Mining Event Logs with SLCT and LogHound

Risto Vaarandi

Reprinted from *Proceedings of the 2008 IEEE/IFIP Network Operations and Management Symposium*. (ISBN: 978-1-4244-2066-7)

Mining Event Logs with SLCT and LogHound

Risto Vaarandi
Cooperative Cyber Defence Centre of Excellence
Tallinn, Estonia
firstname.lastname@mil.ee

***Abstract*—With the growth of communication networks, event logs are increasing in size at a fast rate. Today, it is not uncommon to have systems that generate tens of gigabytes of log data per day. Log data are likely to contain information that deserves closer attention – such as security events – but the task of reviewing logs manually is beyond the capabilities of a human. This paper discusses data mining tools SLCT and LogHound that were designed for assisting system management personnel in extracting knowledge from event logs.**

Keywords-event log analysis; data security; data mining

I. INTRODUCTION

Reviewing event logs is an important task for network administrators, security analysts, and other system management personnel. The obtained knowledge can be useful for various purposes, like writing new rules for event log monitoring tools, handling security incidents, etc. However, today's large networks and IT systems can generate large amounts of log data which makes the manual review of event logs infeasible. Therefore, the automation of event log analysis is an important research problem in network and system management.

In order to tackle this problem, data mining techniques have often been proposed. Recently suggested approaches have been mostly based on the Apriori algorithm for mining frequent itemsets [1], and have been designed for mining frequent event type patterns (e.g., see [2, 3, 4]). The suggested approaches have several drawbacks. First, they focus on mining event type patterns, ignoring patterns of other sorts that are potentially interesting. For example, the detection of *line patterns* is important for *syslog* event logs, since this helps the system administrator to write regular expressions for log monitoring tools. Second, proposed approaches are only able to detect frequent patterns, while infrequent events are highly relevant as well in many cases (e.g., in the domain of network security). Third, the Apriori algorithm is known to be computationally inefficient for mining longer patterns [5] that are common for event log data sets.

This paper discusses data mining utilities SLCT and LogHound that address the shortcomings described above. The main design objectives of these tools were lightweight design, modest resource requirements, and flexibility. Like many UNIX utilities and a number of network diagnostic programs (e.g., *grep* or *tcpdump*), SLCT and LogHound are compact command line tools – they produce textual output and consist of just approximately 2,000 lines of C code. They employ several techniques (which are based on special properties of event log data sets) for speeding up their work and reducing their memory consumption. Because of the lightweight design, SLCT and LogHound can be easily integrated with other applications and used in various shell scripts and pipelines. The tools are able to analyze raw event logs and have command line options for preprocessing the logs on-the-fly with regular expressions, in order to facilitate rapid deployment. The tools can be used for traditional off-line knowledge discovery, but also for automated close-to-real-time data analysis. LogHound is a part of the Sisyphus log mining toolkit developed at Sandia National Labs [6], while SLCT was employed by earlier versions of Sisyphus [7].

The remainder of the paper is organized as follows – section II presents an overview of SLCT, LogHound, and the underlying algorithms; section III discusses two case studies of Snort IDS and Cisco Netflow log analysis with SLCT and LogHound; and section IV concludes the paper.

II. OVERVIEW OF SLCT AND LOGHOUND

A. SLCT (Simple Logfile Clustering Tool)

SLCT employs a data clustering algorithm for analyzing textual event logs where each log line represents a certain event. The *data clustering problem* can be defined as follows – divide a set of data points into groups (*clusters*), so that points from the same cluster are similar to each other; points that do not fit well to any of the detected clusters are called *outliers*. SLCT views each event log line as a data point with categorical (non-numeric) attributes, where the *k*-th word of the line is the value of the *k*-th attribute. SLCT does not rely on the traditional distance based clustering approach, since defining an appropriate distance function for categorical data is not trivial, and the notion of distance becomes meaningless in a high-dimensional data space (see [8] for a detailed discussion). Instead, SLCT uses a density based method for clustering – it identifies dense regions in the data space and forms clusters from them, with each cluster corresponding to a certain frequently occurring line pattern. Since outliers are data points dissimilar to clustered points, SLCT is also able to detect *infrequent events* that possibly represent serious anomalies in the behavior of the system.

Informally, the algorithm employed by SLCT can be described as follows (the *support threshold N* is given by the end user; details of the algorithm can be found in [8]):

1. discover frequent attribute values (frequent words) – all attribute values that are present at least *N* times in

This work is supported by SEB Eesti Ühispank.

the data set (identical attribute values belonging to different attributes are counted separately),

2. for each data point extract *all* frequent attribute values from the point and count data points with the same combination of values (note that each combination represents a certain line pattern, e.g., *User * logged in*),
3. combinations that occur at least *N* times in the data set are selected as clusters.

The first two steps of the algorithm both involve a pass over the data set and could consume large amounts of memory for storing attribute values and their combinations with counters. Fortunately, event log data sets have two important properties – although they could contain a large number of words, the majority of words occur very infrequently; also, there are strong correlations between frequent words, and thus not many combinations of these words appear in the data set [8, 9]. SLCT takes advantage of these properties and employs *summary vectors* for reducing its memory consumption [8].

SLCT is written in C and has been tested on Linux and Solaris. Instead of outputting individual lines that belong to each cluster, it prints out cluster descriptions like several other well-known algorithms [8], e.g., *myhost * log: Password authentication for * accepted*. With the *–r* option given, SLCT makes another pass over the data set, in order to check variable parts (*) of cluster descriptions for constant heads and tails. E.g., the cluster description given above would become *myhost sshd[*]: log: Password authentication for * accepted*. If the *–o* option is given, SLCT will also detect outliers during the extra data pass and write them to a separate file.

Other commonly used options are *–s* and *–d* for setting the support threshold and the word delimiter; *–f <fregexp>* for processing only the lines that match the regular expression *<fregexp>*; and *–t <template>* for converting the lines that have matched *<fregexp>* according to the string *<template>*, replacing *$<number>* variables with substring matches. E.g., if *<fregexp>* is *sshd\[[0-9]+\]: (.+)* and *<template>* is *$1*, then the line *sshd[1344]: connect from 192.168.1.1* will be converted to *connect from 192.168.1.1*. The last three options allow the user to configure various preprocessing schemes in a flexible way without writing a separate script. For a complete description of SLCT, please see the online documentation.

B. LogHound

LogHound employs a frequent itemset mining algorithm for discovering frequent patterns from event logs. Although LogHound can mine both line and event type patterns, we will discuss just the first task in this paper for the sake of brevity.

Let $I = \{i_1,...,i_n\}$ be a set of items. If $X \subseteq I$, X is called an *itemset*, and if $|X| = k$ (i.e., X has k items), X is also called a *k-itemset*. A transaction is a tuple (tid, X), where *tid* is a transaction identifier and X is an itemset. A *transaction database* D is a set of transactions, and the *cover* of an itemset X is the set of identifiers of transactions that contain X: $cover(X) = \{tid \mid (tid, Y) \in D, X \subseteq Y\}$. The *support* of an itemset X is defined as the number of elements in its cover: $supp(X) = |cover(X)|$. The *frequent itemset mining problem* is defined as follows – given the transaction database D and the *support threshold* s, find all itemsets with the support s or higher (each such set is called a *frequent itemset*).

Suppose the m-th event log line is *"w_1 w_2 ... w_k"*, where $w_1,...,w_k$ are words from the line (note that the same word can appear more than once). In order to mine line patterns from event logs, LogHound views that line as a transaction (m, X), where $X = \{(w_1,1),...,(w_k,k)\}$. With that representation, each frequent itemset corresponds to a certain frequently occurring line pattern, e.g., the itemset *{(User,1), (login,3), (failure,4)}* corresponds to a pattern *User * login failure*.

Several prominent algorithms have been proposed for frequent itemset mining, most notably breadth-first Apriori [1] and depth-first FP-growth [5] and Eclat [10]. Although FP-growth and Eclat are reported to outperform Apriori [5, 10], they assume that the whole transaction database fits into the main memory. Unfortunately, this assumption does not hold for larger event log data sets [9]. Therefore, LogHound employs Apriori-like breadth-first approach. Efficient Apriori implementations use a memory-resident *itemset trie* data structure during mining – the trie is built layer by layer until it represents all frequent itemsets [9]. However, when the transaction database contains larger frequent itemsets (this is often the case for event log data sets), the itemset trie will consume large amounts of memory and the runtime cost of the repeated trie traversal will be prohibitive [9]. For speeding up its work and reducing its memory consumption, LogHound employs the following techniques:

- in order to reduce the memory cost of mining frequent items (1-itemsets), a summary vector is used,
- most frequently used transaction data are loaded into a memory-based cache,
- a separate pass is made over the data set, in order to detect correlations between frequent items; obtained knowledge is used for building a reduced itemset trie.

The LogHound algorithm is a generalization of Apriori – LogHound falls back to Apriori-like behavior for a certain trie branch if the trie reduction technique is no longer applicable for building that branch; if the technique is not applicable for the entire trie (i.e., there are no strong correlations between frequent items), LogHound is identical to Apriori. Details of the LogHound algorithm and a formal proof that the reduced trie represents all frequent itemsets can be found in [9, 11].

Like SLCT, LogHound is written in C and has been tested on Linux and Solaris. It also shares several command line options with SLCT (*–s*, *–d*, *–f*, and *–t*) and supports event log preprocessing on-the-fly. If the *–c* option is given, LogHound mines only *closed frequent itemsets* (frequent itemsets with no supersets having the same support). For a complete description of LogHound, please see the online documentation.

III. CASE STUDIES

A. Applying SLCT for off-line analysis of Snort IDS logs

Snort [12] is a widely used IDS sensor package that applies attack signatures for detecting suspicious network traffic and can emit alerts as *syslog* messages. IDS systems are known to

generate a large number of alerts – in [13], the authors have found that some attack signatures produce much more alerts than others, and a significant part of these alerts corresponds to harmless network traffic (e.g., SNMP packets from known sources). When analyzing Snort IDS logs, we have observed a similar phenomenon, which makes it harder for a human analyst to spot true positives from the log.

In this case study, we present an example of how to employ SLCT for off-line clustering of Snort IDS *syslog* messages from the past, in order to achieve a compact representation of alert data and ease the task of alert reviewing. A Snort IDS *syslog* message consists of three main fields – signature info (ID, description, classification, priority, and network protocol), source address, and destination address. Thus, we viewed each alert as a data point *(signature info, source address, destination address)*. Source and destination ports were not considered as attributes but rather as address suffixes, since they are missing for some network protocols like ICMP.

The original data set contained 41,706 alerts from a time frame of 24 hours, and we applied SLCT iteratively two times – first for clustering the original data set, and then for clustering outliers from the first run. Choosing the right support threshold for clustering is sometimes not a straightforward task – if the value is too large, a few generic patterns are detected as clusters (e.g., * * *webserver:80*) and there will be many outliers, while for very small values a large number of patterns are found. When experimenting with different values, we finally chose a value of 10 for the first round of clustering, since it represented a good compromise between the number of alert patterns and their clarity. The first round of clustering yielded 402 alert patterns and 930 outliers, while clustering the outliers with the support threshold of 5 yielded 64 patterns and 266 outliers. In other words, the original data set of 41,706 alerts was reduced about 56.9 times – the security analyst has to review just 466 alert patterns and 266 individual alerts.

When inspecting alert patterns, we discovered that many of them represented either false positives or true positives of low importance (e.g., a worm activity against non-existing services). Fig. 1 depicts SLCT command line and some of the more significant alert patterns, while Fig. 2 presents some outliers that deserved closer attention (for reasons of privacy, IP addresses have been obfuscated in Fig. 1–2 and sensitive *syslog* message fields have been removed from Fig. 2).

B. *Applying LogHound for automated close-to-real-time analysis of Cisco Netflow logs*

Cisco Netflow is a widely used protocol for collecting real-time information about forwarded traffic from routers. Routers that have this protocol configured emit *Netflow records* describing the traffic, e.g., a record is sent for a TCP connection when the connection is terminated or has been active (or inactive) for a certain amount of time. The record contains a number of fields, including the IP protocol number, source and destination IP addresses, source and destination port numbers, the number of transferred packets, and the number of transferred bytes. The *collector* receives records emitted by routers and can use them for a variety of purposes, like performance management or intrusion detection.

```
$ slct -f 'snort\[[0-9]+\]: (\[[0-9:]+\]) (.+\}) ([0-9\.:]+) ->
([0-9\.:]+)' -t '$1 $2;$3;$4' -d ';' -s 10 -r -o outliers
/var/log/snort.log

[1:2001219:14] BLEEDING-EDGE Potential SSH Scan [Classification:
Attempted Information Leak] [Priority: 2]: {TCP} * *:22
Support: 79

[1:2002911:1] BLEEDING-EDGE SCAN Potential VNC Scan 5900-5920
[Classification: Attempted Information Leak] [Priority: 2]:
{TCP} * *:5900
Support: 15

[1:2002998:5] BLEEDING-EDGE SMTP HELO Non-Displayable Characters
MailEnable Denial of Service [Classification: Attempted Denial
of Service] [Priority: 2]: {TCP} XXX:6* XXX:25
Support: 20

[1:2002:8] WEB-PHP remote include path [Classification:
Web Application Attack] [Priority: 1]: {TCP} * XXX:80
Support: 13

[1:2000545:3] BLEEDING-EDGE SCAN NMAP -f -sS [Classification:
Attempted Information Leak] [Priority: 2]: {TCP} * XXX:80
Support: 15

[1:2000537:3] BLEEDING-EDGE SCAN NMAP -sS [Classification:
Attempted Information Leak] [Priority: 2]: {TCP} * XXX:80
Support: 15

[1:2002087:7] BLEEDING-EDGE POLICY Inbound Frequent Emails -
Possible Spambot Inbound [Classification: Misc activity]
[Priority: 3]: {TCP} * XXX:25
Support: 16

[1:2001795:7] BLEEDING-EDGE DOS Excessive SMTP MAIL-FROM DDoS
[Classification: Detection of a Denial of Service Attack]
[Priority: 2]: {TCP} * XXX:25
Support: 83

[1:2001611:9] BLEEDING-EDGE F5 BIG-IP 3DNS TCP Probe 3
[Classification: Misc activity] [Priority: 3]: {TCP} * XXX:53
Support: 81

[116:55:1] (snort_decoder): Truncated Tcp Options
{TCP} *:80 XXX:*
Support: 10

[1:2002897:3] BLEEDING-EDGE WEB Horde README access probe
[Classification: access to a potentially vulnerable web
application] [Priority: 2]: {TCP} XXX:48* XXX:80
Support: 5
```

Figure 1. Sample alert patterns.

```
[1:2541:8] SMTP TLS SSLv3 invalid data version attempt
[Classification: Attempted Denial of Service] [Priority: 2]:
{TCP} XXX:57009 -> XXX:25

[1:2002894:2] BLEEDING-EDGE VIRUS W32.Nugache SMTP Inbound
[Classification: A Network Trojan was detected] [Priority: 1]:
{TCP} XXX:63549 -> XXX:25

[1:11837:2] SMTP MS Windows Mail UNC navigation remote
command execution [Classification: Attempted User Privilege
Gain] [Priority: 1]: {TCP} XXX:39992 -> XXX:25

[1:1288:10] WEB-FRONTPAGE /_vti_bin/ access [Classification:
access to a potentially vulnerable web application]
[Priority: 2]: {TCP} XXX:2539 -> XXX:80

[1:2000016:4] BLEEDING-EDGE DOS SSL Bomb DoS Attempt
[Classification: Attempted Denial of Service] [Priority: 2]:
{TCP} XXX:53267 -> XXX:443

[1:2002997:2] BLEEDING-EDGE WEB PHP Remote File Inclusion
(monster list http) [Classification: Web Application Attack]
[Priority: 1]: {TCP} XXX:57120 -> XXX:80

[1:2410:3] WEB-PHP IGeneric Free Shopping Cart page.php
access [Classification: access to a potentially vulnerable
web application] [Priority: 2]: {TCP} XXX:51968 -> XXX:80

[1:2000537:3] BLEEDING-EDGE SCAN NMAP -sS
[Classification: Attempted Information Leak] [Priority: 2]:
{TCP} XXX:17423 -> XXX:25

[1:2000545:3] BLEEDING-EDGE SCAN NMAP -f -sS
[Classification: Attempted Information Leak] [Priority: 2]:
{TCP} XXX:17423 -> XXX:25

[1:5715:2] WEB-MISC malformed ipv6 uri overflow attempt
[Classification: Web Application Attack] [Priority: 1]: {TCP}
XXX:4870 -> XXX:80

[116:58:1] (snort_decoder): Experimental Tcp Options found
{TCP} XXX:37329 -> XXX:80

[116:46:1] (snort_decoder) WARNING: TCP Data Offset is less
than 5! {TCP} XXX:0 -> XXX:0

[122:2:0] (portscan) TCP Decoy Portscan {PROTO255} XXX -> XXX

[122:17:0] (portscan) UDP Portscan {PROTO255} XXX -> XXX
```

Figure 2. Sample outliers.

```
flow-cat /var/log/netflow-last5m | flow-print -f 3 | tail -n +2
| perl -nae 'for ($i = 0; $i < $F[6]; ++$i)
  {print "$F[0] $F[3] $F[1] $F[4] $F[2]\n"; }' > traffic;
loghound -c -s 1% traffic

* * * * 6
Support: 1148855
* * * 25 6
Support: 14557
* * * 80 6
Support: 160091
* * * 443 6
Support: 317494


* * company-web-server 80 6
Support: 25814
company-web-server 80 * * 6
Support: 32993
* * company-web-server 443 6
Support: 133870
company-web-server 443 * * 6
Support: 156638


financial-portal 443 company-proxy-server * 6
Support: 13423
company-proxy-server * newspaper-portal 80 6
Support: 12516
newspaper-portal 80 company-proxy-server * 6
Support: 13074
website 80 company-proxy-server 24481 6
Support: 13642
```

Figure 3. Sample traffic patterns.

Flow-tools [14] is a widely used collector software package which contains tools for generating reports from Netflow data by various criteria (e.g., top source IP addresses by the number of sent bytes). However, in network security it is often unclear *what* to look for and is thus impossible to specify any criterion.

This case study discusses how to mine traffic patterns from Flow-tools logs in an automated close-to-real-time fashion, where instead of specific search criteria a frequency threshold is provided. In our setup, the Flow-tools *flow-capture* daemon receives Netflow records from a border gateway and stores captured data to a binary log file, switching to a new file once in 5 minutes. When a log file for a 5 minute period is complete, a *cron* job (see Fig. 3) converts it to a text file, where for each packet there is a line *source-IP source-Port destination-IP destination-Port IP-protocol*. Then LogHound is applied several times to this file with different support thresholds and a web page is created from its output. Since each detected pattern represents a frequent traffic pattern and its support equals to the number of packets matching the pattern, network and security administrators can get a quick overview of the most prominent classes of network traffic for the last 5 minutes, which allows them to quickly identify DDoS attacks, worm outbreaks, and other intensive anomalous network activity.

Fig. 3 displays a part of the snapshot from the live system for the 1% threshold (for reasons of privacy, IP addresses have been replaced with string tags in Fig. 3). Altogether, 1,170,858 packets were observed within 5 minutes, and LogHound detected 50 network traffic patterns. The first group in Fig. 3 depicts protocol patterns – 1,148,855 TCP packets (IP protocol 6) were observed, and other commonly used protocols were SMTP, HTTP, and HTTPS. The second group of patterns reflects the use of company's e-services by customers over HTTP and HTTPS protocols. The third group describes employee web browsing through the company's proxy server – the first pattern reflects visits to a financial portal, while the next two patterns represent traffic to a popular newspaper portal. The last pattern is the most interesting because of its unusual nature – individual TCP connections to websites normally don't show up as strong traffic patterns, and *website* is not commonly used by company employees. The closer investigation revealed that the pattern corresponds to a legitimate download of a large document file.

IV. Future Work and Availability

For a future work, we plan to experiment with various anomaly detection methods and combine them with SLCT and LogHound, in order to build an event log anomaly detection system. SLCT and LogHound are licensed under the terms of GNU GPL. Both tools and their online documentation are available at *http://kodu.neti.ee/~risto*.

Acknowledgment

The author expresses his gratitude to Mr. Kaido Raiend, Dr. Paul Leis, Mr. Ants Leitmäe and Mr. Ain Rasva from SEB Eesti Ühispank for supporting this work.

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in Proceedings of the 20th International Conference on Very Large Data Bases, 1994, pp. 478–499.

[2] Q. Zheng, K. Xu, W. Lv, and S. Ma, "Intelligent Search of Correlated Alarms from Database Containing Noise Data," in Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS), 2002, pp. 405–419.

[3] S. Ma and J. L. Hellerstein, "Mining Partially Periodic Event Patterns with Unknown Periods," in Proceedings of the 16th International Conference on Data Engineering, 2000, pp. 205–214.

[4] M. Klemettinen, "A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases," PhD Thesis, University of Helsinki, 1999.

[5] J. Han, J. Pei, and Y.Yin, "Mining Frequent Patterns without Candidate Generation," in Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000, pp. 1–12.

[6] Sisyphus Log Data Mining Toolkit, http://www.cs.sandia.gov/sisyphus/.

[7] J. Stearley, "Towards Informatic Analysis of Syslogs," in Proceedings of the 2004 IEEE International Conference on Cluster Computing, 2004, pp. 309–318.

[8] R. Vaarandi, "A Data Clustering Algorithm for Mining Patterns From Event Logs," in Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM), 2003, pp. 119–126.

[9] R. Vaarandi, "A Breadth-First Algorithm for Mining Frequent Patterns from Event Logs," in Proceedings of the 2004 IFIP International Conference on Intelligence in Communication Systems, LNCS Vol. 3283, 2004, pp. 293-308.

[10] M. J. Zaki, "Scalable Algorithms for Association Mining," in IEEE Transactions on Knowledge and Data Engineering, vol. 12(3), 2000, pp. 372–390.

[11] R. Vaarandi, "Tools and Techniques for Event Log Analysis", PhD Thesis, Tallinn University of Technology, 2005.

[12] Snort, http://www.snort.org/.

[13] J. Viinikka, H. Debar, L. Mé, and R. Séguier, "Time Series Modeling for IDS Alert Management," in Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, 2006, pp. 102–113.

[14] Flow-tools, http://www.splintered.net/sw/flow-tools/.