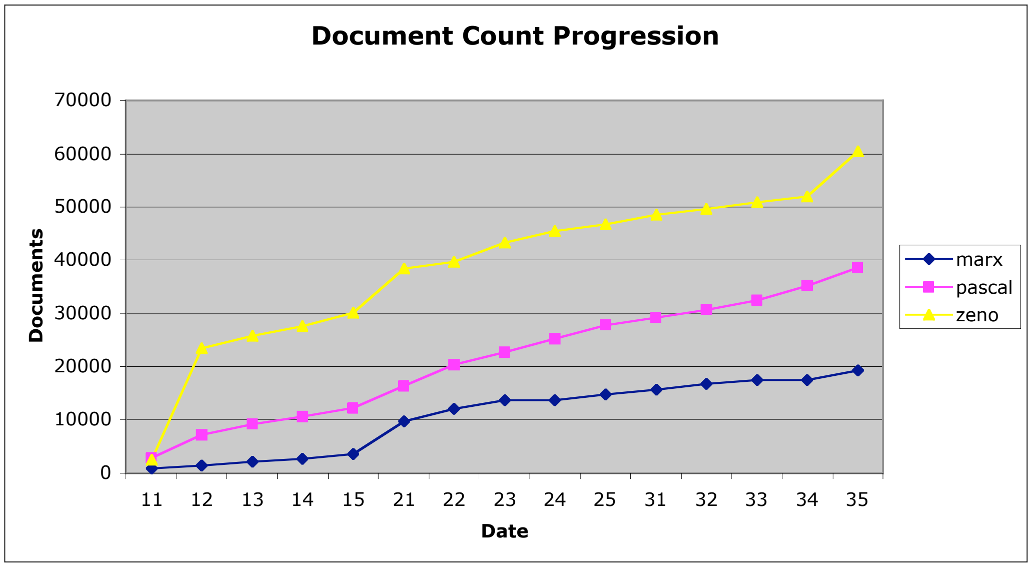
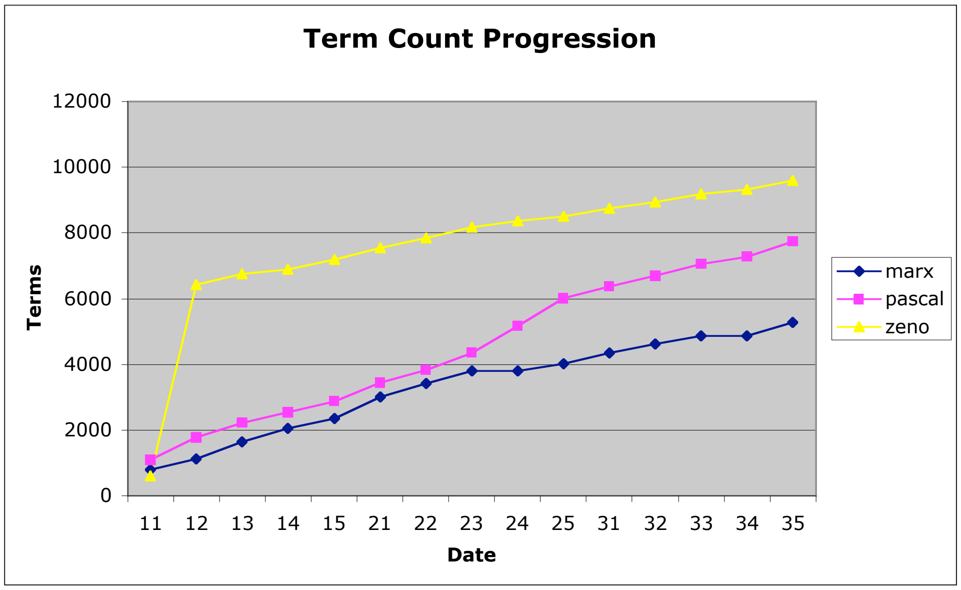


Figure 1 shows a steady increase of total documents in the term indexes during the training process. The number of documents indexed was recorded at each day during the three weeks of training data. The graph indicates linear growth for each of the three hosts, which means that roughly the same amount of log file data was generated each day by the simulated hosts. The sharp increases in the document count for the host zeno appear to be anomalies. Some types of intrusions cause large numbers of log messages to be generated. However, simulated attacks were present only during week two of the training period, not weeks one and three.

Figure 2: Term Count Progression



As in Figure 1, one sees in Figure 2 a regular increase in the index term count during the training period. The change in term count for the host zeno in the first two days corresponds to the event shown in Figure 1. The climbing number of unique terms in the index indicates potential benefits for a larger amount of training material. A higher percentage of indexed (already known) terms should yield better accuracy in identifying anomalies. In fact, results of this study show slightly better accuracy and will be discussed in the following section. However, the increase in unique terms shown in Figure 1 is also partially due to inadequate stop patterns. Some log message types continued to generate a large number of unique terms.

Figure 3: Term Weight Progression

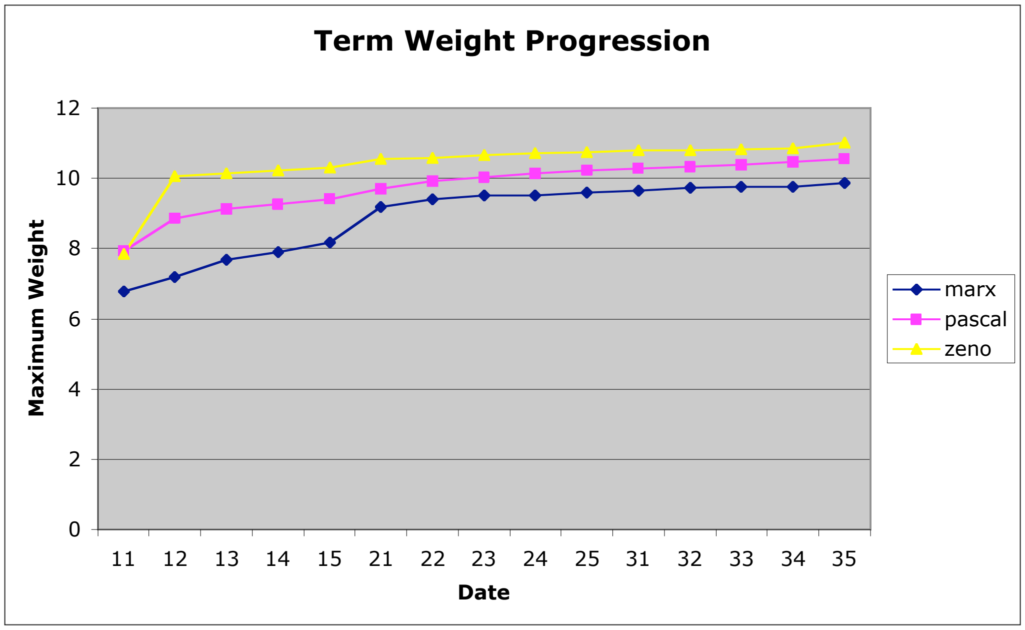


Figure 3 shows the increase in maximum weight during the training period. The small change in maximum weight, especially after the first week, indicates that a reasonable threshold could be chosen sooner than at the end of the third week of training data. The maximum weight in the index affects the weighting of terms in test data that are not present in training data. Unknown terms are deemed somewhat more anomalous than known terms and are assigned a term weight by the system. If the system were further trained after choosing a threshold, and the maximum weight increased significantly, the system would produce a high number of false positives. An increase in the threshold would be required to correct for this phenomenon. The small change in Figure 3 after the first week indicates that additional data could be indexed successfully without having to adjust the weight threshold. This would be useful if the host being analyzed had not yet generated much log file data.

Testing

The testing results section describes the effectiveness of the experimental system in detecting intrusions using term weight based log analysis. In addition to the term weight and ROC graphs, several examples are given to demonstrate both correct detection and false positive generation. After each log message, the host, trained index type (1 day, 1 week, or 3 weeks), the message weight type (normalized or nonnormalized), and the message weight are listed in parentheses.

The two following log messages were correctly flagged by the test system. They correspond to a process table attack, a type of denial service attack, on the victim host. This attack was detected with both raw (nonnormalized) and normalized message weights.

```
Mar 31 10:31:25 zeno vmunix: proc: table is full
```
(zeno, 3 weeks, normalized, 7.60843468838946)

```
Mar 31 10:39:58 zeno sendmail[102]: NOQUEUE: SYSERR: daemon:
cannot fork: No more processes
```
(zeno, 3 weeks, nonnormalized, 54.7247326476436)

The following list of log messages contains examples of false positives. These messages were not generated during an attack. However, such messages could nonetheless prove interesting to a system administrator for other reasons.

• Log messages that were generated during a system boot sequence (While the clock error message does not represent an attack, alerting a sysadmin to this error may aid in correcting a non-security problem.)

```
Apr  2 11:20:18 zeno vmunix: WARNING: clock gained 2 days --
CHECK AND RESET THE DATE!
```
(zeno, 3 weeks, nonnormalized, 81.2944170548963)

```
Mar 31 07:41:51 marx kernel: Linux version 2.0.30
(root@porky.redhat.com) (gcc version 2.7.2.1) #1 Tue Apr 22
10:49:45 EDT 1997
```
(marx, 3 weeks, nonnormalized, 102.134543685226)

• A “host not found” error for the sendmail email application, which does not indicate a security threat

```
Apr  1 11:45:43 marx sendmail[1728]: LAA01726:
to=lucyj@192.168.1.10, ctladdr=bramy (2051/100), delay=00:00:00,
xdelay=00:00:00, mailer=esmtp, relay=192.168.1.10, stat=Host
unknown (Name server: 192.168.1.10: host not found)
```
(marx, 3 weeks, nonnormalized, 109.953043395944)

• A kernel-level network device error message (As above, while this does not indicate a security threat, it may represent a different type of system problem that would require the attention of a sysadmin.)

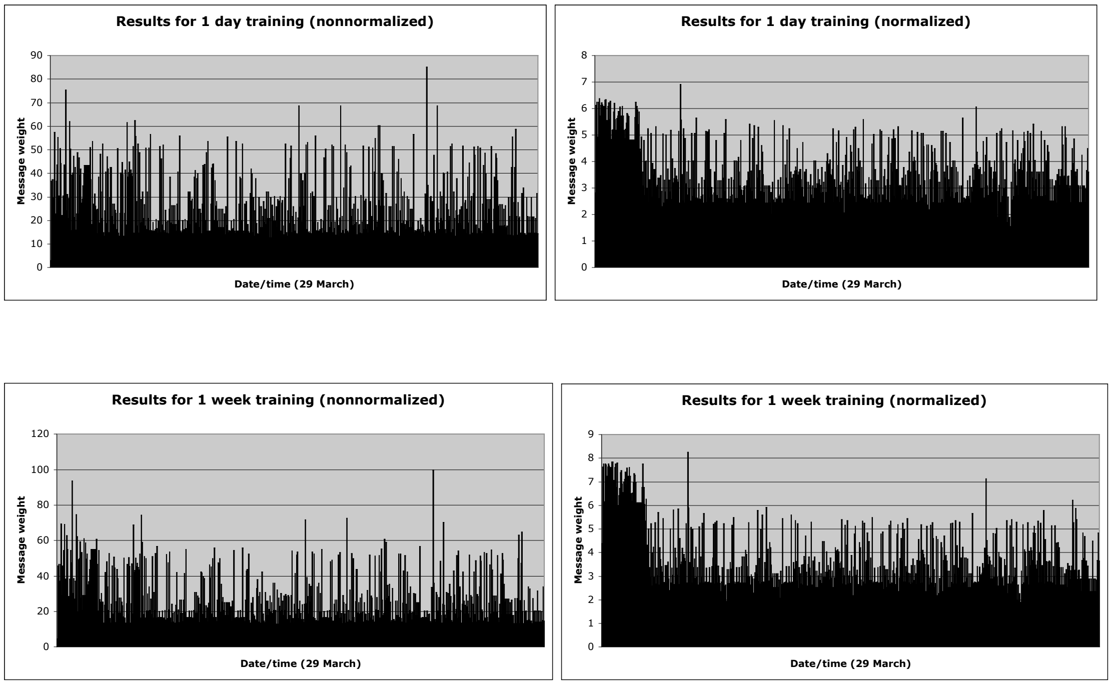
```
Apr  1 06:53:12 marx kernel: eth0: bogus packet: status=0x0
nxpg=0x3a size=1082
```
(marx, 3 weeks, normalized, 8.74120553458676)

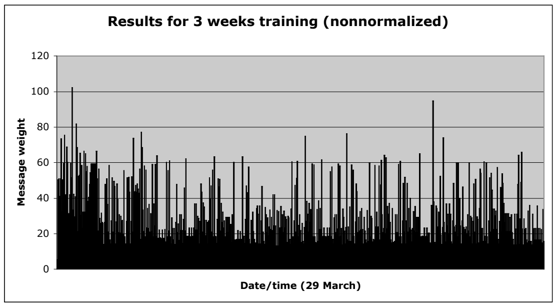
• An informational ntpd (a system clock synchronization application) message, which is a clear false positive

```
Apr  2 11:24:43 zeno xntpd[151]: synchronized to 172.16.112.10,
stratum=5
```
(zeno, full, normalized, 9.71138635120919)

To further describe the results of this study, a series of message weight graphs (Figure 4) is provided. These graphs depict individual message weights for the first day of test log file data for the host marx. One notes slight differences in message weight distribution as the training period varies. The difference between one day and one week is greater than between one week and three weeks. This supports the conclusion suggested in the training results section that an accurate message weight threshold could be chosen after the first week of training data.

Figure 4: Message Weight Graph Series





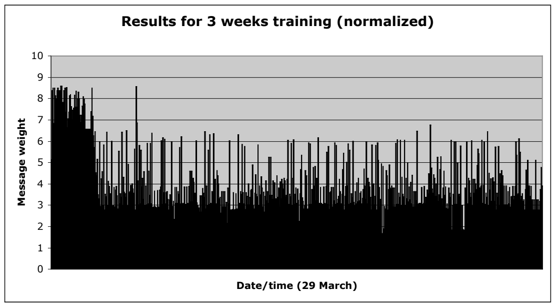
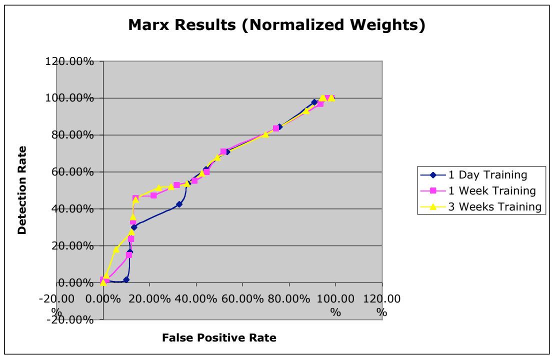
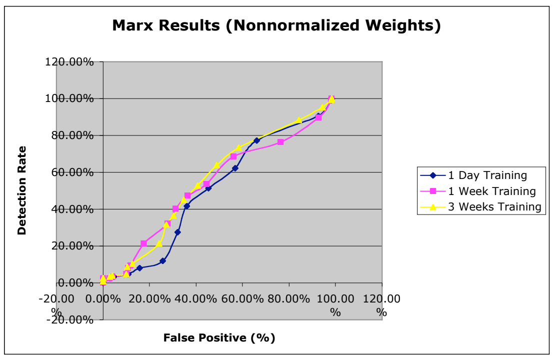
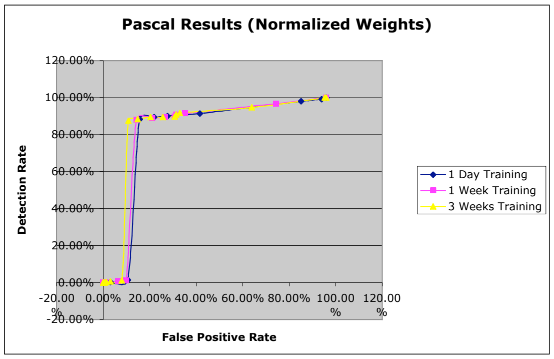


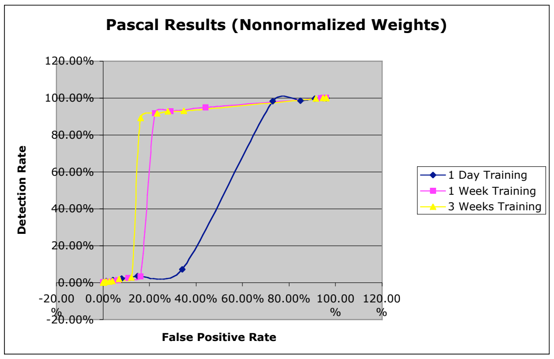
Figure 5 contains ROC graphs for the three simulation hosts and compares normalized and nonnormalized message weight methods. These graphs show the change in detection rate and false positive rate as the anomaly threshold was decreased. A lower threshold resulted in higher rates of both detection and false positive generation.

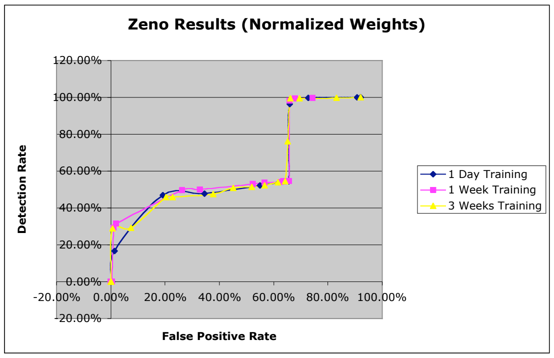
Figure 5: ROC Graph Series

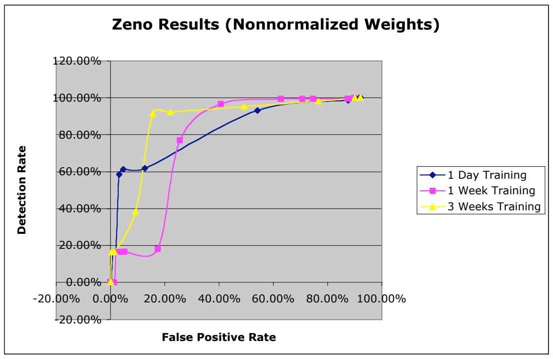












The test system yielded moderately successful results overall in correctly identifying intrusions in the test data. Like most intrusion detection systems, it demonstrated an increase in false positives as the percentage of attack detection increased. As seen in the graphs, the system performed best for the host pascal.

Despite the correct identifications, the high rate of false positive generation would make the test system almost unusable in practice. Each host generated at minimum several hundred log messages per day. Even at a conservative false positive rate of 20%, the system could still flag over 100 messages per day as possible intrusions. A further drawback is the narrowing in scope of this study. To evaluate the log analysis functionality of the test system, a significant number of attacks were excluded from the detection rate calculation. If these attacks had been included, the detection rate vs. false positive rate performance would have been worse than that shown in the ROC graphs.

In addition to overall performance results, one notes the effects of changes in weight normalization and training data. The ROC graphs show slightly better performance of normalized message weights and for using three weeks of training data. Normalized message weights act as an equalizing factor between log messages of different lengths. Normalizing the weights prevents messages with a greater than average number of terms from being flagged as anomalous merely because of length. This focuses anomaly detection on terms rather than message length.

Tests using three weeks of training data were more successful than those using less training data. This finding is consistent with statistical baselining. As the amount of data available for establishing a baseline increases, so does the accuracy with which the system identifies events that deviate from the norm.

Conclusion

A term weight based log analysis is not, by itself, an effective intrusion detection system. The large rate of false positives and the number of intrusions not detectable from syslog data make using log files as a sole source of data unreliable. However, the test system successfully detected anomalous log messages. These messages corresponded not only to attacks in the test data, but they also reflected other types of computer host problems.

Applying term weights to log file data would be very useful as a generalized log analysis tool for system administrators or as one component in a larger intrusion detection system. As a generalized log analysis tool, the system would report not only potential security problems but also direct system administrators to investigate issues such as hardware and software errors. In addition, the system may also be effective as part of an intrusion detection system. Log files contain valuable information. Combined with other techniques and data sources, such as those described in the relevant literature section of this paper, a successful intrusion detection system could be developed.

Avenues for Further Research

Given the results of this study, two avenues for further exploration emerge. The test system in this study analyzed log file data separately for each host. In addition to dividing the log analysis by host, further separate the statistical baseline generation by application. For example, better performance may be achieved by analyzing mail server application log messages differently from kernel subsystem messages or login messages. Improved stop patterns would also affect test results. The patterns used in this study were developed to reduce the number of unique items in the term index by ignoring constantly changing fields. However, more effort could be focused on developing better patterns.

Appendix A: `statlog.pl` Source Code

```
#!/usr/bin/perl -w

#-------------------------------------------------------------#
#
# statlog.pl
#
# statistical log analyzer
#
# uses term weights to identify anomalous log messages
#
#-------------------------------------------------------------#

use strict;
use Getopt::Long;
Getopt::Long::Configure('bundling', 'no_ignore_case');
use FindBin;
use lib $FindBin::Bin;
use indexer;
use analyzer;

# set base directory
my $basedir = $FindBin::Bin;

sub usage ($);
sub print_usage ();
sub help ();

#-------------------------------------------------------------#
my(
$opt_l,$logfile,
$opt_db,$db,
$opt_analyze,$analyze,
$opt_normalize,$normalize,
$opt_train,$train,
$opt_stats,$stats,
$opt_dump,$dump,
$opt_lw,$lineweights,
$opt_t,$test,
$opt_debug,$debug,
$opt_q,$quiet
);

# read conf file arg
GetOptions (
       "h|help"            => \&help,
       "quiet"                    => \$opt_q,
       "debug"                    => \$opt_debug,
       "t|test"            => \$opt_t,
       "train"                    => \$opt_train,
       "analyze"           => \$opt_analyze,
       "normalize"         => \$opt_normalize,
       "stats"                    => \$opt_stats,
       "dump"              => \$opt_dump,
       "lineweights"       => \$opt_lw,
       "db=s"              => \$opt_db,
       "l|logfile=s"       => \$opt_l
);

# analyze option
($opt_analyze) ? ($analyze = 1) : ($analyze = 0);
```

```
# train option
($opt_train) ? ($train = 1) : ($train = 0);

# normalize option
($opt_normalize) ? ($normalize = 1) : ($normalize = 0);

# dump option
($opt_dump) ? ($dump = 1) : ($dump = 0);

# lineweights option
($opt_lw) ? ($lineweights = 1) : ($lineweights = 0);

# stats option
($opt_stats) ? ($stats = 1) : ($stats = 0);

# mode check
if (!$analyze && !$train && !$dump && !$stats) {
       &usage("Mode specification required");
}

# logfile option
if ($opt_l) {
       $logfile = $opt_l;
} elsif (!$dump && !$stats) {
       &usage("Logfile required");
}

# dbfile option
($opt_db) ? ($db = $opt_db) : ($db = "");

# test option
($opt_t) ? ($test = 1) : ($test = 0);

# debug option
($opt_debug) ? ($debug = 1) : ($debug = 0);

# quiet option
($opt_q) ? ($quiet = 1) : ($quiet = 0);

#------------------------------------------------------------#

if ($debug) {
       print "logfile: $logfile\n";
       print "db filename: $db\n";
       print "analyze: $analyze\n";
       print "train: $train\n";
       print "dump: $dump\n";
       print "lineweights: $lineweights\n";
       print "quiet: $quiet\n";
       print "test: $test\n";
}

my ($idxizer,$anlyzer);

if ($logfile && !$quiet) {
       print "Logfile: $logfile\n";
}

if ($train) {
       unless ($quiet) {
              print "Training mode\n";
       }
```

```
        $idxizer = new indexer;
        if ($debug) {
              $idxizer->Configure('DEBUG');
        }
        if ($quiet) {
              $idxizer->Configure('QUIET');
        }
        if ($test) {
              $idxizer->Configure('TEST');
        }
        if ($db) {
              $idxizer->Configure("DB_FILENAME = $db");
        }
        if ($basedir) {
              $idxizer->Configure("BASEDIR = $basedir");
        }

        $idxizer->build_index($logfile);
        unless ($quiet) {
              print "Reporting term index stats\n";
        }
        $idxizer->stats();

} elsif ($analyze || $dump || $stats) {

        $anlyzer = new analyzer;
        if ($debug) {
              $anlyzer->Configure('DEBUG');
        }
        if ($quiet) {
              $anlyzer->Configure('QUIET');
        }
        if ($test) {
              $anlyzer->Configure('TEST');
        }
        if ($db) {
              $anlyzer->Configure("DB_FILENAME = $db");
        }
        if ($basedir) {
              $anlyzer->Configure("BASEDIR = $basedir");
        }
        if ($normalize) {
              $anlyzer->Configure('NORMALIZE');
        }

        if ($analyze) {
              unless ($quiet) {
                    print "Analyze mode\n";
              }
              if ($lineweights) {
                    $anlyzer->Configure('LINEWEIGHTS');
              }
              $anlyzer->analyze($logfile);

        } elsif ($dump) {
              unless ($quiet) {
                    print "Dumping term index\n";
              }
              $anlyzer->dump_index();

        } else {
              unless ($quiet) {
```

```
                        print "Reporting term index stats\n";
                }
                $anlyzer->stats();
        }

} else {
        print "Oops, nothing to do.\n";
}

# Done.
exit;


#-------------------------------------------------------------#
# print usage error message, instructions, then exit
sub usage ($) {
        my $msg = shift();
        print "$msg\n";
        &print_usage();
        exit 1;
}

#-------------------------------------------------------------#
# usage instructions
sub print_usage () {
        print "Usage: logtool.pl\n";
        print "            -l, --logfile          : input log file\n";
        print "            --db                   : term database filename\n";
        print "            --analyze              : invoke analyze mode\n";
        print "            --train                : invoke training mode\n";
        print "            --normalize            : normalize log message weights
by term count\n";
        print "            --dump                 : dump contents of the term
index\n";
        print "            --lineweights          : in analyze mode, display all
log lines \n";
        print "                                       with weights (tab-
separated)\n";
        print "            --test                 : don't really do anything\n";
        print "            --debug                : lots of extra output\n";
        print "            --quiet                : supress messages\n";
        print "            --help                 : I need a hug\n";
}

#-------------------------------------------------------------#
# help function
sub help () {
        &print_usage();
        exit;
}
```

Appendix B: indexer.pm Source Code

```
#-------------------------------------------------------------#
#
# indexer.pm
#
# logtool module for indexing log messages.  adds log files to the index
#      of terms and regenerates the term weights
#
#
#-------------------------------------------------------------#

package indexer;
$VERSION='1.0';

require 5.004;
require Exporter;
use strict;
use tokenizer;

#-------------------------------------------------------------#
# default config options

my %CONFIG;
$CONFIG{"MIN_TOKEN_LENGTH"} = 3;
$CONFIG{"DB_FILENAME"} = "term_weights.db";
$CONFIG{"DBFILE_MODE"} = "6610";
$CONFIG{"QUIET"} = 0;
$CONFIG{"DEBUG"} = 0;
$CONFIG{"BASEDIR"} = ".";
$CONFIG{"TEST"} = 0;

#-------------------------------------------------------------#
# function prototypes

sub build_index;
sub add_logfile;
sub terms2idx;
sub gen_weights;
sub save_weights;
sub Configure;


#-------------------------------------------------------------#

# xxxdoc_count term used to keep track of the number of documents
#      processed for the index.  this allows recalculation of term
#      weights when new log messages are added to the index.

# xxxmax_weight term is the maximum term weight value.  this is used
#      to calculate a weight for terms not found in the index

# term_hash+
#          |
#          +-term-+-"<weight>:<doc_count>"
#          |
#          +-term-+-"<weight>:<doc_count>"
#          |
#          +-term-+-"<weight>:<doc_count>"
#

#-----------------------------------------------------------------------#
```

```
my (%term_hash,%term_weight);
my (%stats_hash);
my $doc_count = 0;
my $max_weight = 0;


#-------------------------------------------------------------#
# constructor
sub new {
	my $self = {};
	bless $self;
	return $self;
}

#-----------------------------------------------------------------------#
# index building functions
#-----------------------------------------------------------------------#

#-----------------------------------------------------------------------#
sub build_index {
# input: input filename
# output: error code

	my $self = shift;
	my ($infile) = @_;
	my $err = 0;

	if ($CONFIG{"DEBUG"}) {
		print "build_index (logfile: $infile)\n";
	}

	# open the db file
	dbmopen(%term_hash, $CONFIG{"DB_FILENAME"}, $CONFIG{"DBFILE_MODE"}) or
			die ("Error opening dbm file " .
				$CONFIG{"DB_FILENAME"} . "\n$!.\n");

	# check for existing entries in the term_hash.  if we find a doc count,
	#	we're updating the index (not creating from scratch)
	if (exists($term_hash{"xxxdoc_count"})) {
		$doc_count = $term_hash{"xxxdoc_count"};
	}

	#-------------------------------------------------------------#
	# construct inverted document index of terms
	unless ($CONFIG{"QUIET"}) {
		print "Constructing inverted document index\n";
	}

	$self->add_logfile($infile,1);

	#-------------------------------------------------------------#
	# calculate term weights
	unless ($CONFIG{"QUIET"}) {
		print "Calculating term weights\n";
	}

	$self->gen_weights();

	#-------------------------------------------------------------#
	# save the term weights to the dbm file
	unless ($CONFIG{"QUIET"}) {
		print "Saving term weights to index dbm file\n";
```

```
        }

        $self->save_weights();

        # save (or update) the doc count
        $term_hash{"xxxdoc_count"} = $doc_count;
        # update stats hash
        $stats_hash{"doc_count"} = $doc_count;

        # save the max term weight value
        $term_hash{"xxxmax_weight"} = $max_weight;
        # update stats hash
        $stats_hash{"max_weight"} = $max_weight;

        # close the db file
        dbmclose(%term_hash);

        return($err);
}

#----------------------------------------------------------------------#
sub add_logfile {
# input: log file, multiplier
# output: error code

        my $self = shift;
        my ($infile,$multiplier) = @_;

        my $tokizer = new tokenizer;
        $tokizer->Configure("MIN_TOKEN_LENGTH = $CONFIG{'MIN_TOKEN_LENGTH'}");
        $tokizer->Configure("BASEDIR = $CONFIG{'BASEDIR'}");
        $tokizer->Configure("UNIQUE");
        $tokizer->init_stoppatterns;

        if ($CONFIG{"DEBUG"}) {
                print "add_logfile (logfile: $infile, multiplier: $multiplier)\n";
                $tokizer->Configure("DEBUG");
        }

        if ($CONFIG{"TEST"}) {
                return(0);
        }

        open(LOG,"<$infile") or die "Unable to open input file $infile\n";
        my ($line,@tmparr);

        # process the log file
        while (defined($line=<LOG>)) {

                @tmparr = $tokizer->tokenize_line($line);
                $self->terms2idx(\@tmparr,$multiplier);

                # increment the total document count
                ++$doc_count;
        }
        close(LOG);

        return(0);
}

#----------------------------------------------------------------------#
# takes an array of terms and adds them to the inverted index
# the multiplier value allows weighting some terms more than others
```

```
sub terms2idx {
# input: array ref, multiplier
# output: error code

       my $self = shift;
       my ($tokens_ref,$multiplier) = @_;

       # check multiplier
       if (!$multiplier) {
              print "terms2idx mulitiplier missing\n";
              return 1;
       }

       my (@tokens,$token,$weight,$count);

       if ($CONFIG{"DEBUG"}) {
              print "terms2idx (multiplier: $multiplier)\n";
       }

       # add the terms to the index
       foreach $token (@{$tokens_ref}) {

              # found a new term
              if (!exists($term_hash{$token})) {
                     # create term entry, set weight to 0 and count to 1
                     $term_hash{$token} = "0:1";

              # term entry already exists, increment the doc count
              } else {
                     # increment (with multiplier) count for term
                     ($weight,$count) = split(/:/,$term_hash{$token});
                     $count = $count + (1 * $multiplier);
                     $term_hash{$token} = "$weight:$count";
              }
       }

       return(0);
}


#------------------------------------------------------------------------#
sub gen_weights() {
# input: none
# output: error code

       # %term_doc
       # %term_weight

       if ($CONFIG{"TEST"}) {
              return(0);
       }

       if ($CONFIG{"DEBUG"}) {
              print "doc_count: $doc_count\n";
       }

       #-------------------------------------------------------------#
       # get doc count for each term & compute weight

       my ($term,$count,$weight,$oldweight);

       foreach $term (keys(%term_hash)) {
              # skip the xxxdoc_count entry
```

```
		if ($term eq "xxxdoc_count") { next; }

		# skip the xxxmax_weight entry
		if ($term eq "xxxmax_weight") { next; }

		# get the current information from the term_hash
		($oldweight,$count) = split(/:/,$term_hash{$term});

		# calculate IDF weight (Croft): log(N-n/n)
		if ($count == $doc_count) {
			$weight = 0;
		} else {
			$weight = log(($doc_count - $count)/$count);
		}

		# keep track of the max term weight value
		($max_weight < $weight) ? ($max_weight = $weight) : ();

		# save term weight and document count
		$term_weight{$term} = "$weight:$count";

		#print "$term: " . $term_weight{$term} . "\n";
	}

	if ($CONFIG{"DEBUG"}) {
		print "max_weight: $max_weight\n";
	}

	return(0);
}

#----------------------------------------------------------------------#
sub save_weights {
# input: none
# output: error code

	my $self = shift;

	my $term;
	my $count = 0;

	if ($CONFIG{"DEBUG"}) {
		print "save_weights ()\n";
	}

	if ($CONFIG{"DEBUG"}) {
		print "Saving " . (scalar(keys(%term_weight)) - 2) . " terms\n";
	}

	foreach $term (keys(%term_weight)) {
		#print "term_weight{$term}: " . $term_weight{$term} . "\n";
		$term_hash{$term} = $term_weight{$term};

		if ($CONFIG{"DEBUG"} && ($count%500 == 0)) {
			print "Saved term $count of $doc_count\n";
		}
		++$count;
	}

	# update stats hash with term count
	# don't count the xxxdoc_count or xxxmax_weight terms
	$stats_hash{"term_count"} = scalar(keys(%term_hash)) - 2;
	if ($CONFIG{"DEBUG"}) {
		print $stats_hash{"term_count"} . " terms in the index\n";
	}

	return(0);
}

#----------------------------------------------------------------------#
sub stats {
# input: none
# output: error code

	my $self = shift;

	# display stats
	print "doc_count: " . $stats_hash{"doc_count"} . "\n";
	print "max_weight: " . $stats_hash{"max_weight"} . "\n";
	print "term_count: " . $stats_hash{"term_count"} . "\n";

	return(0);
}


#----------------------------------------------------------------------#
# helper functions
#----------------------------------------------------------------------#

#----------------------------------------------------------------------#
sub Configure {
	my $self = shift;
	my (@options) = @_;

	my ($option,$value);
	foreach $option (@options) {
		$option = lc($option);

		if ($option eq "debug") {
			$CONFIG{"DEBUG"} = 1;
		} elsif ($option eq "quiet") {
			$CONFIG{"QUIET"} = 1;
		} elsif ($option eq "test") {
			$CONFIG{"TEST"} = 1;
		} elsif ($option =~ /^min_token_length/) {
			($option,$value) = split(/\s*=\s*/,$option);
			$CONFIG{"MIN_TOKEN_LENGTH"} = $value;
		} elsif ($option =~ /^db_filename/) {
			($option,$value) = split(/\s*=\s*/,$option);
			$CONFIG{"DB_FILENAME"} = $value;
		} elsif ($option =~ /^basedir/) {
			($option,$value) = split(/\s*=\s*/,$option);
			$CONFIG{"BASEDIR"} = $value;
		}
	}
}

1;
```

Appendix C: analyzer.pm Source Code

```
#------------------------------------------------------------#
#
# analyzer.pm
#
# logtool module for identifying anomalous log messages
#      looks up terms in the index and calculates TF-IDF weight
#      for the given message
#
#
#------------------------------------------------------------#

package analyzer;
$VERSION='1.0';

require 5.004;
require Exporter;
use strict;
use tokenizer;

#------------------------------------------------------------#
# default config options

my %CONFIG;
$CONFIG{"MIN_TOKEN_LENGTH"} = 3;
$CONFIG{"DB_FILENAME"} = "term_weights.db";
$CONFIG{"DBFILE_MODE"} = "6610";
$CONFIG{"QUIET"} = 0;
$CONFIG{"DEBUG"} = 0;
$CONFIG{"TEST"} = 0;
$CONFIG{"NORMALIZE"} = 0;
$CONFIG{"LINEWEIGHTS"} = 0;
$CONFIG{"BASEDIR"} = ".";
$CONFIG{"P"} = 0.85;
$CONFIG{"Q"} = 0.6;
$CONFIG{"THRESHOLD"} = 5.5;
$CONFIG{"UNKNOWN_TERM_WEIGHT_ADD"} = 1;

#------------------------------------------------------------#
# function prototypes

sub analyze;
sub process_logfile;
sub calc_weight;
sub stats;
sub dump_index;
sub index_term_report;
sub Configure;

#------------------------------------------------------------#

# xxxdoc_count term used to keep track of the number of documents
#      processed for the index.  this allows recalculation of term
#      weights when new log messages are added to the index.

# xxxmax_weight term is the maximum term weight value.  this is used
#      to calculate a weight for terms not found in the index

# term_hash+
#         |
#         +-term-+-"<weight>:<doc_count>"
#         |
#         +-term-+-"<weight>:<doc_count>"
#         |
#         +-term-+-"<weight>:<doc_count>"
#

#------------------------------------------------------------------------#

my %term_hash;
my $max_weight = 0;


#-------------------------------------------------------------#
# constructor
sub new {
       my $self = {};
       bless $self;
       return $self;
}

#------------------------------------------------------------------------#
# index building functions
#------------------------------------------------------------------------#

#------------------------------------------------------------------------#
sub analyze {
# input: input filename
# output: error code

       my $self = shift;
       my ($infile) = @_;
       my $err = 0;

       if ($CONFIG{"DEBUG"}) {
              print "analyze (logfile: $infile)\n";
       }

       # open the db file
       dbmopen(%term_hash, $CONFIG{"DB_FILENAME"}, $CONFIG{"DBFILE_MODE"}) or
                     die ("Error opening dbm file " .
                            $CONFIG{"DB_FILENAME"} . "\n$!.\n");

       # retrieve the maximum term weight value
       if (!exists($term_hash{"xxxmax_weight"})) {
              print "Oops, couldn't find xxxmax_weight dbm hash key.";
              print "Unable to proceed.\n";
              return 1;
       } else {
              $max_weight = $term_hash{"xxxmax_weight"};
       }

       $self->process_logfile($infile);

       # close the db file
       dbmclose(%term_hash);

       return($err);
}


#------------------------------------------------------------------------#
sub process_logfile {
# input: log file
# output: error code

	my $self = shift;
	my ($infile) = @_;

	my $tokizer = new tokenizer;
	$tokizer->Configure("MIN_TOKEN_LENGTH =
				$CONFIG{'MIN_TOKEN_LENGTH'}");
	$tokizer->Configure("BASEDIR = $CONFIG{'BASEDIR'}");
	$tokizer->init_stoppatterns;

	if ($CONFIG{"DEBUG"}) {
		print "process_logfile (logfile: $infile)\n";
		$tokizer->Configure("DEBUG");
	}

	if ($CONFIG{"TEST"}) {
		return(0);
	}

	# threshold = log(p(1-q)/q(1-p))
	#my $threshold = log(($CONFIG{"P"}*(1 - $CONFIG{"Q"})) /
			($CONFIG{"Q"}*(1 - $CONFIG{"P"})));
	#my $threshold = ($CONFIG{"P"}*(1 - $CONFIG{"Q"})) /
			($CONFIG{"Q"}*(1 - $CONFIG{"P"}));
	my $threshold = $CONFIG{'THRESHOLD'};

	unless ($CONFIG{'QUIET'}) {
		print "threshold: $threshold\n";
	}

	open(LOG,"<$infile") or die "Unable to open input file $infile\n";
	my ($line,@tmparr,$lineweight);

	# find statistically anomalous lines in the log file
	while (defined($line=<LOG>)) {
		chomp($line);

		# tokenize the line
		@tmparr = $tokizer->tokenize_line($line);

		# calculate the TF-IDF weight for the line
		$lineweight = $self->calc_weight(\@tmparr);

		# if LINEWEIGHTS is set, print: log message<tab>weight
		if ($CONFIG{'LINEWEIGHTS'}) {
			print "$line\t$lineweight\n";

		# otherwise, compare the line weight with the
		#	TF-IDF threshold value
		} elsif ($lineweight > $threshold) {
			print "anomaly: $line (weight: $lineweight)\n";
		}

	}
	close(LOG);

	return(0);
}

#---------------------------------------------------------------------#
# calculates the tf-idf weight for a log message
sub calc_weight {
```

```
# input: array ref
# output: weight value

	my $self = shift;
	my ($tokens_ref) = @_;

	if ($CONFIG{"TEST"}) {
		return(0);
	}

	if ($CONFIG{"DEBUG"}) {
		print "calc_weight (tokens_ref: arr_ref)\n";
	}

	my (@tokens,$token,$weight,$count);
	my $msg_weight = 0;

	# add the terms to the index
	foreach $token (@{$tokens_ref}) {

		# term not in index, assign a weight
		if (!exists($term_hash{$token})) {
			$weight = $max_weight +
					$CONFIG{"UNKNOWN_TERM_WEIGHT_ADD"};

			if ($CONFIG{"DEBUG"}) {
				print "Term $token not found in index, ";
				print "assigned weight: $weight\n";
			}

		# term exists, retrieve weight
		} else {
			($weight,$count) = split(/:/,$term_hash{$token});
		}

		$msg_weight += $weight;
	}

	if ($CONFIG{"DEBUG"}) {
		print "calc_weight weight: $msg_weight\n";
	}

	if ($CONFIG{"NORMALIZE"}) {
		$msg_weight = $msg_weight/($#{$tokens_ref});
	}

	return($msg_weight);
}


#----------------------------------------------------------------------#
# run the term_index_report w/o dumping the index term
# this only reports index statistics
sub stats {
	my $self = shift;
	$self->term_index_report(0);
}


#----------------------------------------------------------------------#
# run the term_index_report with dumping the index term enabled
# this displays all index terms and term weights
sub dump_index {
        my $self = shift;
        $self->term_index_report(1);
}


#----------------------------------------------------------------------#
sub term_index_report ($) {
# input: none
# output: error code

        my $self = shift;
        my $dump = shift;

        my $term;

        # open the db file
        dbmopen(%term_hash, $CONFIG{"DB_FILENAME"}, $CONFIG{"DBFILE_MODE"});

        if ($dump) {
                # dump term index contents
                foreach $term (keys(%term_hash)) {
                        # skip the xxxdoc_count entry
                        if (($term eq "xxxdoc_count") ||
                                        ($term eq "xxxmax_weight"))
                                { next; }

                        print "$term => " . $term_hash{$term} . "\n";
                }
        }

        # display doc count & max weight at the end
        print "doc_count: " . $term_hash{"xxxdoc_count"} . "\n";
        print "max_weight: " . $term_hash{"xxxmax_weight"} . "\n";

        # get the term count (don't include xxxdoc_count & xxxmax_weight)
        my $term_count = scalar(keys(%term_hash)) - 2;
        print "term_count: $term_count\n";

        # close the db file
        dbmclose(%term_hash);

        return(0);
}

#----------------------------------------------------------------------#
# helper functions
#----------------------------------------------------------------------#

#----------------------------------------------------------------------#
sub Configure {
        my $self = shift;
        my (@options) = @_;

        my ($option,$value);
        foreach $option (@options) {
                $option = lc($option);

                if ($option eq "debug") {
                        $CONFIG{"DEBUG"} = 1;
                } elsif ($option eq "quiet") {
                        $CONFIG{"QUIET"} = 1;
                } elsif ($option eq "test") {
                        $CONFIG{"TEST"} = 1;
```

```
            } elsif ($option eq "lineweights") {
                  $CONFIG{"LINEWEIGHTS"} = 1;
            } elsif ($option eq "normalize") {
                  $CONFIG{"NORMALIZE"} = 1;
            } elsif ($option =~ /^min_token_length/) {
                  ($option,$value) = split(/\s*=\s*/,$option);
                  $CONFIG{"MIN_TOKEN_LENGTH"} = $value;
            } elsif ($option =~ /^db_filename/) {
                  ($option,$value) = split(/\s*=\s*/,$option);
                  $CONFIG{"DB_FILENAME"} = $value;
            } elsif ($option =~ /^basedir/) {
                  ($option,$value) = split(/\s*=\s*/,$option);
                  $CONFIG{"BASEDIR"} = $value;
            }
      }
}

1;
```

Appendix D: tokenizer.pm **Source Code**

```
#----------------------------------------------------------#
#
# tokenizer.pm
#
# logtool module for tokenizing strings (e.g. log messages) and
#      returning an array of tokens (terms)
# Includes decision-making components such as discarding terms
#      under a given length and employing stop patterns
#
#----------------------------------------------------------#

package tokenizer;
$VERSION='1.0';

require 5.004;
require Exporter;
use strict;

#----------------------------------------------------------#
# default config options

my %CONFIG;
$CONFIG{"MIN_TOKEN_LENGTH"} = 4;  # minimum length of a valid term
$CONFIG{"UNIQUE"} = 1;                  # return only unique terms?
$CONFIG{"QUIET"} = 0;
$CONFIG{"DEBUG"} = 0;
$CONFIG{"TEST"} = 0;
$CONFIG{"BASEDIR"} = ".";
$CONFIG{"STOPPATTERN_FILE"} = "stop_patterns.conf";

my %stophash;

#----------------------------------------------------------#
# function prototypes

sub init_stoppatterns;
sub tokenize_line;
sub make_unique;
sub get_stoppatterns;
sub stoppattern_match;
sub Configure;


#----------------------------------------------------------#
# constructor
sub new {
       my $self = {};
       bless $self;
       return $self;
}

#----------------------------------------------------------#
# initialize the stop patterns list
sub init_stoppatterns {
       my $self = shift();

       # import list of stop words
       $self->get_stoppatterns;
}
```

```
#-------------------------------------------------------------#
# converts a unix syslog file line into an array of terms
#	- ignore white space
#	- convert to lower case
#	- weed out duplicate terms
#	- skip 'xxxdoc_count' & 'xxxmax_weight' if it happens to
#		exist in a log file line this shouldn't happen,
#		but it's here just in case
#	- ignore stop words
#	- remove non-word characters from terms

sub tokenize_line {
	my $self = shift();
	my ($line) = @_; chomp($line);

	if ($CONFIG{"DEBUG"}) {
		print "MIN_TOKEN_LENGTH: " . $CONFIG{"MIN_TOKEN_LENGTH"} . "\n";
	}

	my (@tokens,$token);
	my @term_array = ();

	# convert format (drop pid numbers)
	# <daemon>[<pid>]:
	$line =~ s/://g;
	$line =~ s/\[[0-9]+\]//g;

	@tokens = split(/\s+/,$line);

	# get rid of month, day, time
	splice(@tokens,0,3);

	foreach $token (@tokens) {
		# xxxdoc_count and xxxmax_weight are reserved.
		#	they shouldn't appear but check just in case
		if (($token eq "xxxdoc_count") ||
				($token eq "xxxdoc_count")) { next; }

		# remove non-word characters
		$token =~ s/\W//g;

		# skip words under $CONFIG{"MIN_TOKEN_LENGTH"} characters long
		if (length($token) < $CONFIG{"MIN_TOKEN_LENGTH"}) {
			if ($CONFIG{"DEBUG"}) {
				print "Skipping token $token, too short ";
				print length($token) . " < " .
					$CONFIG{"MIN_TOKEN_LENGTH"} . "\n";
			}
			next;
		}

		# lower case
		$token = lc($token);

		# skip stop patterns
		if ($self->stoppattern_match($token)) { next; }

		# finally, add the approved term to the term_array
		unshift(@term_array,$token);
	}

	if ($CONFIG{"UNIQUE"}) {
		# we don't want terms in the same log line to be counted twice, so
		#	let's keep track of the terms we've seen for this line
		$self->make_unique(\@term_array);
	}
	return(@term_array);
}

#--------------------------------------------------------------------#
# takes an array and removes the duplicate entries
sub make_unique {
# input: array reference
# output: error code

	my $self = shift;
	my ($arr_ref) = @_;

	my ($term,%unique_terms);

	# go through each term in the array, removing the elements
	#      as we proceed
	while ($term = shift(@{$arr_ref})) {

		# skip duplicate terms
		if (exists($unique_terms{$term})) {
			#print "Duplicate term skipped: $term\n";
			next;

		# add new ones to the hash table
		} else {
			$unique_terms{$term} = 1;
		}
	}

	# put the unique terms back into the array
	@{$arr_ref} = keys(%unique_terms);

	return(0);
}

#--------------------------------------------------------------------#
# helper functions
#--------------------------------------------------------------------#

#--------------------------------------------------------------------#
# matches the term to patterns contained in the stop_words.txt file
# returns 1 if a stop word is matched, 0 otherwise
sub stoppattern_match {
# input: term
# output: error code

	my $self = shift;
	my ($term) = @_;
	my $key;
	my $retr = 0;

	foreach $key (keys(%stophash)) {
		if ($term =~ /^$key$/) {
		$retr = 1;
		if ($CONFIG{"DEBUG"}) {
			print "Matched stop pattern $key in $term\n";
		}
			last;
		}
	}
	return($retr);
}

#--------------------------------------------------------------------------#
# retrieves the list of patterns from the stop patterns config file
sub get_stoppatterns {
my $self = shift;

	my $line;

	if ($CONFIG{"DEBUG"}) {
		print "Building stop words list\n";
	}

	my $stoppatterns = $CONFIG{"BASEDIR"} . "/" .
			$CONFIG{"STOPPATTERN_FILE"};

	open(INFILE,"<$stoppatterns") or die
			"Unable to open input file: $stoppatterns\n";
	while (defined($line = <INFILE>)) {
		chomp($line);
		$line = lc($line);
		$stophash{$line} = 1;

		if ($CONFIG{"DEBUG"}) {
			print "Set stop word: $line\n";
		}
	}
	close(INFILE);
}

#--------------------------------------------------------------------------#
sub Configure {

	my $self = shift;
	my (@options) = @_;

	my ($option,$value);
	foreach $option (@options) {
		$option = lc($option);

		if ($option eq "debug") {
			$CONFIG{"DEBUG"} = 1;
		} elsif ($option eq "quiet") {
			$CONFIG{"QUIET"} = 1;
		} elsif ($option eq "test") {
			$CONFIG{"TEST"} = 1;
		} elsif ($option =~ /^min_token_length/) {
			($option,$value) = split(/\s*=\s*/,$option);
			$CONFIG{"MIN_TOKEN_LENGTH"} = $value;
		} elsif ($option =~ /^db_filename/) {
			($option,$value) = split(/\s*=\s*/,$option);
			$CONFIG{"DB_FILENAME"} = $value;
		} elsif ($option =~ /^basedir/) {
			($option,$value) = split(/\s*=\s*/,$option);
			$CONFIG{"BASEDIR"} = $value;
		} elsif ($option eq "unique") {
			$CONFIG{"UNIQUE"} = 1;
		}
	}
}
1;
```

References

Barbará, D., & Jajodia, S. (2002). *Applications of Data Mining in Computer Security*. Boston: Kluwer.

Croft, W.B., & Harper, D.J. (1997). "Using probabilistic models of document retrieval without relevance information." In K. Sparck Jones & P. Willett (Eds.), *Readings in Information Retrieval* (pp. 339-344). San Francisco: Morgan Kaufmann.

Fulton, R., & Hoffman, H. (n.d.). SL2 source code. Retrieved November 11, 2003, from http://www.ip-solutions.net/syslog-ng/sl2.

Harman, D. (1997). "The TREC conferences." In K. Sparck Jones & P. Willett (Eds.), *Readings in Information Retrieval* (pp. 247-256). San Francisco: Morgan Kaufmann.

Innella, P., & McMillan, O. (2001). "An Introduction to Intrusion Detection Systems." Retrieved November 11, 2003, from http://www.securityfocus.com/infocus/1520.

Kumar, S., & Spafford, E.H. (1994). "A Pattern matching model for misuse intrusion detection." *17th National Computer Security Conference*, pp. 11-21.

Lincoln Laboratory, Massachusetts Institute of Technology. (n.d.). "1999 Darpa Intrusion Detection Evaluation Plan." Retrieved March 18, 2004, from http://www.ll.mit.edu/IST/ideval/docs/1999/id99-eval-ll.html.

Lincoln Laboratory, Massachusetts Institute of Technology. (n.d.). "Detection scoring truth." Retrieved March 18, 2004, from http://www.ll.mit.edu/IST/ideval/docs/1999/master-listfile-condensed.txt.

Lincoln Laboratory, Massachusetts Institute of Technology. (n.d.). "Simulation network hosts - 1999." Retrieved March 20, 2004, from http://www.ll.mit.edu/IST/ideval/docs/1999/hosts.html.

Lippmann, R.P., Fried, D.J., Graf, I., Haines, J.W., Kendall, K.R., McClung, D., Weber, D., Webster, S.E., Wyschogrod, D., Cunningham, R.K., & Zissman, M.A. (2000). "Evaluating Intrusion Detection Systems: the 1998 DARPA Off-Line Intrusion Detection Evaluation." *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition, 2000, Vol. 2*. Retrieved June 29, 2004, from http://www.ll.mit.edu/IST/ideval/pubs/2000/discex00_paper.pdf.

Muscat, A. (2003). "A Log analysis based intrusion detection system for the creation of a specification based intrusion prevention system." *Proceedings of the 2003 University of Malta Computer Science Annual Research Workshop, July, 2003*. Retrieved November 12, 2004, from http://www.cs.um.edu.mt/~csaw/Proceedings/LogAnalysisIDS.pdf.

Nemeth, E., Snyder, G., Seebass, S., and Hein, T. (2001). *Unix System Administration Handbook* (3rd ed.). Upper Saddle River, NJ: Prentice Hall PTR.

Russell, R. (n.d.). "Statistics-based Intrusion Detection System." Retrieved November 11, 2003, from http://www.internettradecraft.com/sids/sids.ppt.

Sequeira, K., & Zaki, M. (2002). "ADMIT: anomaly-based data mining for intrusions." *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July, 2002.*

Sparck Jones, K. (1997). "Search term relevance weighting given little relevance information." In K. Sparck Jones & P. Willett, (Eds.), *Readings in Information Retrieval* (pp. 329-338). San Francisco: Morgan Kaufmann.

Ugurel, S., Krovetz, R., & Giles, C.L. (2002). "What's the code?: automatic classification of source code archives." *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July, 2002.*

Wagner, D., & Dean, D. (2001). "Intrusion detection via static analysis." *Proceedings of the IEEE Symposium on Security and Privacy, May 2001*. Retrieved November 6, 2003, from http://www.cs.berkeley.edu/~daw/papers/ids-oakland01.pdf.

Wagner, D. & Soto, P. (2002). "Intrusion detection: Mimicry attacks on host-based intrusion detection systems." *Proceedings of the 9th ACM conference on Computer and Communications Security, November, 2002*. Retrieved November 6, 2003, from http://www.cs.berkeley.edu/~daw/papers/mimicry.pdf.

Ye, N., Xu, M., & Emran, S.M. (2000). "Probabilistic networks with undirected links for anomaly detection." *Proceedings of the 2000 IEEE Workshop on Information Assurance and Security, June, 2000.*

Zanero, S., Savaresi, S.M. (2004). "Unsupervised learning techniques for an intrusion detection system." *Proceedings of the 2004 ACM Symposium on Applied Computing, March, 2004*. Retrieved November 6, 2003, from http://www.elet.polimi.it/upload/zanero/papers/IDS-SAC.pdf.

http://www.loganalysis.org/

http://www.logwatch.org/