Time Series Analysis: Unsupervised Anomaly Detection Beyond Outlier Detection

Max Landauer[1]✉, Markus Wurzenberger[1], Florian Skopik[1], Giuseppe Settanni[1], and Peter Filzmoser[2]

[1] Austrian Institute of Technology,
Austria,
firstname.lastname@ait.ac.at
[2] Vienna University of Technology,
Austria,
peter.filzmoser@tuwien.ac.at

Abstract. Anomaly detection on log data is an important security mechanism that allows the detection of unknown attacks. Self-learning algorithms capture the behavior of a system over time and are able to identify deviations from the learned normal behavior online. The introduction of clustering techniques enabled outlier detection on log lines independent from their syntax, thereby removing the need for parsers. However, clustering methods only produce static collections of clusters. Therefore, such approaches frequently require a reformation of the clusters in dynamic environments due to changes in technical infrastructure. Moreover, clustering alone is not able to detect anomalies that do not manifest themselves as outliers but rather as log lines with spurious frequencies or incorrect periodicity. In order to overcome these deficiencies, in this paper we introduce a dynamic anomaly detection approach that generates multiple consecutive cluster maps and connects them by deploying cluster evolution techniques. For this, we design a novel clustering model that allows tracking clusters and determining their transitions. We detect anomalous system behavior by applying time-series analysis to relevant metrics computed from the evolving clusters. Finally, we evaluate our solution on an illustrative scenario and validate the achieved quality of the retrieved anomalies with respect to the runtime.

Keywords: log data, cluster evolution, anomaly detection

1 Introduction

Recent technological advancements have led to an increase of network communication between computer systems. Unfortunately, this also causes the appearance of novel attack vectors and other previously unimaginable threats. Potential entry points allowing intrusions thereby include legacy systems that are not updated regularly or products that loose vendor support and are insufficiently protected because of outdated security measures.

It is therefore necessary to deploy Intrusion Detection Systems (IDS) that are differentiated between three forms: (i) signature-based detection, a blacklisting approach that compares events with a known set of patterns, (ii) anomaly-based detection, which is able to detect deviations from learned normal system behavior, and (iii) stateful protocol analysis, a whitelisting approach that requires expert knowledge to build a model of allowed system behavior [13]. However, complex computer systems generally require too much effort to be appropriately modeled and blacklisting approaches are not protecting against unknown forms of attacks. Thus, we argue that anomaly detection offers a feasible alternative while being able to flexibly adapt to changing system environments.

Many anomaly detection techniques base on machine learning algorithms that operate in three different settings: (i) supervised, where a training set that contains labeled events both for normal and malicious behavior is analyzed to classify future events, (ii) semi-supervised, where only normal system behavior is provided as training input, and (iii) unsupervised, where no training set is required and learning happens on-the-fly during detection [4]. We recommend an unsupervised approach for several reasons. First, creating a comprehensive labeled data set for supervised algorithms that considers all types of attacks is a difficult task that requires time-consuming manual work and expert knowledge. Second, capturing normal system behavior for semi-supervised algorithms requires anomaly-free environments that can hardly be guaranteed in practice. Finally, dynamic networks that exhibit changing system behavior over time frequently require regenerations of the training data even in anomaly-free settings.

Attacks are usually planned to only show minor visible effects on the system. Fortunately, even very subtle intrusions manifest themselves in log files that record all events taking place in a system. Moreover, it is possible to trace a detected attack to its origin by analyzing the corresponding log lines. Such an investigation on historic data that detects anomalies in hindsight is known as forensic analysis. Contrary to that, online anomaly detection processes the lines as they are generated and identifies anomalies that do not comply with the learned behavior, thereby identifying attacks close to the time when they occur.

There exist norms on what characters are allowed in log data (e.g., RFC3164) and standards that define the syntax of log messages for specific services (e.g., syslog for UDP). However, log files often accumulate logs from multiple services and thus several standards may be mixed together, each of which requiring its own parser. Therefore, a more general approach that employs string metrics for grouping similar log lines independent from their structure is beneficial. Methods that form such cluster maps, i.e., sets of grouped log lines, successfully detected anomalous lines in [17], however provide only a static view on the data. Such existing solutions do not focus their attention on the following challenges:

- Log data is inherently dynamic and thus insufficiently analyzed by static cluster maps. Cluster Evolution (CE) techniques solve this problem by identifying connections between clusters from different maps.
- Anomalous log lines not only differ in their similarity but also relate to sudden changes in frequency, correlation or interruptions of temporal patterns.

- Cluster features, i.e., metrics retrieved from CE, require time-series analysis (TSA) methods for detecting anomalies in their continuous developments.
- Parsers cannot be defined for text-based log lines without known syntaxes and thus string metrics are required for similarity-based clustering.

Therefore, there is a need for dynamic log file anomaly detection that does not only retrieve lines that stand out due to their dissimilarity with other lines, but also identifies spurious line frequencies and alterations of long-term periodic behavior. We therefore introduce an anomaly detection framework containing the following novel features:

- An algorithm for consolidating the evolution of clusters from a continuous and potentially endless series of static cluster maps,
- the computation of metrics based on the temporal cluster developments,
- time-series modeling and one-step ahead prediction for anomaly detection,
- linear scalability on the number of log lines allowing real-time analysis,
- detection of contextual anomalies, i.e., outliers within their neighborhood,
- a realistic scenario evaluating the efficiency and effectiveness of our method.

The paper is structured as follows: Section 2 summarizes the field of CE for anomaly detection. Section 3 gives an overview about the concept of our approach. Sections 4 and 5 explore the theoretical background of CE and TSA respectively. Section 6 contains the evaluation and Section 7 concludes the paper.

2 Related Work

A large amount of research in the field of Cluster Evolution (CE) focuses on graphs (e.g., [3]). With its well-founded theoretical basis that covers both static and dynamic techniques, graph theory is a powerful tool for analyzing many kinds of network structures. For example, social networks conveniently represent graphs and are therefore frequently the target of so-called community evolution analyses. Similarly, the network connections between users within a computer system are often represented as a graph that allows the derivation of several relevant metrics that facilitate reasoning over the current state and behavior of the system. This idea has successfully been extended to anomaly detection by approximating and examining the dynamic development of metrics with time-series models [12]. However, most graph-based algorithms are not designed for a direct application of text-based CE.

When observing clusters over time it is important to identify any occurring changes of individual clusters or the overall cluster structure. Spiliopoulou et al. [15] introduces an algorithm on detecting these changes. Potentially applicable metrics derived from cluster interdependencies are given in [16].

He et al. [8] generate an event count matrix as a template for storing the frequencies of log lines. They then employ machine learning on fixed time windows, sliding time windows and session identifiers in order to identify deviations from the template. Applications that require tracking clusters over time also exist in research areas other than security, such as GPS tracking [9] where groups of points move across a plane. The clusters are described by relevant properties such as size, location and direction of movement, all of which are incrementally updated in every time step. Zhou et al. [19] introduce a similar dynamic collection of cluster features called Exponential Histogram of Cluster Features. Lughofer and Sayed-Mouchaweh [11] discuss an incremental method that supports adding and removing elements from clusters as well as merges and splits that can occur when clusters collide into or move through each other.

Chi et al. [5] suggest to smooth the time-series for retrieving more robust insights into the cluster developments and introduce two frameworks that focus on preserving the cluster quality and cluster memberships respectively. Xu et al. [18] extend these techniques by an evolutionary clustering algorithm. Chakrabarti et al. [2] outline the importance of alignment with snapshots of historical clusterings and propose an adapted hierarchical and K-Means algorithm as a solution.

3 Concept

This section uses an illustrative example to describe the concept of the anomaly detection approach that employs Cluster Evolution (CE) and time-series analysis (TSA). For this, consider log lines that correspond to three types of events, marked with $\bigcirc$, $\triangle$ and $\square$. The bottom of Fig. 1 shows the occurrence of these lines on the continuous time scale that is split up by t_0, t_1, t_2, t_3 into three time windows. The center of the figure shows the resulting sequence of cluster maps $\mathcal{C}, \mathcal{C}', \mathcal{C}''$ generated for each window. Note that in this example the clusters are marked for clarity. Due to the isolated generation of each map it is usually not possible to draw this connection and reason over the developments of clusters beyond one time window. The cluster transitions shown in the top of the figure, including changes in position ($C_\triangle$ in $[t_1, t_2]$), spread ($C_\triangle$ in $[t_2, t_3]$), frequency ($C_\square$ in $[t_2, t_3]$) as well as splits ($C_\bigcirc$ in $[t_2, t_3]$), are thus overseen.

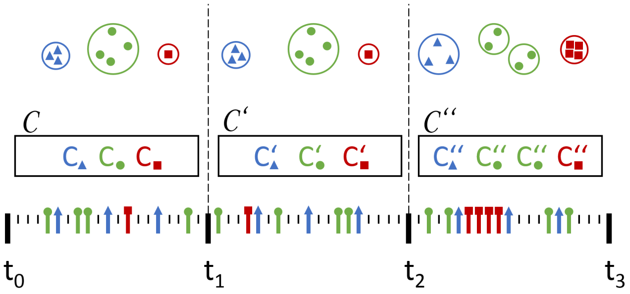

Fig. 1. Bottom: Log lines occurring within time windows. Center: Static cluster maps for every window. Top: Schematic clusters undergoing transitions.

We therefore introduce an approach for dynamic log file analysis that involves CE and TSA in order to overcome these problems (Fig. 2). In step (1), the algorithm iteratively reads the log lines either from a file or receives them as a stream. Our approach is able to handle any log format, however, preprocessing may be necessary depending on the log standard at hand. In our case, we use the preprocessing step (2) to remove any non-displayable special characters that do not comply to the standard syslog format defined in RFC3164. Moreover, this step extracts the time stamps associated with each log line as they are not relevant for the clustering. This is due to the fact that the online handling of lines ensures that each line is processed almost instantaneously after it is generated.

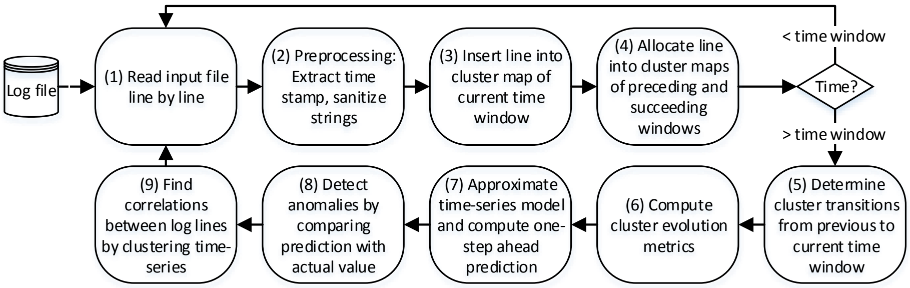
Log file
(1) Read input file line by line
(2) Preprocessing: Extract time stamp, sanitize strings
(3) Insert line into cluster map of current time window
(4) Allocate line into cluster maps of preceding and succeeding windows
< time window
Time?
> time window
(5) Determine cluster transitions from previous to current time window
(6) Compute cluster evolution metrics
(7) Approximate time-series model and compute one-step ahead prediction
(8) Detect anomalies by comparing prediction with actual value
(9) Find correlations between log lines by clustering time-series

Fig. 2. Flowchart of the dynamic clustering and anomaly detection procedure.

Step (3) involves grouping log lines within each time window according to their similarity, resulting in a sequence of cluster maps. It is non-trivial to determine how clusters from one map relate to clusters from the maps created during their preceding or succeeding time windows. Clustering the lines constituting each map into the neighboring maps (4) establishes this connection across multiple time windows and allows the determination of transitions (5). A cluster from one time window evolves to another cluster from the following time window if they share a high fraction of common lines. More sophisticated case analysis is also able to differentiate advanced transitions such as splits or merges.

Several features of the clusters are computed (6) and used for metrics that indicate anomalous behavior. As the computations of these metrics follow the regular intervals of the time windows, we use TSA models (7) to approximate the development of the features over time. The models are then used to forecast a future value and a prediction interval lying one step ahead. If the actual recorded value occurring one time step later does not lie within these limits (8), an anomaly is detected. Figure 3 shows how the prediction limits (dashed lines) form "tubes" around the measured cluster sizes. Anomalies appear in points where the actual cluster size lies outside of that tube.

Finally, the time-series of the cluster properties are also grouped according to their pairwise correlations. An incremental algorithm groups the time-series similarly to the clustering of log lines. Carrying out this correlation analysis in regular intervals allows determining whether time-series that used to correlate with each other over a long time suddenly stop or whether new correlations between clusters appear, which are indicators of anomalous events (9).

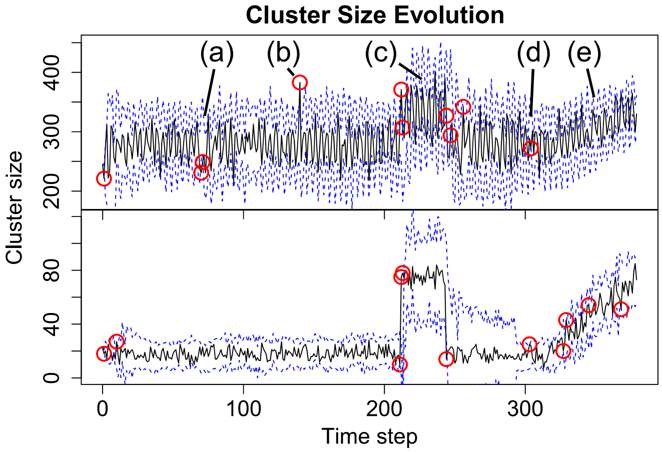

Fig. 3. Time-series representing the sizes of two evolving clusters (black solid lines) with prediction intervals (blue dashed lines) and detected anomalies (red circles). Top: A cluster affected by all anomalies. Bottom: A cluster not affected by periodic events. Anomalies are caused by (a) incorrect periodicity, (b) sudden frequency increase, (c) long-term frequency increase, (e) slow frequency increase. (d) is a false positive.

4 Cluster Evolution

This section describes in detail how online CE is performed on log lines. The approach is introduced stepwise, starting with a novel clustering model that establishes connections between cluster maps. Subsequently, we explain the process of tracking individual clusters and determining their transitions.

4.1 Clustering Model

Considering only the lines of a single time window, we employ our incremental clustering approach introduced in [17]. The procedure is as follows: The first line always generates a new cluster with itself as the cluster representative, a surrogate line for the cluster contents. For every other incoming line the most similar currently existing cluster is identified by comparing the Levenshtein distances between all cluster representatives and the line at hand. The processed line is then either allocated to the best fitting cluster or forms a new cluster with itself as the representative if the similarity does not exceed a predefined threshold t.

This clustering procedure is repeated for the log lines of every time window. The result is an ordered sequence of independent cluster maps $\mathcal{C}, \mathcal{C}', \mathcal{C}'', \ldots$. While the sequence itself represents a dynamic view of the data, every cluster map created in a single time window only shows static information about the lines that occurred within that window. The sequence of these static snapshots is a time-series that only provides information about the development of the cluster maps as a whole, e.g., the total number of clusters in each map. However, no dynamic features of individual clusters can be derived. It is not trivial to determine whether a cluster $C \in \mathcal{C}$ transformed into another cluster $C' \in \mathcal{C}'$ due to the fact that a set of log lines from a different time window was used to generate the resulting cluster. This is due to the nature of log lines that are only observed once in a specific point of time, while other applications employing CE may not face this problem as they are able to observe features of the same element over several consecutive time windows.

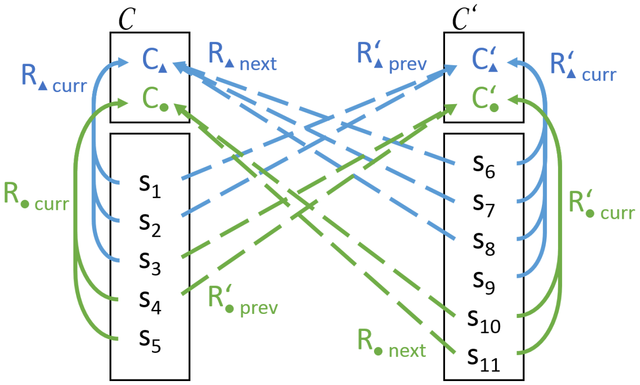

Fig. 4. Solid lines: Construction of cluster map. Dashed lines: Log lines allocated to neighboring map.

In order to overcome the problem of a missing link between the cluster maps, we propose the following model: Every log line is not only clustered once to establish the cluster map in the time window in which it occurred, but is also allocated to the cluster maps created in the preceding and succeeding time windows. These two cases are called construction and allocation phase respectively. The construction phase establishes the cluster map as previously described and each cluster stores the references to the lines that it contains. The allocation phase allocates the lines to their most similar clusters from the neighboring cluster maps. This is also carried out using the incremental clustering algorithm, with the difference that no new clusters are generated and no existing clusters are changed, but only additional references to the allocated lines are stored.

Figure 4 shows the phases for two consecutive cluster maps. The solid lines represent the construction of the cluster maps $\mathcal{C}$ and $\mathcal{C}'$ by the log lines $s_1, ..., s_{11}$ that occurred in the respective time window, e.g., clusters $C_\triangle$ and $C_\bigcirc$ store references to the lines in $R_{\triangle curr}$ and $R_{\bigcirc curr}$ respectively, and $C'_\triangle$ and $C'_\bigcirc$ store their references in $R'_{\triangle curr}$ and $R'_{\bigcirc curr}$. The dashed lines represent the allocation of the lines into the neighboring cluster maps. Clusters in $\mathcal{C}$ store references to allocated log lines from the succeeding time window in $R_{\triangle next}$ and $R_{\bigcirc next}$. Analogously, clusters in $\mathcal{C}'$ store references to allocated log lines from the preceding time window in $R'_{\triangle prev}$ and $R'_{\bigcirc prev}$. Note that in the displayed example, s_3 was allocated to $C_\triangle$ in $\mathcal{C}$ but to $C_\bigcirc$ in $\mathcal{C}'$. Further, s_5 and s_9 are not allocated at all. The following section describes how this model is used for tracking individual clusters over multiple time windows.

4.2 Tracking

For any cluster $C \in \mathcal{C}$ and any other cluster $C' \in \mathcal{C}'$, a metric is required that measures whether it is likely that C transformed into C', i.e., whether both clusters contain logs from the same system process. An intuitive metric that describes the relatedness of C and C' is their fraction of shared members. As previously mentioned, it is not possible to determine which members of each cluster are identical and it is therefore necessary to make use of the previously introduced clustering model that contains references to the neighboring lines. There exists an overlap metric based on the Jaccard coefficient for binary sets introduced in [7] that was adapted for our model by formulating it in the following way:

$$overlap(C, C') = \frac{\left|\left(R_{curr} \cap R'_{prev}\right) \cup \left(R_{next} \cap R'_{curr}\right)\right|}{\left|R'_{curr} \cup R'_{prev} \cup R_{next} \cup R_{curr}\right|} \tag{1}$$

Note that the sets of references R_{curr} and R'_{prev} both correspond to log lines that were used to create cluster map $\mathcal{C}$ and can thus be reasonably intersected, while R_{next} and R'_{curr} both reference log lines from cluster map $\mathcal{C}'$. The overlap lies in the interval $[0, 1]$, where 1 indicates a perfect match, i.e., all log lines from one cluster were allocated into the other cluster, and 0 indicates a total mismatch.

Clusters can also be tracked over multiple time windows by applying the same idea to C' and C'', C'' and C''', and so on. In a simplistic setting where clusters remain very stable over time, this is sufficient for tracking all log line clusters separately. However, in realistic scenarios with changing environments clusters frequently undergo transitions such as splits or merges which negatively influence the overlap and may indicate anomalies. In the following chapter, the tracking of clusters is therefore extended with a mechanism for handling transitions.

4.3 Transitions

Clusters are subject to change over time. There exist internal transitions that only influence individual clusters within single time windows, and external transitions that affect other clusters as well [15]. We consider the cluster size denoted by $|C|$ as the most important internal feature as it directly corresponds to the frequency of log lines allocated to cluster C. Formally, a cluster C grows in size from one time step to another if $|C'| > |C|$, shrinks if $|C'| < |C|$ and remains of constant size otherwise. Alternative internal features derived from the distribution of the cluster members are their compactness measured by the standard deviation, their relative position as well as their asymmetry, i.e., their skewness.

Clusters from different time windows are affected by external transitions. In the following, θ is a minimum threshold for the overlap defined in Equation (1) and θ_{part} is a minimum threshold for partial overlaps that is relevant for splits and merges. In general, partially overlapping clusters yield smaller overlap scores, thus $\theta_{part} < \theta$. We take the following external transitions into account:

1. Survival: A cluster C survives and transforms into C' if $overlap(C, C') > \theta$ and there exists no other cluster $B \in \mathcal{C}$ or $B' \in \mathcal{C}'$ so that $overlap(B, C') > \theta_{part}$ or $overlap(C, B') > \theta_{part}$.
2. Split: A cluster C splits into the parts $C'_1, C'_2, ..., C'_p$ if all individual parts share a minimum amount of similarity with the original cluster, i.e., $overlap(C, C'_i) > \theta_{part}, \forall i$, and the union of all parts matches the original cluster, i.e., $overlap(C, \bigcup C'_i) > \theta$. There must not exist any other cluster that yields an overlap larger than θ_{part} with any of the clusters involved.
3. Absorption: The group of clusters $C_1, C_2, ..., C_p$ merge into a larger cluster C' if all individual parts share a minimum amount of similarity with the resulting cluster, i.e., $overlap(C_i, C') > \theta_{part}, \forall i$, and the union of all parts matches the resulting cluster, i.e., $overlap(\bigcup C_i, C') > \theta$. Again, there must not exist any other cluster that yields an overlap larger than θ_{part} with any of the clusters involved.
4. Disappearance or Emergence: A cluster C disappears or a cluster C' emerges if none of the above cases holds true.

By this reasoning it is not possible that a connection between two clusters is established if their overlap does not exceed θ_{part}, which prevents partial clusters that do not exceed this threshold from contributing to the aggregated cluster in the case of a split or merge. In order to track single clusters it is often necessary to follow a specific "path" when a split or merge occurs. We suggest to prefer paths to clusters based on the highest achieved overlap, largest cluster size, longest time that the cluster exists or combinations of these.

4.4 Evolution Metrics

Knowing all the interdependencies and evolutionary relationships between the clusters from at least two consecutive time windows, it is possible to derive in-depth information about individual clusters and the interactions between clusters. Definite features such as the cluster size that directly corresponds to the frequency of the log lines within a time window are relevant metrics for anomaly detection, however do not necessarily indicate anomalies regarding changes of cluster members.

A more in-depth anomaly detection therefore requires the computation of additional metrics that also take the effects of cluster transitions into account. Toyoda and Kitsuregawa [16] applied several inter-cluster metrics in CE analysis that were adapted for our purposes. For example, we compute the stability of a cluster by $s = \left|R'_{prev}\right| + \left|R_{curr}\right| - 2 \cdot \left|R'_{prev} \cap R_{curr}\right|$, where low scores indicate small changes of the cluster and vice versa. For a better comparison with other clusters, a relative version of the metric is computed by dividing the result by $\left|R'_{prev}\right| + \left|R_{curr}\right|$. There exist numerous other metrics that each take specific types of migrations of cluster members into account.

A simple anomaly detection tool could use any of the desired metrics, compare them with some predefined thresholds and raising alarms if one or more of them exceeds this threshold. Even more effectively, these metrics conveniently form time-series and can thus be analyzed with TSA methods.

5 Time-series Analysis

The time-series derived from metrics such as the cluster size are the foundation for analytical anomaly detection. This section therefore describes how TSA methods are used to model the cluster developments and perform anomaly detection by predicting future values of the time-series.

Model. Time-series are sequences of values associated with specific time points. For our purposes, a time step therefore describes the status of the internal and external transitions and their corresponding metrics of each cluster at the end of a time window. These sequences are modeled using appropriate methods such as autoregressive integrated moving-average (ARIMA) processes. ARIMA is a well-reasearched modeling technique for TSA that is able to include the effects of trends and seasonal behavior in its approximations [6].

Clearly, the length of the time-series is ever increasing due to the constant stream of log messages and at one point will become problematic either by lack of memory or by the fact that fitting an ARIMA model requires too much runtime. As a solution, only a certain amount of the most recent values are stored and used for the model as older values are of less relevance.

Forecast. With appropriate estimations for the parameters, an extrapolation of the model into the future allows the computation of a forecast for the value directly following the last known value. In our experiments an ARIMA model is fitted in every time step and we are interested only in predictions one time step ahead rather than long-term forecasts.

The smoothness of the path that a time-series follows can be highly different. Therefore, neither a threshold for the absolute nor the relative deviation between a prediction and the actual value is an appropriate choice for anomaly detection. Assuming independent and normally distributed errors, the measured variance of previous values is therefore used to generate a prediction interval which contains the future value with a given probability. Using the ARIMA estimate $\hat{y}_t$, this interval is computed by

$$I_t = \left[\hat{y}_t - \mathcal{Z}_{1-\frac{\alpha}{2}} s_e, \hat{y}_t + \mathcal{Z}_{1-\frac{\alpha}{2}} s_e\right] \tag{2}$$

where $\mathcal{Z}_{1-\frac{\alpha}{2}}$ is the quantile $1 - \frac{\alpha}{2}$ of the standard normal distribution and s_e is the standard deviation of the error, $s_e = \sqrt{\frac{1}{n-1} \sum (y_t - \bar{y}_t)^2}$. **Correlation.** Some types of log lines appear with almost identical frequencies during certain intervals, either because the processes that generate them are linked in a technical way so that a log line always has to be followed by another line, or the processes just happen to overlap in their periodical cycles. In any way, the time-series of these clusters follow a similar pattern and they are expected to continue this consistent behavior in the future. The relationship between two time-series y_t, z_t is expressed by the cross-correlation function [6], which can be estimated for any lag k as

$$CCF_k = \begin{cases} \frac{\sum_{t=k+1}^{N} (y_t - \bar{y}) \cdot (z_{t-k} - \bar{z})}{\sqrt{\sum_{t=1}^{N} (y_t - \bar{y})^2} \sqrt{\sum_{t=1}^{N} (z_t - \bar{z})^2}} & \text{if } k \geq 0 \\ \frac{\sum_{t=1}^{N+k} (y_t - \bar{y}) \cdot (z_{t-k} - \bar{z})}{\sqrt{\sum_{t=1}^{N} (y_t - \bar{y})^2} \sqrt{\sum_{t=1}^{N} (z_t - \bar{z})^2}} & \text{if } k < 0 \end{cases} \tag{3}$$

where $\bar{y}$ and $\bar{z}$ are the arithmetic means of y_t and z_t, respectively. Using the correlation as a measure of similarity allows grouping related time-series together. **Detection.** For every evolving cluster, the anomaly detection algorithm checks whether the actual retrieved value lies within the boundaries of the forecasted prediction limits calculated according to Equation 2. An anomaly is detected if the actual values falls outside of that prediction interval, i.e., $y_t \notin I_t$. Figure 3 shows the iteratively constructed prediction intervals forming "tubes" around the time-series. The large numbers of clusters, time steps and the statistical chance of random fluctuations causing false alarms often make it difficult to pay attention to all detected anomalies. We therefore suggest to combine the anomalies identified for each cluster development into a single score. At first, we mirror anomalous points that lie below the tube on the upper side by

$$s_t = \begin{cases} y_t & \text{if } y_t > \hat{y}_t + \mathcal{Z}_{1-\frac{\alpha}{2}} s_e \\ 2\hat{y}_t - y_t & \text{if } y_t < \hat{y}_t - \mathcal{Z}_{1-\frac{\alpha}{2}} s_e \end{cases} \tag{4}$$

With the time period τ_t describing the number of time steps a cluster is already existing we define $\mathcal{C}_{A,t}$ as the set of clusters that contain anomalies at time step t and exist for at least 2 time steps, i.e., $\tau_t \geq 2$. We then define the anomaly score a_t for every time step by

$$a_t = 1 - \frac{\sum_{C_t \in \mathcal{C}_{A,t}} \left(\left(\hat{y}_t + \mathcal{Z}_{1-\frac{\alpha}{2}} s_e \right) \cdot log\left(\tau_t\right) \right)}{\left| \mathcal{C}_{A,t} \right| \sum_{C_t \in \mathcal{C}_{A,t}} \left(s_t \cdot log\left(\tau_t\right) \right)} \tag{5}$$

When there is no anomaly occurring in any cluster at a specific time step, the anomaly score is set to 0. The upper prediction limit in the numerator and the actual value in the denominator ensure that $a_t \in [0, 1]$, with 0 meaning that no anomaly occurred and scores close to 1 indicating a strong anomaly. Dividing by $|\mathcal{C}_{A,t}|$ and incorporating the cluster existence time τ_t ensures that anomalies detected in multiple clusters and clusters that have been existing for a longer time yield higher anomaly scores. The logarithm is used to dampen the influence of clusters with comparatively large τ_t.

Finally, we detect anomalies based on changes in correlation. Clusters which correlate with each other over a long time during normal system operation should continue to do so in the future. In the case that some of these cluster permanently stop correlating, an incident causing this change must have occurred and should thus be reported as an anomaly. The same reasoning can be applied to clusters which did not share any relationship but suddenly start correlating. Therefore, after the correlation analysis has been carried out sufficiently many times to ensure stable sets of correlating clusters, such anomalies are detected by comparing which members joined and left these sets.

6 Evaluation

This section describes the evaluation of the introduced anomaly detection methodology. At first, the attack scenario and evaluation method are outlined. Then the detection capabilities of our method with different values for the similarity threshold and time window size are assessed and discussed.

6.1 Attack Scenario

In order to identify many clusters, we pursue high log data diversity. For this, we propose the following evaluation scenario that adapts an approach introduced in [14]: A MANTIS Bug Tracker System[3] is deployed on an Apache Web Server. Several users frequently perform normal actions on the hosted website, e.g., reporting and editing bugs. At some point, an unauthorized person gains access to the system with user credentials stolen in a social engineering attack. The person then continues to browse on the website, however following a different scheme, e.g., searching more frequently for open issues which simulates suspicious espionage activities. Such actions do not cohere with the behavior of the other users and we therefore expect to observe corresponding alterations in the developments of the log clusters. Due to the fact that only the probabilities for clicking on certain buttons are changed, we expect that the log lines produced by the attacker will be clustered together with the log lines describing normal behavior and that this causes an increase in the measured cluster size. In addition, an automatized program that checks for updates in regular intervals is compromised by the attacker and changes its periodic behavior. In this case, we expect that the changes of the periodic cycles are also reported as anomalies. The injected attacks include one missing periodic pulse, two sudden increases of cluster size with different length and one slowly increasing cluster size.

6.2 Evaluation Environment

The log data was generated on a general purpose workstation, with an Intel Xeon CPU E5-1620 v2 at 3.70 GHz 8 cores and 16 GB memory, running Ubuntu 16.04 LTS operating system. The workstation runs a virtual Apache Web server hosting the MANTIS Bug Tracker System, a MySQL database and a reverse proxy. The log messages are aggregated with syslog. The anomaly detection algorithm was implemented in Java version 1.8.0.141 and runs on a 64-bit Windows 7 machine, with an Intel i7-3770 CPU at 3.4 GHz and 8 GB memory.

6.3 Method

The log data was collected for 96 hours from the previously mentioned Bug Tracker System. Furthermore, sample log lines that correspond to the injected system changes were extracted. These lines were aggregated with their respective occurrence time points in a ground truth table. One of these entries is counted as a true positive (TP) if the algorithm detects an anomalous log cluster with a representative similar to the log line specified in the ground truth table, i.e., the computed string similarity is not smaller than the similarity threshold t used during clustering, and additionally the detection time is not earlier than 30 minutes or later than 60 minutes of the time specified in the ground truth table. If one of these requirements is not met, the entry is counted as a false negative (FN). Detected anomalies that do not correspond to any entries are counted as false positives (FP). True negatives (TN) are determined computationally.

[3] https://www.mantisbt.org/

With this setting, statistically relevant characteristics regarding the quality of the resulting classification were measured. These include the true positive rate ($TPR = \frac{TP}{TP+FN}$), false positive rate ($FPR = \frac{FP}{FP+TN}$), precision ($P = \frac{TP}{TP+FP}$) and recall ($R = TPR$). Plotting the latter two against each other leads to the Receiver Operating Characteristic (ROC) curve, a common evaluation and comparison method for classification systems. Curves are created by running the anomaly detection algorithm with different parameter settings, with well-performing classifiers being located in the top-left corner of the ROC diagram (high TPR, low FPR). We also added the first median as it describes the performance of a random guesser and every reasonable classifier has to lie above this line. Finally, also the well-known $F_1 - score = \frac{2 \cdot P \cdot R}{P+R}$ is computed.

6.4 Results

Figure 3 shows the cluster size developments of two log line clusters, the one-step ahead prediction limits forming tubes around the curves and the anomalies that are detected whenever the actual size falls outside of this tube. The present types of anomalies in the plot are: (a) a periodic process skipping one of its peaks, (b) a spike formed by a rapid short-term increase in line frequency, (c) a plateau formed by a long-term frequency increase, (d) a false positive and (e) a slowly increasing trend. The curve in the top part of the figure corresponds to a cluster affected by all injected anomalies. While anomalies (a)-(c) are appropriately detected, anomaly (e) is not detected in this cluster because the model adapts to the slow increase of frequency that occurs within the prediction boundaries, thereby learning the anomalous behavior without triggering an alarm. We intentionally injected (e) in order to show these problems that occur with most self-learning models. These issues can be solved by employing change point analysis methods that detect long-term changes in trends [10]. The bottom part of the figure corresponds to a cluster containing only log lines that are specifically affected by anomalies (c) and (e). Accordingly, the anomalies manifest themselves more clearly and the high deviations from the normal behavior makes their detection easier. The fact that each of the numerous evolving clusters are specific to certain log line types is a major advantage of our method. In particular, more than 300 evolving clusters representing more than 90% of the total amount of log lines were identified.

The anomaly score aggregated over all evolving clusters that exist for at least 20 time steps is displayed in Fig. 5. The figure clearly shows that the anomaly score increases at the beginning and end of every attack interval. This corresponds to the fact that our algorithm detects changes of system behavior, but almost immediately adapts to the new state. Only returning from this anomalous state to the normal behavior is again detected as an anomaly.

Different parameters were used to create the ROC curves displayed in Fig. 6. In the left plot, the similarity threshold $t \in [0, 1]$ from the incremental clustering procedure was varied. A high similarity threshold causes that only highly similar lines are allocated to the same cluster, i.e., the total number of clusters per time window increases. A low similarity threshold causes the opposite. We discovered

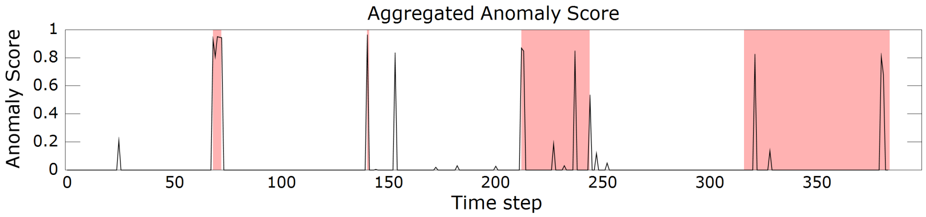

Fig. 5. The aggregated anomaly score displayed as a time-series and correctly increasing when the system behavior changes (red shaded intervals).

that low similarity thresholds ($t < 0.5$) cause too many different log line types being grouped into the same clusters and the cluster representatives therefore not appropriately describing their content. This in turn leads to mismatching clusters between the time windows that do not reach the minimum required threshold for establishing a connection.

The curves were created by changing the prediction level $(1 - \alpha)$, i.e., the width of the prediction interval, with narrow tubes leading to higher TPR and FPR and broad tubes leading to lower TPR and FPR. Favorable values (high TPR, low FPR) are located close to the top-left corner of the plot. The figure shows that a moderate width is superior to the extremes as they suffer from either low TPR or high FPR. All threshold values yield reasonably good performances in the ROC plot because our injected anomalies always manifest themselves in multiple clusters, but there is a preference towards thresholds around 0.85 achieving $TPR = 61.8\%$ with only $FPR = 0.7\%$. In general, higher thresholds enable an increased granularity and should therefore be preferred for detecting anomalies that only affect a single or few log line types.

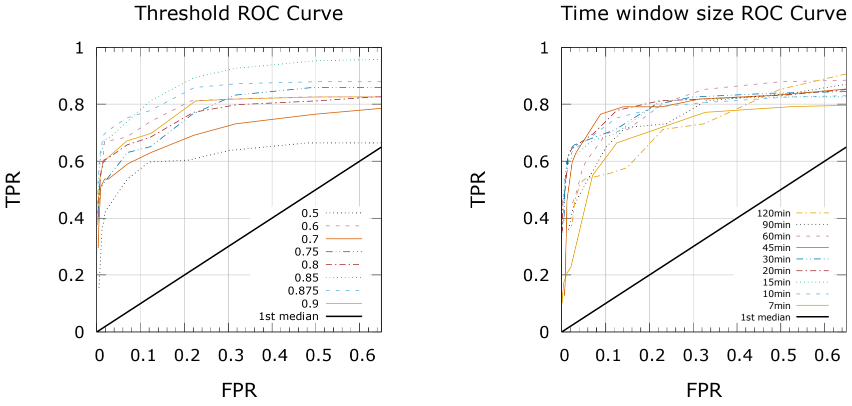

Fig. 6. Left: ROC curves for different threshold values. Right: ROC curves for different time window sizes.

The top left part of Fig. 7 shows the runtime with respect to different threshold values. Moderate threshold values yield lower runtimes then values closer to 0 or 1. The top right part of the figure shows that the runtime scales linearly with the number of log lines, which is important for processing continuous streams.

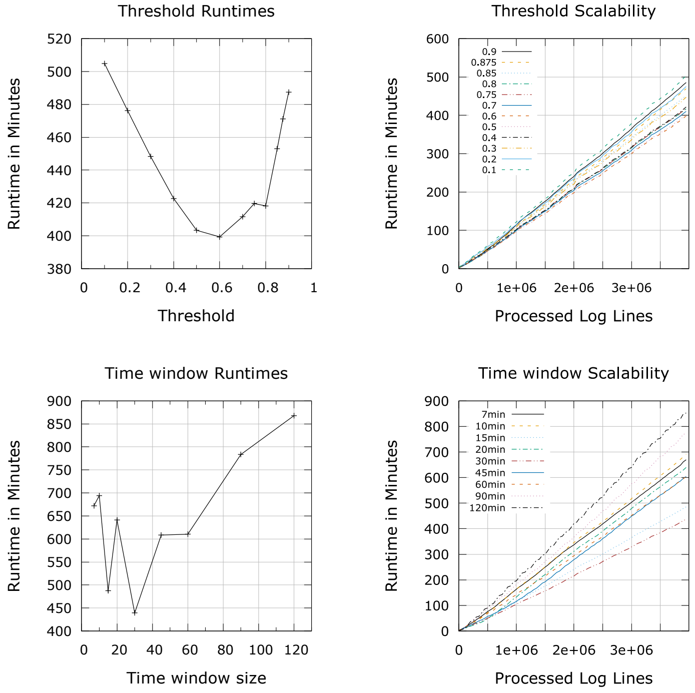

Fig. 7. Left: Runtime comparison for different parameter settings. Right: Runtime measured with respect to the number of processed log line shows linear scalability. Top: Threshold as changed parameter. Bottom: Time window size as changed parameter.

In addition to the threshold, the influence of the time window size was investigated. The right side of Fig. 6 shows ROC curves where the same data set was analyzed with a similarity threshold of 0.9 and varying time window sizes. The curves indicate that good results are achieved with time window sizes similar to the attack durations (10-30 minutes). In general, very large time windows are not sufficiently fine-grained and therefore easily miss anomalies that only occur during very short intervals. Clearly, smaller time windows yield finer granularities (i.e., more time steps in any given period) and also reduce the average reaction time, i.e., the average amount of time that passes between an anomaly occurring and being detected ($\frac{time\ window}{2}$). On the other hand, time windows smaller than the appearance frequency of certain log line types may result in incomplete cluster maps that do not contain evolving clusters of these logs. Thus, the correct choice for the time window size largely depends on the log frequencies.

Finally, the measurements regarding the runtime are shown in the bottom part of Fig. 7. Time window sizes that performed well in the ROC analysis also showed low runtimes, because generating the time-series model is easier when the time window is aligned to the period. Again, the runtimes scaled linearly with the number of log lines independent from the size of the time window.

For brevity, we only discuss the results of the evaluation centered around the $F_1 - score$ but omit the plots. The results showed that the recall increases for a higher threshold almost up to 1. Moreover, the size of the prediction interval had a clear influence on the recall for any given threshold level, with smaller sizes increasing the achieved recall score. This is due to the fact that actual anomalies fall outside of the tube more easily and thus improve the recall. While the precision also improves with a higher threshold, the results showed just the opposite characteristic regarding the prediction interval size, with large tubes increasing the precision. This is due to the fact that from all the detected anomalies, only highly diverging points that are likely to be actual anomalies exceeded the limits of the tube. For high similarity thresholds, precision scores between 0.2 and 0.3 are reached. Only when precision and recall are combined in the $F1 - score$ the superiority of moderate tube sizes over the extremes becomes apparent. These observations emphasize the importance of the tube size and confirm the superiority of higher similarity thresholds already ascertained in the ROC analysis.

7 Conclusion and Future Work

In this work we introduced a dynamic anomaly detection algorithm for log data. By deploying an incremental clustering algorithm on multiple time windows rather than the whole data, we were able to establish a sequence of static cluster maps that collectively represent dynamic system behavior. We used cluster evolution techniques in order to identify developments of single clusters and employed time-series analysis for detecting anomalous deviations of relevant metrics.

The evaluation showed that clusters formed by groups of log lines belonging to a certain event are successfully tracked over time. Furthermore, the results showed that injected anomalies manifested themselves as sudden changes in the generated time-series and were appropriately detected by our algorithm.

We computed the overlap between cluster maps from two neighboring time windows. However, the quality of the connections between clusters could be enhanced by taking more distanced time windows into account. Moreover, there exist other time-series models able to predict future values, some of which may show a higher precision or runtime enhancements compared to ARIMA models.

As most unsupervised self-learners, our model suffers from poisoning of the data, i.e., anomalous behavior affecting future detections [1]. For example, regularly occurring log lines from malicious processes are learned after some time. An attacker is able to exploit this vulnerability by carefully injecting log lines that slowly adapt the learner to the changed system behavior. We are planning to investigate methods for change point analysis in order to solve these issues.

Acknowledgment

This work was partly funded by the FFG project synERGY (855457).

Bibliography

[1] Biggio, B., Rieck, K., Ariu, D., Wressnegger, C., Corona, I., Giacinto, G., Roli, F.: Poisoning behavioral malware clustering. In: Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop. pp. 27–36. ACM (2014)

[2] Chakrabarti, D., Kumar, R., Tomkins, A.: Evolutionary clustering. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 554–560. ACM (2006)

[3] Chan, J., Bailey, J., Leckie, C.: Discovering correlated spatio-temporal changes in evolving graphs. Knowledge and Information Systems 16(1), 53–96 (2008)

[4] Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR) 41(3), 15 (2009)

[5] Chi, Y., Song, X., Zhou, D., Hino, K., Tseng, B.L.: On evolutionary spectral clustering. ACM Transactions on Knowledge Discovery from Data (2009)

[6] Cryer, J., Chan, K.: Time Series Analysis: With Applications in R. Springer Texts in Statistics, Springer (2008), https://books.google.at/books?id=MrNY3s2difIC

[7] Greene, D., Doyle, D., Cunningham, P.: Tracking the evolution of communities in dynamic social networks. In: Advances in social networks analysis and mining (ASONAM). pp. 176–183. IEEE (2010)

[8] He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: System log analysis for anomaly detection. In: Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on. pp. 207–218. IEEE (2016)

[9] Jensen, C.S., Lin, D., Ooi, B.C.: Continuous clustering of moving objects. IEEE Trans. Knowl. Data Eng. 19(9), 1161–1174 (2007)

[10] Killick, R., Fearnhead, P., Eckley, I.A.: Optimal detection of changepoints with a linear computational cost. Journal of the American Statistical Association 107(500), 1590–1598 (2012)

[11] Lughofer, E., Sayed-Mouchaweh, M.: Autonomous data stream clustering implementing split-and-merge concepts–towards a plug-and-play approach. Information Sciences 304, 54–79 (2015)

[12] Pincombe, B.: Anomaly detection in time series of graphs using arma processes. Asor Bulletin 24(4), 2 (2005)

[13] Scarfone, K., Mell, P.: Guide to intrusion detection and prevention systems (idps). NIST special publication 800(2007), 94 (2007)

[14] Skopik, F., Settanni, G., Fiedler, R., Friedberg, I.: Semi-synthetic data set generation for security software evaluation. In: Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on. pp. 156–163. IEEE (2014)

[15] Spiliopoulou, M., Ntoutsi, I., Theodoridis, Y., Schult, R.: Monic: modeling and monitoring cluster transitions. In: Proceedings of the 12th ACM

SIGKDD international conference on Knowledge discovery and data mining. pp. 706–711. ACM (2006)

[16] Toyoda, M., Kitsuregawa, M.: Extracting evolution of web communities from a series of web archives. In: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia. pp. 28–37. ACM (2003)

[17] Wurzenberger, M., Skopik, F., Landauer, M., Greitbauer, P., Fiedler, R., Kastner, W.: Incremental clustering for semi-supervised anomaly detection applied on log data. In: Proceedings of the 12th International Conference on Availability, Reliability and Security. p. 31. ACM (2017)

[18] Xu, K.S., Kliger, M., Hero Iii, A.O.: Adaptive evolutionary clustering. Data Mining and Knowledge Discovery 28(2), 304–336 (2014)

[19] Zhou, A., Cao, F., Qian, W., Jin, C.: Tracking clusters in evolving data streams over sliding windows. Knowledge and Information Systems 15(2), 181–214 (2008)