Log Abstraction for Information Security: Heuristics and Reproducibility

Abstract

The collection of log messages regarding the operation of deployed services and application is an integral component to the forensic analysis for the identification and understanding of security incidents. Approaches for parsing and abstraction of such logs, despite widespread use and study, do not directly account for the individualities of the domain of information security. This, in return, limits their applicability on the field. In this work, we analyze the state-of-the-art log parsing and abstraction algorithms from the perspective of information security. First, we reproduce / replicate previous analysis of such algorithms from the literature. Then, we evaluate their ability for parsing and abstraction of log files for forensic analysis purposes. Our study demonstrates that while the state-of-the-art techniques are accurate in log parsing, improvements are necessary in terms of achieving a holistic view to aid in forensic analysis for the identification and understanding of security incidents.

CCS Concepts: • **Security and privacy → Intrusion detection systems**; **Distributed systems security**.

Keywords: application logs, system logs, information security, abstraction

ACM Reference Format:

. 2021. Log Abstraction for Information Security: Heuristics and Reproducibility. In *3rd International Workshop on Information Security Methodology and Replication Studies (IWSMR 2021), August 17–21, 2021, All Digital Conference.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 Introduction

An integral component of network / service operations and management pertains to the identification and understanding of security incidents when problems occur in the hardware/software components of the network. However, the complex interdependence between coupled networking and service functionalities poses a significant challenge in characterizing an alert. This is due to the fact that such messages can be generated by network/system/service elements beyond the actual source of the event. In this research, we are interested in log parsing and abstraction of events and alerts from messages generated by different services for security incident analysis. By way of example, this might include logs of enterprise environments such as web servers (e.g. Apache, NGINX), intrusion detection systems (e.g. Snort, Suricata, Zeek), firewalls (e.g. ufw, iptables), operating system logs (e.g. Windows, macOS, Linux), and core internet protocols (e.g. DNS, HTTP, FTP).

In most operational networks, all messages and alarms from distributed network / service elements are logged with time stamps into data logs. The logs from different elements could be pooled together in a central database for subsequent analysis. While log analysis has been studied in the literature in multiple contexts, we maintain that previous approaches have adopted techniques that fail to address the broad range of log files and/or are not specific enough to security issues. Our goal is to provide holistic insights into threats and malicious activity by way of forensic incident analysis in log pools. In this work, we analyze how the current state-of-the-art in log parsing and abstraction handles security-related logs and how they could be evaluated in such context.

The rest of the paper is organized as follows. Section 2 summarizes related literature while corresponding replication (reproduced) experiments are presented in Section 3. A novel method for evaluating log parsing and abstraction algorithms for security analysis is then presented in Section 4. A set of heuristics for measuring the relevance of the results is proposed and evaluated in Section 5. Finally, conclusions are drawn and future work is presented in Section 6.

2 Related Works

Previous literature has proposed many approaches to automatic log parsing and abstraction by analyzing data from different system/network/service components. In general, three types of data sources are assumed: (i) system logs [20, 21], (ii) network logs [7, 13], and (iii) service/application logs [8, 27]. System logs seem to have received the most attention as a source for automatic log parsing and abstraction for alert detection, root-cause analysis and monitoring. Several commercial and open source tools provide event log monitoring tools. Specific examples include [1–6]. Recently, Zhu et. al. [34] and El Masri et. al. [10] reviewed and compared

IWSMR '21, August 17–20, 2021, All Digital Conference

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

log parsing and abstraction tools/algorithms and compared their outcomes from different perspectives.

Zhu et. al. evaluated 13 algorithms that were introduced between 2003 to 2018 (detailed below). The algorithms are compared according to the execution time, whether they are online/offline, ability to handle large logs, whether they can parse any log or only logs under specific conditions and whether they require pre-processing steps. They also ranked the algorithms according to their performance over a set of logs that were manually annotated. Moreover, they made the code used for comparing the algorithms publicly available. For algorithms that were not open-source, they provided an implementation following the original papers. A summary of their results [34] are given in Table 1. A synopsis of each algorithm is summarized as follows:

SLCT (Simple Logfile Clustering Tool) [31] treats log abstraction as a series of (density-based) grouping tasks. Only two passes over the file are performed, with one used to build a word vocabulary and a second to create candidates for grouping based on the frequency of common words.

AEL (Abstracting Execution Logs) [17] considers word similarity as an indication of event similarity. The task of log abstraction is reduced to a task of finding similar strings based on a distance metric. It also uses an anonymization task to remove easily identifiable words belonging to the log domain.

IPLoM (Iterative Partitioning Log Mining) [22] begins with the assumption that all log messages belong to the same group and iteratively partitions it using heuristics. In total a 4-step partitioning method is assumed, taking into account factors such as the size of the log, the frequency of words, and word/term occurrence.

LKE (Log Key Extraction) [11] is similar to AEL in terms of considering word similarity as an indication of event similarity. It uses an approach that mixes heuristics and hierarchical grouping while adding speedups through parallel operation.

LFA (Log File Abstraction) [25] is built as an attempt to fix SLCT's shortcomings. Like SLCT, it makes two passes over the log file, but builds a frequency table that takes both word and position frequencies into account while grouping.

LogSig [29] treats word frequency and order of appearance as a type of signature for an origin event. As a consequence, the size of a message and positioning of the words are not considered parameters, i.e. word frequency and order of appearance are considered to already encapsulate both concepts.

SHISO (Scalable Handler for Incremental System log) [24] is a 3-step log classification and abstraction algorithm that uses a tree structure generated on-the-fly. Each tree node is filled with a list of words that are extracted from each log record by splitting it at common separators such as spaces, quotes, periods, and others.

LogCluster [33] is an improved version of SLCT, i.e. shortcomings such as word position sensitivity and delimiter noise are addressed. Specifically, a new format for representing log templates is assumed (registers a minimum and a maximum number of words that each wildcard symbol (usually '*') can represent in the template).

LenMa (Length Matters) [28] uses the length of the words in the message as a metric of similarity, as opposed to word distance. Each line is transformed into a vector containing all of the lengths of the words in the message. This is compared to others by calculating the cosine similarity of their respective length vectors.

LogMine [14] was created with the objective of quickly evaluating large sets of log messages, which led to its use of MapReduce. After executing an abstraction step based on domain knowledge, it makes use of the friends-of-friends algorithm to create message clusters.

Spell (Streaming Parser for Event Logs using an LCS) [9] is based on the idea that the longest common sub-sequence (LCS) between log messages is representative of an original event. New messages are placed into existing groups according to matching LCSs, or into new groups if no matching LCS can be found.

Drain [16] is a 5-step algorithm that uses a fixed-depth parse tree, avoiding the construction of long and/or unbalanced trees that negatively impact performance. Each tree node is encoded with a parsing rule, which is used during the algorithms similarity calculation step.

MoLFI (Multi-objective Log message Format Identification) [23] observes that the log abstraction problem has two conflicting objectives: (i) grouping as many messages under the same event, while also (ii) being very thorough when defining new events in order to separate more general events from more specific events. It uses a multi-objective evolutionary search-based approach based on NSGA-II.

Also, El Masri et. al [10] followed a similar approach considering 17 log analysis algorithms. With the exception of LenMa [28],[1] all 12 algorithms from the study of Zhu et. al. were considered. In addition, a further five algorithms were benchmarked: POP [15], HLAer [26], LogHound [32], Nlp-Ltg [18], and Nlm-Fse [30]. These five algorithms are summarized as follows:

POP (Parallel Log Parsing) [15] is built on the observation that most algorithms fail to complete an abstraction task in a reasonable time period, i.e. do not scale to log files over 200 million lines. It has similar properties to IPLoM, but it focuses on a parallel implementation using Spark [2].

HLAer (Heterogeneous Log Analyzer) [26] uses an approach based on hierarchical clustering and pattern recognition. It makes use of a density-based OPTICS algorithm for clustering, the Smith-Waterman algorithm for pattern generation, and the Unweighted Pair Group Method with Arithmetic Mean for generating the event types.

[1] LenMa was not published in a peer-reviewed venue.

Table 1. Evaluations performed by Zhu et. al. [34]

Log Parser	Year	Technique	Mode	Efficiency	Coverage	Preprocessing	Open Source	Industrial Use	Average Accuracy
Drain	2017	Parsing Tree	Online	High	YES	YES	YES	NO	86.5%
IPLoM	2012	Iterative partitioning	Offline	High	YES	NO	NO	NO	77.7%
AEL	2008	Heuristics	Offline	High	YES	YES	NO	YES	75.4%
Spell	2016	Longest common subsequence	Online	High	YES	NO	NO	NO	75.1%
LenMa	2016	Clustering	Online	Medium	YES	NO	YES	NO	72.1%
LogMine	2016	Clustering	Offline	Medium	YES	YES	NO	YES	69.4%
SHISO	2013	Clustering	Online	High	YES	NO	NO	NO	66.9%
LogCluster	2015	Frequent pattern mining	Offline	High	YES	NO	NO	NO	66.5%
LFA	2010	Frequent pattern mining	Offline	High	YES	NO	NO	NO	65.2%
SLCT	2003	Frequent pattern mining	Offline	High	NO	NO	YES	NO	63.7%
MoLFI	2018	Evolutionary algorithms	Offline	Low	YES	YES	YES	NO	60.5%
LKE	2009	Clustering	Offline	Low	YES	YES	NO	YES	56.3%
LogSig	2011	Clustering	Offline	Medium	YES	NO	NO	NO	48.2%

LogHound [32] treats log messages as transactions where each word and its position are grouped into a single token. By measuring frequent occurrences of tokens, LogHound assumes that rare tokens are dynamic values, whereas the others are static, thus generating the event template.

nlp-ltg (Natural Language Processing–Log Template Generation) [18] considers the problem of log abstraction as a problem of labelling sequential data in natural language. It makes use of Conditional Random Fields (CRF) to classify static and dynamic fields.

nlm-fse (Neural language Model For Signature Extraction) [30] trains a character-based neural network to classify static and dynamic fields. It uses four layers, including one to embed context into a word to perform the classification, and a fully connected feed-forward neural network that predicts the event template.

As per the work of Zhu et. al. [34], El Masri et. al. [10] also present an initial categorization of the algorithms according to their characteristics. However, El Masri et. al. use more specific properties, such as whether the algorithm requires pre-processing of the data (yes/no/optional) and the time complexity of the algorithm (computational cost). A summary of their categorization is provided in Table 2.

3 Reproducibility of the state-of-the-art

In order to evaluate the performance of log abstraction algorithms from the perspective of information security – as well as their ability to retain information – we needed to select a set of algorithms that was representative of the state-of-the-art (and available open-source). Furthermore, it was desirable to get most implementations from the same source to reduce the risk of integration errors that could impact the results. With this in mind, the work of Zhu et. al. [34] was assumed for replication purposes. It provides not only the code (available through GitHub on the logpai/logparser repository), but also the annotated (labelled) datasets used for benchmarking the algorithms.

Thus, we replicated their experiments using the code and annotated dataset provided publicly. They measured the performance in the form of classification accuracy as seen in Table 3. The values obtained in our replication experiments using the same dataset and algorithms are presented in Table 4. During our replication study, while analyzing Android logs, LogSig could not produce any results after running for an amount of time comparable to its previous runs over other logs. Therefore, we were not able to obtain LogSig's accuracy for this case.

These results confirm that the replication experiments obtain similar performance in most cases. However, some of the obtained results were significantly different, i.e. more than 10 percentage points over the original. These were highlighted in Table 4. Although it is hard to pinpoint what the causes of the observed differences are, we can assume that some of them are the result of variations between the systems used to perform the experiments. However, differences higher than 10 percentage points are striking given the deterministic nature of some of these algorithms. This seems to indicate that those algorithms producing different results on the same dataset are less robust [19].

4 Impact of Abstraction

As discussed in the previous sections, different tools have been developed to address event log management for specific use cases, such as event monitoring, event archiving, event information extraction, and event querying. In addition, several companies such as Splunk Inc. [6], Apache [2], Elastic [5], and Solarwinds [4] provide log management products for machine generated data. These products potentially facilitate an IT team's daily tasks, but do not necessarily represent a comprehensive end-to-end solution for security forensic analysis or incident reporting. On the one hand, the products developed for log management are generic and efficient, but their analytic modules still have much room for improvement. For example, Splunk is able to collect the logs from the machines and is operating system independent. However, it only provides the most fundamental analytics such as dashboards with basic statistics and keyword-based search. Conversely, the products developed by IT service companies are highly customized and equipped with more advanced analytics, but they are often adherent to their own IT products.

Table 2. Evaluations performed by El Masri et. al. [10]

Algorithm	Pre-processing	Parallel mode	Time complexity	Event types update
Drain	YES	N/A	O(n)	online
IPLoM	NO	NO (parallel mode ready)	O(n)	re-train, offline (batch mode)
AEL	YES	NO	O(n)	re-train, offline (batch mode)
Spell	NO	N/A	O(n)	online
LogMine	OPTIONAL	YES	O(n)	re-train, offline (batch mode)
SHISO	NO	N/A	O(n)	online
LogCluster	NO	NO	O(n)	re-train, offline (batch mode)
LFA	NO	NO	O(n)	re-train, offline (batch mode)
SLCT	NO	NO	O(n)	re-train, offline (batch mode)
MoLFI	YES	NO	$O(n^2)$	re-train, offline (batch mode)
LKE	YES	NO	$O(n^2)$	incremental; generates new event type
LogSig	OPTIONAL	NO	O(n)	re-train, offline (batch mode)
HLAer	NO	NO (parallel mode ready)	$O(n^2)$	online
LogHound	NO	NO	-	re-train, offline (batch mode)
POP	YES	YES	O(n)	re-train, offline (batch mode). Offers solution for new logs detection
nlp-tlg	YES	N/A	O(n)	require re-labelling and re-training
nlm-fse	NO	N/A	O(n)	incremental re-training

Table 3. Performance results given by Zhu et. al. [34]

	HDFS	Hadoop	Spark	Zookeeper	OpenStack	BGL	HPC	Thunderbird	Windows	Linux	Mac	Android	HealthApp	Apache	OpenSSH	Proxifier
Drain	0.998	0.948	0.920	0.967	0.733	0.963	0.887	0.955	0.997	0.690	0.787	0.911	0.780	1	0.788	0.527
IPLoM	1	0.954	0.920	0.962	0.871	0.939	0.824	0.663	0.567	0.672	0.673	0.712	0.822	1	0.802	0.515
AEL	0.998	0.538	0.905	0.921	0.758	0.758	0.903	0.941	0.690	0.673	0.764	0.682	0.568	1	0.521	0.518
Spell	1	0.778	0.905	0.964	0.764	0.787	0.654	0.844	0.989	0.605	0.757	0.919	0.639	1	0.554	0.527
LogMine	0.851	0.870	0.576	0.688	0.743	0.723	0.784	0.919	0.993	0.612	0.872	0.504	0.684	1	0.431	0.517
SHISO	0.998	0.867	0.906	0.660	0.722	0.711	0.325	0.576	0.701	0.701	0.595	0.585	0.397	1	0.619	0.517
LogCluster	0.546	0.563	0.799	0.732	0.696	0.835	0.788	0.599	0.713	0.629	0.604	0.798	0.531	0.709	0.426	0.951
LFA	0.885	0.900	0.994	0.839	0.200	0.854	0.817	0.649	0.588	0.279	0.599	0.616	0.549	1	0.501	0.026
SLCT	0.545	0.423	0.685	0.726	0.867	0.573	0.839	0.882	0.697	0.297	0.558	0.882	0.331	0.731	0.521	0.518
MoLFI	0.998	0.957	0.418	0.839	0.213	0.960	0.824	0.646	0.406	0.284	0.636	0.788	0.440	1	0.500	0.013
LKE	1	0.670	0.634	0.438	0.787	0.128	0.574	0.813	0.990	0.519	0.369	0.909	0.592	1	0.426	0.495
LogSig	0.850	0.633	0.544	0.738	0.200	0.227	0.354	0.694	0.689	0.169	0.478	0.548	0.235	0.582	0.373	0.967
LenMa	0.998	0.885	0.884	0.841	0.743	0.690	0.830	0.943	0.566	0.701	0.698	0.880	0.174	1	0.925	0.508

Table 4. Performance results obtained in the replication study in our work

	HDFS	Hadoop	Spark	Zookeeper	OpenStack	BGL	HPC	Thunderbird	Windows	Linux	Mac	Android	HealthApp	Apache	OpenSSH	Proxifier
Drain	0.998	0.948	0.920	0.967	0.733	0.963	0.887	0.955	0.997	0.690	0.787	0.911	0.780	1	0.788	0.527
IPLoM	1	0.952	0.920	0.967	**0.320**	0.971	0.875	**0.951**	0.568	0.671	0.772	0.691	**0.718**	1	0.733	0.504
AEL	0.998	**0.869**	0.905	0.921	0.758	**0.957**	0.903	0.941	0.690	0.673	0.764	0.682	0.568	1	0.538	0.495
Spell	1	0.778	0.905	0.964	0.764	0.787	0.654	0.844	0.989	0.605	0.757	0.919	0.639	1	0.554	0.527
LogMine	0.850	0.869	0.575	0.687	0.743	0.724	0.784	0.918	0.992	0.611	0.876	0.504	0.686	1	0.430	0.516
SHISO	0.998	0.867	0.906	0.660	0.722	0.711	0.325	0.576	0.701	0.672	0.595	0.585	0.397	1	0.619	0.517
LogCluster	0.546	0.563	0.798	0.731	0.695	0.835	0.787	0.598	0.713	0.628	0.603	0.797	0.530	0.708	0.425	**0.478**
LFA	0.885	0.900	0.994	0.839	0.200	0.854	0.817	0.649	0.588	0.279	0.599	0.616	0.549	1	0.501	0.026
SLCT	0.545	0.422	0.685	0.725	0.867	0.572	0.838	0.882	0.696	0.296	0.557	0.881	0.331	0.730	0.521	0.518
MoLFI	0.997	0.937	**0.570**	0.839	0.213	0.939	0.824	0.651	**0.711**	0.290	0.659	0.741	**0.32**	1	0.541	0.013
LKE	1	0.669	**0.002**	**0.854**	0.787	0.127	**0.845**	0.812	0.989	0.518	0.368	0.908	0.591	1	0.425	0.495
LogSig	**0.508**	0.632	0.543	0.782	**0.838**	0.232	0.382	0.755	0.681	0.107	0.517	-	**0.092**	**0.730**	0.440	**0.493**
LenMa	0.997	0.885	0.883	0.840	0.732	0.689	0.829	0.943	0.565	0.701	0.698	0.879	0.174	1	0.925	0.516

Considering the increasing heterogeneity, scalability and complexity of machine generated log data, designing and developing a comprehensive security event analysis system with a holistic view is not a trivial task.

When we look at the problem of log abstraction from the perspective of security forensics and event analysis, using a holistic view (approach) becomes very important. This enables security analysts to recognize the correlation of different events and minimize gaps in information supporting security forensics. Thus, as the next step, we evaluate the log analysis tools used in our replication study from a holistic perspective. We recognize that conventional network performance metrics might not provide any insight for such a holistic view. Moreover, machine generated data (logs) generally does not include a 'ground truth' either. Therefore, trying to evaluate the algorithms from the perspective of security forensics and event analysis becomes very challenging. Nevertheless, we aim to explore this further by designing new experiments and measurements to address some of the current limitations.

In order to achieve this, we begin by recognizing that we need to be able to measure what we call the *ability to abstract*. An algorithm can be considered "able to abstract", if it can

demonstrate acceptable results when abstracting a set of logs that cover the same domain. To the best of our knowledge, there are no metrics to measure such an ability. Thus, we designed an experiment to be able to observe the ability to abstract by the aforementioned log parsing algorithms. In this experiment, each algorithm is run over three distinct sets of logs, each with its own characteristic (indicating the nature of the log file), as follows:

1. **Homogeneous**: Logs extracted from a single service such as the Apache web server.
2. **Heterogeneous**: Logs extracted from a system running several services such as Linux syslog files (running services like web, printer, file system, etc.).
3. **Mixed Service**: Logs extracted from a range of distinct services, namely HTTP, SSH, DNS, FTP and SMTP, analyzed together (as if they are a single set of logs).

Each of these sets of logs was chosen to represent collection methods commonly present in enterprise deployments. Thus, a log parsing and analysis algorithm needs to be able to abstract the events from these logs into "groups" based on their shared service/application domains and variability. Given that each of the aforementioned algorithms yields a set of groups of logs represented by a template, in a scenario where no common groups can be formed, the algorithm will naturally yield as many groups as there are logs in the set. This means that all logs are different from the algorithm's perspective. In this case, an upper bound is set by the maximum number of groups that can be found for a given set of logs. Given the yielded groups, we obtain a result in the form of the ratio of the yielded groups relative to the upper bound, as seen in Figure 1.

It should be noted that LogSig and SLCT did not execute properly during this experiment. For example, LogSig typically threw exceptions of the form:

```
  Traceback (most recent call last):
  File "homogeneous.py", line 51, in <module>
  result_LogSig = runLogSig(sample, analysis_dir)
  File "/home/ubuntu/log_analysis/algorithms/
    LogSig/execution.py", line 331, in run
    df = parser.parse(file)
  File "/home/ubuntu/log_analysis/algorithms/
    LogSig/execution.py", line 280, in parse
    self.signatConstr()
  File "/home/ubuntu/log_analysis/algorithms/
   LogSig/execution.py", line 200, in signatConstr
    sig = max(candidateSeq[i].items(),
    key=operator.itemgetter(1))[0]
ValueError: max() arg is an empty sequence
```

This seems to indicate that, at some point, LogSig ran out of tokens to parse a log line. This leads to it having an empty sequence, which seems to be the main cause of the exception. SLCT, on the other hand, does finish execution, but it results in a single group with "NONE" for its template. Given these unexpected behaviours, LogSig and SLCT are not considered hereafter.

These results show the abstraction ability of the remaining algorithms from the perspective of how much of the log data they were able to group. For abstraction purposes, the first issue we consider is whether the number of groups generated by an algorithm is more than 50% of the total number of log lines given in a set. If this is the case, it means that there is, at least, one group generated by the algorithm that contains only a single log line. In other words, given an algorithm, A, a set of log lines, L, and the groups generated by A over L, $G(A, L) = [g_0, g_1, ..., g_n]$, we have:

$$|G(A, L)| > \frac{|L|}{2} \implies \exists i \text{ for which } |g_i| = 1$$

A result that is higher than 50% indicates that the algorithm is not able to identify a similarity between the log lines when creating groups. Therefore – given the nature of the logs used for this experiment – such algorithm does not demonstrate "ability to abstract". We called this the 50% rule.

Moreover, when we analyze the output of these algorithms more closely, we recognize that LogCluster yielded results where all message lines belonged to the same group, which was represented by a wildcard template ("*"). In other words, LogCluster was not able to identify any differences between these log messages. We argue that this deems LogCluster unhelpful from a holistic perspective.

In summary, AEL, Drain, LFA, SHISO and Spell are the only algorithms that demonstrate "ability to abstract" and are therefore used in the remaining experiments.

5 Heuristics

In security, it is not only important to identify anomalies and threats, but also to understand the reasons behind them. Let us consider, for example, the following log message:

```
2021-01-01 12:00 - User John logged in to
        machine A using TELNET.
```

There might be several reasons why this message could be considered anomalous: perhaps the use of TELNET as a remote access protocol is uncommon; maybe the user John is not supposed to be accessing machine A; or even because no one should be accessing the infrastructure on January 1st. In summary, different elements in a log message or their correlations could indicate suspicious events and could be used for forensic analysis. Thus, we propose the use of the following heuristics based on the different log message elements:

- **Time** - Events that occur in unexpected or uncommon timestamps. Events outside of working hours, during holidays, or just out of a commonly observed schedule all fit this category.
- **Actor** - Events involving one or more unexpected or uncommon users, devices and its combinations. An

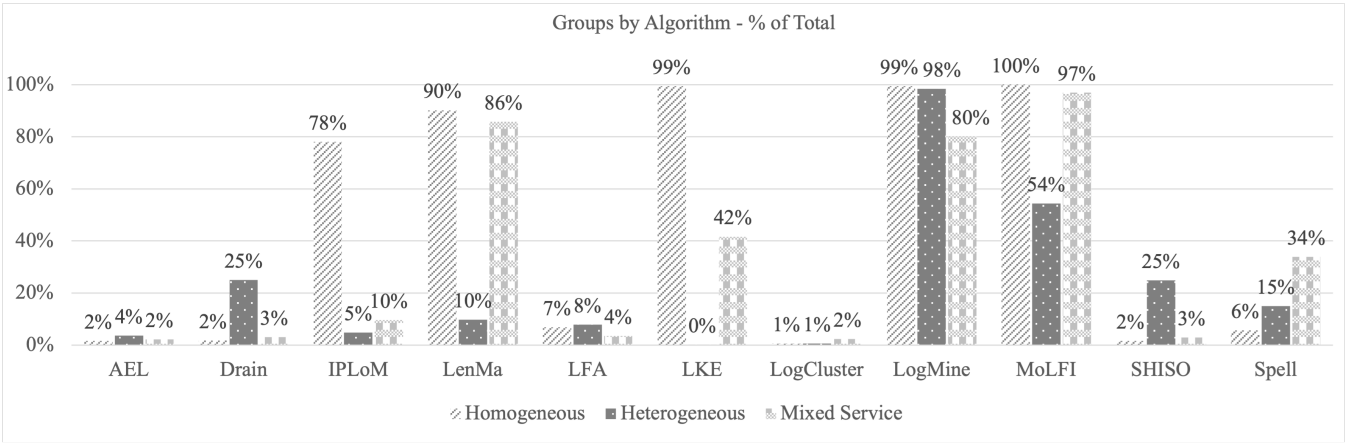

Figure 1. Results on "Ability to Abstract"

unknown user attempting to access a machine, two servers that should not know each other attempting to communicate, or a valid user accessing a valid machine which they are not supposed to, all fit this category.

- **Protocol** - Events that are related to an unexpected or uncommon technology. An access using TELNET instead of SSH, an unencrypted HTTP request to a secure server, or a port scan all fit this category.

Given the above heuristics, we argue that a log abstraction algorithm could be more robust if it follows a heuristics(s) consistently for grouping the log messages. Our reasoning is that an algorithm which is able to follow a heuristic provides a different perspective on *why* it considers a log message to be anomalous. That is, it gives further meaning to the event templates it generates, other than just static measure of field similarity. This is typically referred as explain/interperability and is very important to form a holistic view for an enterprise security system.

5.1 Determining Heuristics

Let's assume a human expert analyzing a set of event templates in order to determine whether they follow a certain heuristic. If so, it is reasonable to assume that they would look for keywords (in log messages) related to one of the heuristics. For example, given an event template:

```
[Apache] Connection * over HTTP
```

It is reasonable to assume that the static words "Apache" and "HTTP" would indicate that this template is based on a *Protocol* heuristic since all of the messages grouped here would have to match this template.

Given that it is not feasible to expect human experts to analyze/categorize every event template generated by a log abstraction algorithm manually, we developed a script to mimic this behaviour, i.e. identifying keywords that indicate the heuristic used for the grouping of log messages by a given log abstraction algorithm employed in our evaluations.

Table 5. Number of times each heuristic is used for abstraction given the different nature of log files

Algorithm	Dataset	Time	Actor	Protocol	Undefined
AEL	Heterogeneous	18 (100%)	0 (0%)	0 (0%)	0 (0%)
AEL	Homogeneous	0 (0%)	2 (25%)	6 (75%)	0 (0%)
AEL	Mixed	0 (0%)	9 (42%)	2 (9%)	10 (47%)
Drain	Heterogeneous	109 (87%)	0 (0%)	4 (3%)	12 (9%)
Drain	Homogeneous	0 (0%)	2 (22%)	7 (77%)	0 (0%)
Drain	Mixed	1 (3%)	11 (37%)	7 (24%)	10 (34%)
LFA	Heterogeneous	39 (100%)	0 (0%)	0 (0%)	0 (0%)
LFA	Homogeneous	0 (0%)	4 (11%)	28 (82%)	2 (5%)
LFA	Mixed	0 (0%)	2 (5%)	3 (8%)	29 (85%)
SHISO	Heterogeneous	120 (97%)	0 (0%)	3 (2%)	1 (1%)
SHISO	Homogeneous	0 (0%)	7 (87%)	1 (12%)	0 (0%)
SHISO	Mixed	0 (0%)	10 (35%)	6 (21%)	12 (42%)
Spell	Heterogeneous	65 (86%)	0 (0%)	1 (1%)	9 (12%)
Spell	Homogeneous	0 (0%)	0 (0%)	28 (100%)	0 (0%)
Spell	Mixed	10 (3%)	221 (67%)	28 (8%)	70 (21%)

To this end, for every event template generated by a log abstraction algorithm, a list of words is made by parsing it on every space. Then, each of these words is compared to regular expressions that are associated with one of the three possible heuristics. No two expressions will accept the same string and strings that do not match any expression get discarded. Finally, the count of words that is associated with each heuristic is compared and a simple majority determines what heuristic is representative of that template. In case of a tie, the template is associated with an "Undefined" heuristic.

5.2 Measuring Heuristics

Having identified a subset of algorithms (Section 4) that have the "ability to abstract", we evaluated them using the proposed heuristics. Table 5 summarizes the results of the five algorithms over the three sets of log files.

It is apparent that there were only a few instances where an algorithm was consistent in terms of what heuristic it used to group the log messages. On the instances where the evaluation is done over homogeneous datasets, we see high

percentages mostly for the "protocol" heuristic, the only exception being SHISO, which favours the "actor" heuristic. On the other hand, when the heterogeneous datasets are evaluated, we see a high percentage of the "time" heuristic being used by these algorithms. For the mixed datasets (extremely heterogeneous), however, results are mostly showing the "undefined" heuristic, i.e. many ties. This seems to indicate that log abstraction algorithms are not able to choose a heuristic as the dataset gets more and more heterogeneous (in terms of the services and applications generating the log messages).

In summary, it seems that no algorithm shows consistency in terms of what heuristic it uses to generate its groups. We argue that this is because of the underlying objective of generating event templates, which are composed of recurring words in log messages. That makes the format and the wording of the log very influential in determining the resulting heuristic. Unfortunately, this might not be helpful from a holistic perspective for analyzing log files. It does not allow a human expert to more easily determine what message groupings are more likely to indicate suspicious behaviours given the policies of an enterprise.

5.3 Heuristics on a security-centered dataset

The results of Section 5.2 appear to indicate that the choice of dataset has a significant impact on the behaviour of each log abstraction algorithm. In order to further evaluate this, we employed additional log files specific to the domain of information security. In particular, we use log files of applications that would be deployed in a small-scale enterprise environment. The logs selected for this experiment include DNS, Firewall, FTP, HTTP, Malformed HTTP packets, SNORT IDS, SSH, SSL, Linux System Logs, macOS System Logs, Windows System Logs[2], and Cyber-attacks over HTTP[3]. To this end, two thousand randomly selected log messages from each log file were used for the following evaluations. The results obtained can be seen in Table 6, where the count of heuristics used are given.

These results seem to indicate that, in most cases, different algorithms choose the same heuristic, namely undefined, in order to group a given log file. For example, all algorithms yield groups with undefined heuristics on SSH and Windows logs. In fact, most of the algorithms yield groups with undefined heuristics on most of the logs used. This again shows their inconsistency in terms of chosen method for abstraction. The few exceptions include Drain on Cyber-attack logs and LFA on macOS logs, that show a majority of groups with a non-undefined heuristic (protocol and time, respectively).

The results suggest that these algorithms are highly dependent on the characteristics of log files. Besides not showing a consistent choice of heuristic over the results of any individual log, no algorithm seems to show any sort of preference

[2]All logs will be made publicly available upon the acceptance of the paper
[3]The Attack Challenge - ECML/PKDD 2007 [12]

Table 6. Number of times each heuristic is used for abstraction using small-scale enterprise information security logs

Algorithm	Dataset	Protocol	Undefined	Time	Actor
AEL	Apache	0	5	4	0
AEL	DNS	0	3	1	0
AEL	Firewall	4	1	1	0
AEL	FTP	0	6	0	1
AEL	HTTP	6	10	0	0
AEL	Linux	0	14	4	0
AEL	macOS	5	2	37	3
AEL	Cyber-attack	3	2	0	0
AEL	SNORT	8	12	1	0
AEL	SSH	1	8	1	0
AEL	SSL	1	3	2	1
AEL	Malformed	1	2	0	0
AEL	Windows	1	13	2	0
Drain	Apache	14	35	45	0
Drain	DNS	8	2	0	1
Drain	Firewall	13	8	1	0
Drain	FTP	0	5	0	13
Drain	HTTP	31	4	0	3
Drain	Linux	5	25	191	1
Drain	macOS	18	75	199	5
Drain	Cyber-attack	76	0	1	0
Drain	SNORT	33	18	9	0
Drain	SSH	4	22	16	0
Drain	SSL	3	2	3	3
Drain	Malformed	6	12	1	14
Drain	Windows	4	19	9	1
LFA	Apache	0	5	31	0
LFA	DNS	20	8	0	0
LFA	Firewall	5	19	1	0
LFA	FTP	2	9	0	8
LFA	HTTP	4	32	0	19
LFA	Linux	0	28	0	0
LFA	macOS	0	0	42	0
LFA	Cyber-attack	7	3	0	0
LFA	SNORT	0	28	0	0
LFA	SSH	0	10	5	0
LFA	SSL	18	7	0	7
LFA	Malformed	0	7	0	6
LFA	Windows	0	24	0	0
SHISO	Apache	0	5	15	0
SHISO	DNS	1	2	0	0
SHISO	Firewall	4	1	1	0
SHISO	FTP	2	4	0	11
SHISO	HTTP	22	6	1	0
SHISO	Linux	3	16	41	1
SHISO	macOS	19	57	201	8
SHISO	Cyber-attack	3	2	0	0
SHISO	SNORT	34	29	14	0
SHISO	SSH	8	10	6	0
SHISO	SSL	2	2	3	3
SHISO	Malformed	1	3	0	0
SHISO	Windows	2	20	14	0
Spell	Apache	2	10	44	0
Spell	DNS	13	248	66	205
Spell	Firewall	3	1	1	0
Spell	FTP	1	7	1	38
Spell	HTTP	1274	23	1	31
Spell	Linux	7	53	200	0
Spell	macOS	38	233	149	9
Spell	Cyber-attack	1461	24	0	0
Spell	SSH	7	18	14	0
Spell	SSL	0	118	53	25
Spell	Malformed	9	22	26	38
Spell	Windows	5	23	6	1

for a heuristic over multiple logs. Thus, for any algorithm, the most used heuristic seems to vary from log file to log file. Lastly, we can observe that even across different algorithms, the same log file tends to be grouped by, mainly, the same heuristics.

5.4 Best Practice Security Recommendations

In the previous sections, we evaluated log parsing and abstraction algorithms for qualities that are essential for best practice security recommendations. To this end, a log parsing algorithm needs to be able to abstract regardless of the nature of a log file, that is, whether it deals with application logs, service logs or systems logs, it should be able to balance their similarities and/or differences as indicated by the 50% rule. Moreover, the algorithm needs to be able to follow a specific heuristic despite the presence of log messages caused by multiple sources (processes). In other words, given a heterogeneous or mixed system/service log file, the resulting groupings identified by the algorithm need to signal a clear heuristic. Finally, the algorithm needs to be able to describe given security-centered data using the previously introduced heuristics (time, actor and protocol). To be able to capture these recommendations in our analysis, we use the following metrics:

1. **Ability to Abstract - AA**: The ability to respect the 50% rule.
2. **Ability to Handle Heterogeneity - HH**: The ability to use produce groups using, in its majority, the same heuristic when analyzing heterogeneous / mixed service log files.
3. **Ability to Describe using Heuristic - DH**: The ability to yield groups that are not associated with an undefined heuristic.

Next, we evaluate the aforementioned log parsing algorithms using these three metrics. The ability to abstract was verified based on the results presented in section 4, Figure 1. The ability to handle heterogeneity was verified based on the results in section 5, Table 5. The ability to describe using heuristics was verified based on the results in section 5.3, Table 6. The evaluation results found are presented in Table 7.

As these results demonstrate, the algorithms explored in this work only partially display the aforementioned abilities for best security practice recommendations. This indicates that more research is necessary to be able to achieve a holistic approach for operational security purposes not only in terms of improved or new algorithms but also in terms of security metrics to evaluate such algorithms.

6 Conclusion

The collection of log messages regarding the operation of deployed hardware/software is an integral component to the identification and understanding of security incidents in any enterprise. The analysis and abstraction of such logs, despite widespread use and study, does not directly account for the individualities of the domain of information security which, in return, limits its applicability on the field.

Table 7. Analysis of algorithms in terms of abilities

Algorithm	AA	HH	DH
AEL	✓	✗	✗
Drain	✓	✗	✗
LFA	✓	✗	✗
SHISO	✓	✗	✗
Spell	✓	✗	✗

In this work, we focused on the analysis of the current log parsing (grouping) and abstraction state-of-the-art algorithms based on not only their performance on annotated data but also in terms of best practice security recommendations. We observed that conventional performance metrics such as computational cost, accuracy and so on are important but not enough to provide insight from a holistic point of view. Thus, we designed and proposed three metrics based on security heuristics including Ability to Abstract, Ability to Handle Heterogeneity and Ability to Describe using Heuristics. Our results replicating the state-of-the-art from the literature show that while 11 of the 17 algorithms could run robustly from the perspective of the performance metric, only 5 of them have the ability to abstract. On the other hand, none has neither the ability to handle heterogeneity nor to describe using security heuristics. Future work will explore improvements to the algorithms to enable them to handle heterogeneity and describe as well as identify heuristics for best practice security recommendations. Moreover, the use of other log files and security metrics will also be studied.

Acknowledgement

This research was enabled in part by support provided by 2Keys Inc. and the Natural Science and Engineering Research Council of Canada (NSERC) Alliance Grant. The first author gratefully acknowledges the support by the province of Nova Scotia. The research is conducted as part of the Dalhousie NIMS Lab at: https://projects.cs.dal.ca/projectx/.

References

[1] 2016. Datadog. https://www.datadoghq.com/
[2] 2021. Apache Spark™ - Unified Analytics Engine for Big Data. https://spark.apache.org/
[3] 2021. Fluentd - Open Source Data Collector. https://www.fluentd.org/
[4] 2021. Log Analysis: Log Management by Loggly. https://www.loggly.com/
[5] 2021. Logstash: Collect, Parse, Transform Logs. https://www.elastic.co/logstash
[6] 2021. Splunk - Cloud-Based Data Platform for Cybersecurity, IT Operations and DevOps. https://www.splunk.com/
[7] Riyad Alshammari and A. Nur Zincir-Heywood. 2012. The Impact of Evasion on the Generalization of Machine Learning Algorithms to Classify VoIP Traffic. In *21st International Conference on Computer*

Communications and Networks, ICCCN 2012, Munich, Germany, July 30 - August 2, 2012. IEEE, 1–8. https://doi.org/10.1109/ICCCN.2012.6289243

[8] Himel Dev and Zhicheng Liu. 2017. Identifying Frequent User Tasks from Application Logs. Association for Computing Machinery, New York, NY, USA.

[9] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.

[10] Diana El-Masri, Fabio Petrillo, Yann-Gaël Guéhéneuc, Abdelwahab Hamou-Lhadj, and Anas Bouziane. 2020. A systematic literature review on automated log abstraction techniques. *Information and Software Technology* 122 (2020), 106276.

[11] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. IEEE, 149–158.

[12] Brian Gallagher and Tina Eliassi-Rad. 2009. *Classification of http attacks: a study on the ECML/PKDD 2007 discovery challenge*. Technical Report. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).

[13] Fariba Haddadi and A. Nur Zincir-Heywood. 2016. Benchmarking the Effect of Flow Exporters and Protocol Filters on Botnet Traffic Classification. *IEEE Syst. J.* 10, 4 (2016), 1390–1401. https://doi.org/10.1109/JSYST.2014.2364743

[14] Hossein Hamooni, Biplob Debnath, Jianwu Xu, Hui Zhang, Guofei Jiang, and Abdullah Mueen. 2016. Logmine: Fast pattern recognition for log analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1573–1582.

[15] Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. 2017. Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing* 15, 6 (2017), 931–944.

[16] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 33–40.

[17] Zhen Ming Jiang, Ahmed E Hassan, Gilbert Hamann, and Parminder Flora. 2008. An automated approach for abstracting execution logs to execution events. *Journal of Software Maintenance and Evolution: Research and Practice* 20, 4 (2008), 249–267.

[18] Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. 2014. Towards an NLP-based log template generation algorithm for system log analysis. In *Proceedings of The Ninth International Conference on Future Internet Technologies*. 1–4.

[19] Duc C. Le and Nur Zincir-Heywood. 2020. A Frontier: Dependable, Reliable and Secure Machine Learning for Network/System Management. *J. Netw. Syst. Manag.* 28, 4 (2020), 827–849.

[20] Chris Lonvick. 2001. RFC3164: The BSD Syslog Protocol.

[21] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2012. A Lightweight Algorithm for Message Type Extraction in System Application Logs. *IEEE Trans. Knowl. Data Eng.* 24, 11 (2012), 1921–1936. https://doi.org/10.1109/TKDE.2011.138

[22] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. 2009. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1255–1264.

[23] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A search-based approach for accurate identification of log message formats. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 167–16710.

[24] Masayoshi Mizutani. 2013. Incremental mining of system log format. In *2013 IEEE International Conference on Services Computing*. IEEE, 595–602.

[25] Meiyappan Nagappan and Mladen A Vouk. 2010. Abstracting log lines to log event types for mining software system logs. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 114–117.

[26] Xia Ning, Geoff Jiang, Haifeng Chen, and Kenji Yoshihira. 2014. 1HLAer: a System for Heterogeneous Log Analysis. (2014).

[27] K Savitha and MS Vijaya. 2014. Mining of web server logs in a distributed cluster using big data technologies. *International Journal of Advanced Computer Science and Applications (IJACSA)* 5, 1 (2014).

[28] Keiichi Shima. 2016. Length matters: Clustering system log messages using length of words. *arXiv preprint arXiv:1611.03213* (2016).

[29] Liang Tang, Tao Li, and Chang-Shing Perng. 2011. LogSig: Generating system events from raw textual logs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*. 785–794.

[30] Stefan Thaler, Vlado Menkonvski, and Milan Petkovic. 2017. Towards a neural language model for signature extraction from forensic logs. In *2017 5th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE, 1–6.

[31] Risto Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*. Ieee, 119–126.

[32] Risto Vaarandi. 2008. Mining event logs with slct and loghound. In *NOMS 2008-2008 IEEE Network Operations and Management Symposium*. IEEE, 1071–1074.

[33] Risto Vaarandi and Mauno Pihelgas. 2015. Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International conference on network and service management (CNSM)*. IEEE, 1–7.

[34] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. 2019. Tools and Benchmarks for Automated Log Parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 121–130. https://doi.org/10.1109/ICSE-SEIP.2019.00021