Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Dynamic log file analysis: An unsupervised cluster evolution approach for anomaly detection[☆]

Max Landauer[a,*], Markus Wurzenberger[a], Florian Skopik[a], Giuseppe Settanni[a], Peter Filzmoser[b]

[a] Austrian Institute of Technology, Austria
[b] Vienna University of Technology, Austria

ARTICLE INFO

Article history:
Received 5 June 2018
Revised 7 August 2018
Accepted 29 August 2018
Available online 4 September 2018

Keywords:
Log data
Cluster evolution
Anomaly detection
String clustering
Unsupervised learning
Incremental clustering
Time series analysis

ABSTRACT

Technological advances and increased interconnectivity have led to a higher risk of previously unknown threats. Cyber Security therefore employs Intrusion Detection Systems that continuously monitor log lines in order to protect systems from such attacks. Existing approaches use string metrics to group similar lines into clusters and detect dissimilar lines as outliers. However, such methods only produce static views on the data and do not sufficiently incorporate the dynamic nature of logs. Changes of the technological infrastructure therefore frequently require cluster reformations. Moreover, such approaches are not suited for detecting anomalies related to frequencies, periodic alterations and interdependencies of log lines. We therefore propose a dynamic log file anomaly detection methodology that incrementally groups log lines within time windows. Thereby, a novel clustering mechanism establishes links between otherwise isolated collections of clusters. Cluster evolution techniques analyze clusters from neighboring time windows and determine transitions such as splits or merges. A self-learning algorithm then detects anomalies in the temporal behavior of these evolving clusters by analyzing metrics derived from their developments. We apply a prototype in an illustrative scenario consisting of a log file containing known anomalies. We thereby investigate the influences of certain parameters on the detection ability and the runtime. The evaluation of this scenario shows that 61.8% of the dynamic changes of log line clusters are correctly identified, while the false alarm rate is only 0.7%. The ability of efficiently detecting these anomalies while self-adjusting to changes of the system environment suggests the applicability of the introduced approach.

1. Introduction

Nowadays, digital systems that exist in all kinds of forms and scales are omnipresent. Despite many benefits that can be drawn from such an interconnected world, the dangers encompassed by recent technological advancements must be recognized. Larger and more complex networks generally entail the emergence of threats and novel attack vectors. Not just the amount of potential entry points becomes larger in a

☆ Massive extension of Landauer et al. (2018)
* Corresponding author.
E-mail addresses: max.landauer@ait.ac.at (M. Landauer), markus.wurzenberger@ait.ac.at (M. Wurzenberger), florian.skopik@ait.ac.at (F. Skopik), giuseppe.settanni@ait.ac.at (G. Settanni), peter.filzmoser@tuwien.ac.at (P. Filzmoser).

https://doi.org/10.1016/j.cose.2018.08.009

growing network, there is also a substantial increase of the attack surface when more complex technologies are present. This allows attackers to infiltrate the system in more diverse and previously unimaginable ways. In order to counteract such intrusions, Cyber Security employs Intrusion Detection Systems (IDS) that are able to differentiate between benign and malicious system processes and raise alerts whenever a prohibited action is executed. However, traditional IDS do so by comparing the current state of the system with known signatures. While the involved methods are usually very efficient, they fail to detect previously unknown attacks due to the fact that no corresponding entry exists in their ruleset. There is therefore a need for more flexible methods that do not rely on predefined rules derived from expert knowledge, but rather detect suspicious events occurring in large-scale ICT systems on their own.

IDS that perform such an unsupervised analysis are known as Anomaly Detection Systems and are frequently used for monitoring system logs. The advantage of log files is that they keep track of every single event that is carried out, including artifacts of attacks. An Anomaly Detection System that is able to process the log lines at least at the same rate as they appear is therefore able to detect attacks in real-time.

Due to the fact that logs are designed to be human-readable, they often contain text messages and also give information about parameters and other values related to the currently running processes. There are uncountable different ways how log files are structured in practice and the contents of most real-world log files exhibit highly different features as they depend on the type of application, configurations defining what type of messages are logged (e.g., informative messages, errors or debug output), the verbosity of the log lines, what kind of components are placed in the system and in which way they are writing their messages to the log file. Moreover, logs from many different sources are often assembled into single files or streams. For example, the syslog protocol only imposes minimal restrictions regarding the log contents when aggregating messages from different services.

This kind of content diversity apparent in many existing applications renders an automated analysis difficult and thus requires methods that provide a more flexible way of extracting relevant data out of the logs.

Several existing approaches do so by employing unsupervised or semi-supervised text clustering approaches that operate independent from the structure of the log file at hand. These methods group similar log lines into a collection of clusters, i.e., a cluster map. However, the cluster maps resulting from these algorithms usually only give a static view of the data. In general, locating outliers in these maps or single lines that contain significant words like "error" is not adequate for a thorough analysis of the system and neither is the presence or absence of certain lines sufficient to indicate problems, but rather the dynamic relationships and correlations between lines have to be considered (Xu et al., 2009).

Note that this kind of clustering is different to clustering log traces, i.e., ordered sequences of log lines, that is frequently pursued in existing literature on process mining. While the extraction of log traces requires some kind of process ID that refers to the task that generated the log messages, clustering individual log lines does not rely on any assumptions about the data. In this article, we therefore refer to static cluster maps as a collection of individual log lines rather than log sequences.

Another challenge with such static cluster maps is that they cannot be used as permanent templates for a computer system. This is due to the fact that any system generating log lines is constantly subject to changes and therefore cluster maps generated during separate time windows often turn out to consist of highly different structures. It is therefore necessary to incorporate dynamic features that span over multiple cluster maps.

This task is known as cluster evolution analysis. Fig. 1 shows an example of three cluster maps generated during three different time windows. In the first time window, the cluster map consists only of a single cluster. This cluster contains a set of log lines displayed as points and is defined by a representative, i.e., a specific element marked by a star that represents the contents of the cluster. In the second time window, two clusters exist, but only one of them is a descendant of the cluster from the first time window. This relationship between the clusters is marked by the arrow pointing from the original to the resulting cluster. In the third time window, three clusters exist, but two of them originate from a single cluster, thereby forming a split.

Cluster evolution aims at an analytical and automatic identification of such transitions between clusters. However, existing cluster evolution techniques rely on the principle that the same elements are observed and clustered over time. Log lines on the other hand are non-recurring objects, i.e., a log line occurs exactly at one single point in time and that same line is never observed again. This means that it is not possible to simply match log lines with each other without previous work, such as identifying and omitting time stamps, IDs and variable artifacts in the strings. We already mentioned that despite the fact that clustering will by definition group similar log lines into clusters, the structure and message content of lines within clusters do not necessarily have to be homogeneous. Even more so, log lines within clusters from different time windows may have structurally changed due to system events or modifications, for example, software updates that change the syntaxes of the logged messages. While fuzzy string matching algorithms exist that alleviate these issues, their extensive computational complexity in combination with the immense amount of log lines distributed in numerous clusters makes it non-trivial to determine the transitions between clusters.

Anomaly detection always relies on some kind of metric that determines whether a specific instance such as a log line, group of log lines or point in time is anomalous or not. Predefined limits are frequently used to trigger alarms for these metrics, however are not always an appropriate solution in an unsupervised setting. This is due to the fact that different systems usually show highly different behavior and also the behavior of a single system changes over time. A self-learning procedure should therefore be able to dynamically adjust to any environment it is placed into and adapt the limits for triggering alarms on its own.

Finally, an anomaly detection system that deals with all the previously mentioned issues must also exhibit a reasonable computational complexity regarding runtime and