Journal of Network and Computer Applications 208 (2022) 103515

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

Micro2vec: Anomaly detection in microservices systems by mining numeric representations of computer logs

Marcello Cinque [a,1], Raffaele Della Corte [a,*,1], Antonio Pecchia [b,1]

[a] Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Università degli Studi di Napoli Federico II, Via Claudio 21, Napoli, 80125, Italy
[b] Dipartimento di Ingegneria, Università degli Studi del Sannio, Palazzo Bosco Lucarelli C.so Garibaldi 107, Benevento, 82100, Italy

ARTICLE INFO

Keywords:
Log mining
Anomaly detection
Microservice
Monitoring

ABSTRACT

This paper describes a study on log mining in the domain of microservices technologies. We focus on the detection of anomalies from logs, i.e., events requiring deeper inspection by analysts. Log mining is challenging in microservices systems due to the high number of heterogeneous logs. We present Micro2vec, a novel approach to mine numeric representations of computer logs without making assumptions on the format of underlying data and requiring no application knowledge; representations computed by Micro2vec are suited for anomaly detection. To cope with the lack of publicly-available datasets of labeled logs from production systems, we validate our approach by means of a mixture of direct measurements from logs, one-class classification experiments and generation of log variants. The study has been conducted in the context of a Clearwater IP Multimedia Subsystem setup consisting of microservices deployed in Docker containers, and on a real-world critical information system from the Air Traffic Control domain, which implements a communication model typically used with microservices.

1. Introduction

Microservices represent an evolution of the service-oriented architecture (SOA) paradigm (Wolff, 2016). They put forth *reduced size*, *independency*, *flexibility* and *modularity* principles, which well cope with ever-changing business environments; as such, they are strongly intertwined with the current industry mainstream in Agile and DevOps (Bass et al., 2015). We observe that architectural advantages brought by microservices pose novel challenges to **log analysis**, which has been extensively used over the past decades to evaluate production systems (Oliner et al., 2012; Cinque et al., 2016, 2020b). A **log** is a sequence of text *lines* stored in a file, reporting information on the runtime behavior of a computer system.

Microservice systems' deployments tend to create a distinct log *per*-microservice, which causes substantial *cognitive work* by human experts in traversing the logs in order to pinpoint and correlate relevant lines for **forensics** and **troubleshooting**. Moreover, current trends put forth the composition of microservices by different vendors – and thus differently skilled development teams – that exacerbates format and semantic **heterogeneity** of logs. In consequence, it becomes hard for practitioners to maintain exhaustive catalogues of *keywords*, *regular expressions* or *correlation rules*, which are typically used to monitor runtime logs in many state-of-art log management and SIEM[2] tools, such as Logstash[3] or Splunk.[4] Microservices "bring heterogeneity of distributed systems to its maximum expression" (Dragoni et al., 2017): a real-world deployment can easily reach *tens* of microservices and – in turn – log files.

This paper describes our study on log mining in microservice systems. We take a different perspective from current research trends in microservices, which leverage OS metrics, e.g., *CPU usage* and *free memory*, as opposite to text logs (Fadda et al., 2016; Thalheim et al., 2017). We claim that microservices exacerbate the role of event logs: their analysis is extremely relevant to practitioners since they provide more fine-grained data with respect to OS metrics. To the best of our knowledge there is a lack of approaches that leverage logs in this

* Corresponding author.
E-mail addresses: macinque@unina.it (M. Cinque), raffaele.dellacorte2@unina.it (R. Della Corte), antonio.pecchia@unisannio.it (A. Pecchia).

[1] All authors equally contributed to this work.
[2] Security Information and Event Management (Kavanagh et al., 2016; Cardenas et al., 2013).
[3] https://www.elastic.co/products/logstash.
[4] http://www.splunk.com/.
[5] The **label** – also known as *ground truth* – pertains to the knowledge of the events occurred in a certain system when the logs were collected.
[6] We denote by **event** an operating condition, i.e., either normal or anomalous, of a given system; the occurrence of an event may be reported by one or more lines in the log(s) of the system.

https://doi.org/10.1016/j.jnca.2022.103515
Received 30 May 2022; Received in revised form 19 August 2022; Accepted 6 September 2022
Available online 1 October 2022

domain. More importantly, no *publicly-available* datasets of *labeled*[5] logs are available in this context. The label is particularly important, as it is typically used to train/test classifiers for anomaly detection. We present a specific application in this context, related to the detection of **anomalies** (or anomalous events) from logs, i.e., events[6] requiring deeper inspection by analysts (D'Amico and Whitley, 2008; Cinque et al., 2020a), such as *failures* and *misuse*. According to D'Amico and Whitley (2008), the detection of anomalies is one of the first steps foreseen in computer security defense analysis. The automatic detection of these events allows reducing the manual work required to dig into logs, in case of incidents. The proposed approach has been applied in a Clearwater IP Multimedia Subsystem (IMS) setup consisting of microservices deployed in Docker containers, which well-represents a typical microservices setup. Further, to address threats to validity concerns, the proposal has been applied in the context of a critical information system from the Air Traffic Control (ATC) domain, which implements a communication model typically used with microservices.

The contribution of the paper is *twofold*. First, we propose **Micro2vec**, a novel log mining approach that embeds no application knowledge, makes no assumptions on the format/semantics of logs, and requires no *a-priori* catalogues of anomalies' symptoms/patterns to cope with the challenges mentioned above. This is pursued by computing **numeric representations** of heterogeneous computer logs, which allows inferring "actionable" relationships for anomaly detection. Second, we aim to overcome the scarceness of publicly available labeled logs in this context, which is a long-standing research challenge in many *field data* studies. In this respect, we address the validation of the proposed approach through a *mixture* of experiments that involve both (i) direct collection of logs related to *normal* and *anomalous* events in a controlled testbed, and (ii) the synthesis of **log variants** by means of perturbation of the content of the normal logs. Log variants are generated by solely looking at normal logs, with no knowledge on anomalies. Most notably, we generated a comprehensive sample of labeled logs, which we made publicly available for research purposes. We mitigate the threats to validity of our detection approach by analyzing the data with different techniques, including multi-class and one-class classification, and classification done on the top of log variants. The **key outcomes and findings** of our study – with respect to the data and systems in hand – are:

- *Micro2vec, a novel mining approach that is able to capitalize on the availability of multiple – potentially distributed – logs*. At the time being, microservices are often designed to trust each other; once gained access to *one* microservice, an attacker might bring down an entire application (Dragoni et al., 2017). Nevertheless, real-life anomalies will likely involve many microservices within an application (e.g., the Netflix compromise described in Sun et al. (2015)). As such, we claim that the ability to read "across the lines" of multiple logs is a crucial feature in this context as well as in any distributed system.
- *Anomalies do reflect into the relationships of the metrics across different logs*. As a confirmation of the first finding, we discovered that different events are characterized by precise signatures, which reflect into complex patterns across microservices' logs. We pave the way for discovery approaches in inferring explicable detection rules that cannot be caught by human experts.
- *Variants obtained from normal logs can be used to detect **real** anomalies*. Differently from some recent contributions in this area (Cao et al., 2016; Xie et al., 2018), we generate *log variants* with no knowledge of anomalies, which makes our approach potentially independent from the availability of labeled logs. Moreover, the obtained results suggest that the usage of log variants provides an advantage with respect to one-class approaches in detecting real anomalies.

The rest of the paper is organized as follows. Section 2 covers the related work in the area. Section 3 discusses Micro2vec, while Section 4 introduces the Clearwater case study. Section 5 describes the datasets collected in Clearwater, and the procedure to generate log variants. Section 6 presents a discussion of the implications of the approach, while Section 7 describes the results obtained in the context of Clearwater. Section 8 describes the ATC case study and related results. Section 9 discusses the threats to validity, while Section 10 concludes the work.

2. Related work

This section summarizes the related work in the area of microservices monitoring, mining metrics for event detection and security applications.

2.1. Microservices monitoring

Commercial and open-source tools are available for microservices monitoring. For example, Dynatrace, AppDynamics, CA and New Relic are commercial Application Performance Management (APM) tools that leverage source code instrumentation to monitor microservices. Instana is a commercial APM solution that leverages the *span data* model (Benjamin et al., 2010) to trace all the requests generated by properly instrumented microservices.

A similar approach is used in Zipkin (2022), the open-source distributed tracer for microservices developed by Twitter. Sysdig (Sysdig, 2022) is a container-native monitoring solution, which allows collecting resource usage, network statistics, as well as tracing applications running inside containers. Netflix Hystrix (Netflix Hystrix, 2022) is a latency and fault tolerance Java library designed to prevent cascading failures in distributed systems.

A transparent tracing for microservice-based applications is presented in Santana et al. (2019), which leverages proxies for relieving the burden of tracing activities from applications. The work Noor et al. (2019) presents a generic monitoring framework for applications based on multi-virtualization (e.g., containers/VMs). The framework includes agents, collecting system-level statistics, and a manager that retrieves and analyzes the collected data.

A microservices monitoring tool, named *MetroFunnel*, is presented in Cinque et al. (2022). The tool aims to accompany microservices logs with passive tracing in order to support informed decisions by practitioners. In Brandón et al. (2020) a root cause analysis framework for microservice architectures is presented. It performs anomaly detection leveraging a graph representation of the system architecture, obtained by different data sources, e.g., network activity and resource usage.

MicroRCA detects performance issue and locates their root causes in microservices using application and system level metrics, e.g., resource usage and response time (Wu et al., 2020). *Sage* (Gan et al., 2020) performs root cause analysis for cloud microservices, leveraging Causal Bayesian Networks and Graphical Variational Auto-Encoder to model dependencies between microservices, latency propagation, and detect QoS violations. In Srirama et al. (2020) a microservices scheduling strategy for cloud environments is proposed. The approach leverages a monitoring mechanism based on system-level metrics (i.e., CPU and memory usage) to monitor the resources used by microservices (on both physical and virtual machines) for resource scaling.

2.2. Mining numeric metrics for anomaly detection

Extracting *OS*, *middleware* and *application* metrics from logs and other monitoring tools is valuable for many applications. In Farshchi et al. (2018) is presented an anomaly detection technique for sporadic cloud operation. The technique correlates event logs and cloud metrics to detect anomalies during operation. An anomaly detection technique

leveraging Natural Language Processing is proposed in Bertero et al. (2017); analysis is performed through the Google *word2vec* algorithm.

An approach for mining console logs to detect runtime problems in large-scale systems is presented by Xu et al. (2008). The approach extracts structured information from console logs and constructs vectors of features. In Campos et al. (2018) a study on the use of machine learning for supporting failure prediction is presented. The study uses datasets containing failure and non-failure data, which consist of numeric features representing the system behavior. Zoppi et al. (2016) presents an anomaly detection approach for Service-Oriented Architectures (SOAs), which copes with SOAs dynamicity, by collecting metrics at different system layers.

In Ahuja et al. (2021) a number of metrics (e.g., per flow packet count, packet rate) are extracted from network flows, which are then used to detect DDoS attacks in software defined networks by using machine learning models. The work in Behal et al. (2018) proposes a DDoS attack detection approach that leverages the generalized entropy metric. Network flows collected at different points of an Internet Service Provider network are translated in entropy observations, which are then combined to detect DDoS attacks.

2.3. Security applications

Several approaches have been proposed for *retaining* relevant security data by means of feature extraction, attribute enrichment and various classifiers. For example, the filtering technique in Spathoulas and Katsikas (2010) makes use of a statistical approach to combine different features, such as number of occurrences, frequency of signatures, and prior knowledge regarding the alerts. Alerts are filtered using a threshold-based approach. The work (Bakar et al., 2005) proposes an attribute enrichment approach. It leverages a set of quality parameters among traditional features, which are used to compute a score for classifying alerts.

The outlier detection algorithm in Fu et al. (2010) addresses filtering through the use of *weights*, which highlight the importance of the attributes of alerts, e.g., the destination port or type. Frequent pairs of attribute-value are used as features to discriminate false positives. Differently from this literature, Julisch and Dacier (2002) puts forth the idea that alerts belong to a limited number of clusters, which are inferred through a generalization hierarchy. Work by Valeur et al. (2004) discards irrelevant events based on active monitoring; every time an alert is triggered, it is analyzed to identify potential vulnerabilities the attacker is trying to exploit.

In Cinque et al. (2017) is presented an entropy-based security analytics approach, which aims to measure the occurrence of interesting activities. A hierarchical approach to mine high threat alarms from logs generated by Intrusion Prevention System is presented in Meng et al. (2018). The approach in Du et al. (2017), named *DeepLog*, uses a deep neural network to model a system log as a natural language sequence. DeepLog learns patterns from normal executions in order to detect anomalies. In Yang et al. (2019) is proposed *nLSALog*, an anomaly detection framework that leverages log files as data source. The framework models the log as a natural language sequence and uses Long Short-Term Memory (LSTM), built using nominal training data, to detect security anomalies.

The work in Liu et al. (2019) presents a heterogeneous graph embedding based approach for cyber threats detection, named *log2vec*. The approach converts log entries into a heterogeneous graph, which is used to detect malicious log entries. *OmegaLog* (Hassan et al., 2020) is a tracker that merges application and system logs to build a Universal Provenance Graph (UPG). The UPG combines the causal reasoning strengths of whole-system logging with the rich semantic context of application logs to improve the reasoning of investigators about the nature of attacks.

ADA (Adaptive Deep Log Anomaly Detection) (Yuan et al., 2020) allows the detection of security-related anomalies in system logs, leveraging LSTM and dynamic adaptive thresholds. *ReLog* (Luo et al., 2020) leverages reinforcement learning techniques to perform the analysis of MPI logs for anomalous user detection. The Authors also present a synthetic data generation method based on Generative Adversarial Networks (GAN) to face the lack of anomalous data for training. Similarly, in Cao et al. (2016) is presented a framework for the generation of attack variants. An attack is represented by an event sequence. Given a sequence of events in an attack, these events are replaced with interchangeable events to generate new sequences, which represent attack variants.

A similar method is introduced in Xie et al. (2018), using an existing attack dataset for training a GAN, which is then used for the attack variants generation. It is important to note that, differently from our log variants generation method, Luo et al. (2020) generates variants in terms of feature vectors instead of log entries, Cao et al. (2016) requires a detailed analysis of past observed attacks, while Cao et al. (2016) and Xie et al. (2018) generate variants from existing attack data instead of using normal data.

2.4. Our contribution

Notwithstanding the amount of work and tools for microservices monitoring, to the best of our knowledge there is a lack of approaches that leverage logs in this domain. Recent work started to address monitoring challenges in deploying Virtual Machines (VMs), which – similarly to *containers* – can be used to host microservices. Given the large availability of Cloud providers, Authors in Fadda et al. (2016) propose a multi-objective mixed integer linear optimization approach to maximize the *quality* of monitoring, addressing the collection of metrics, such as *CPU usage*, *free memory* and *power consumption*. Similarly, in Thalheim et al. (2017) is presented *Sieve*, i.e., a platform to analyze the communication between containers hosting microservices. Sieve capitalizes on OS metrics and *sysdig*, which requires a kernel module to observe the system calls used by microservices. While metrics are widely-used in networked systems for anomaly detection (Ibidunmoye et al., 2018), we take a different perspective by analyzing text logs. In this respect, most microservices monitoring approaches require service instrumentation to pinpoint anomalies. On the other hand, logs are a by-product of the system's execution and are ubiquitously emitted by almost any software components, such as microservices. For example, the framework in Zuo et al. (2020) uses system logs to detect anomalies in microservice architectures, but it also needs the collection of query traces, which requires instrumentation or placement of a collector inside each microservice. Our contribution, instead, is a *non intrusive* approach, based on the use of unmodified logs.

3. Proposed approach

Our approach, named **Micro2vec**,[7] consists in transforming distributed microservices logs into **numeric representations**. This is done with no application knowledge, no assumptions on the format/semantics of logs, and no *a-priori* catalogues of anomalies.

Let us consider a system composed by the microservices M_j ($1 \leq j \leq S$), such as shown by Fig. 1. Overall the microservices generate a given number of *logs* denoted by L_i ($1 \leq i \leq N$). Logs are seen as *streams* of text lines and undergo three tasks, i.e., *sampling*, *scoring* and *detection*.

Sampling is a periodic task where logs are continuously sampled at regular intervals. Given a log L_i, every T time units the task acquires the new lines in L_i generated by the microservice during the past T, creating a *chunk*. The specific choice of T for our study is discussed in Section 4. Sampling is done individually for each L_i: therefore, the task

[7] The *2vec* suffix is used since the proposal leverages vector representation as done by other existing **2vec* approaches (Mikolov et al., 2013; De Koninck et al., 2018).

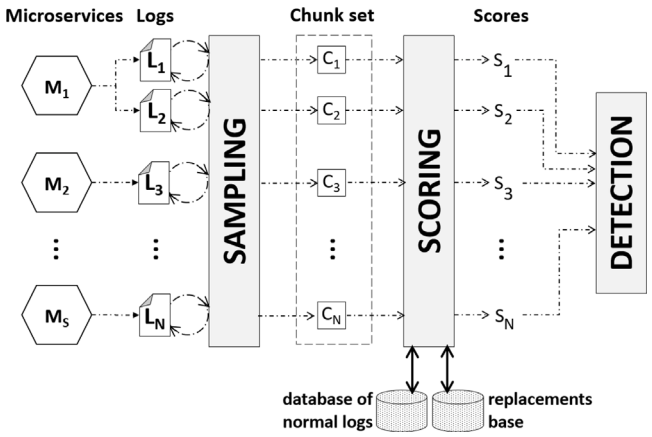


Fig. 1. Overview of the tasks of Micro2vec.

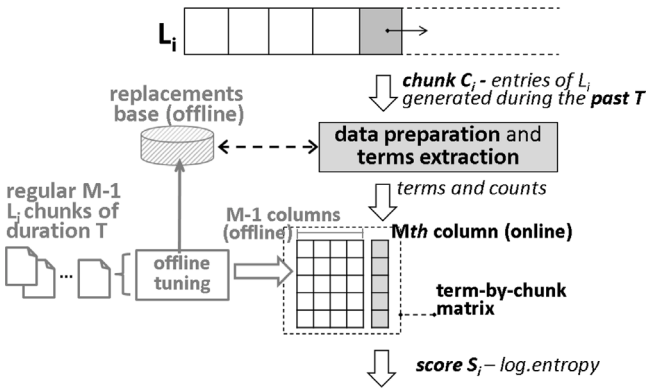


Fig. 2. Scoring task details.

generates a new set of chunks – **chunk set** hereinafter – at every T, encompassing chunks C_i corresponding to the logs L_i.

The **scoring** task computes a quantitative metric – **score** hereinafter – for each chunk in the *chunk set*. There exist many methods for extracting metrics from logs, such as (Farshchi et al., 2018; Bertero et al., 2017; Xu et al., 2008; Cinque et al., 2017; Stearley and Oliner, 2008). After having reviewed the methods, we selected the **log.entropy**, i.e., the term weighting technique used in Cinque et al. (2017) and Stearley and Oliner (2008), which was demonstrated to be effective for handling text logs. Although other methods might have been adopted for scoring, *digging* into these methods is not the purpose of our study; rather we focus on the use of log-driven numeric metrics in a novel application domain.

Fig. 2 details the scoring task of Micro2vec. For each chunk C_i, the scoring task (i) performs a *data preparation* (which consists of parsing the logs and clearing special characters, as described in Section 3.1), (ii) extracts all the terms – a term is a sequence of characters separated by whitespace(s) – from the chunk, (iii) counts the occurrences of each term within the chunk, and (iv) computes a numeric score, i.e., the log.entropy.

The output of the *scoring task* is a **vector of scores** S_i ($1 \leq i \leq N$), i.e., one score per chunk in the *chunk set*; a new vector is generated at every T. Each score of the vector summarizes the past T time unit of one log. The vector organizes the scores computed during the same sampling round, which gives Micro2vec the ability to read 'across' the logs. The use of vectors allows leveraging the relationships across the logs to detect anomalies. The **detection** task is continuously fed with vectors of scores, with each vector representing a chunk set; it determines whether a chunk set represents normal or anomalous events based on the values of the scores. The proposal is not tailored to a specific detection algorithm. There exist many methods for detecting anomalies from scores, e.g., threshold-based approaches, use of classifiers. We discuss the detection approach in Section 6.

3.1. Data preparation

The data preparation step aims to mitigate the inherent variability of log lines, which are fraught with very infrequent terms (often occurring just once), such as timestamps, value of variables, and process identifiers.

A typical log line contains both *invariant* and *variable* fields, as can be seen in the following example:

```
16:29:39 Status sip_connect: Recycle TCP connection slot 14
16:29:43 Status sip_connect: Recycle TCP connection slot 2
```

It can be noted that the log lines share a common **pattern**, which can be inferred by removing the variable fields, i.e., the timestamp and the slot number in the example:

```
* Status sip_connect: Recycle TCP connection slot *
```

The presence of variable fields might distort the scoring task, which aims to assign larger scores to infrequent terms. Data preparation is a core component and a well-consolidated practice in log analysis. For instance, in the scoring approach proposed in Stearley and Oliner (2008) all terms that occur once are discarded; similar considerations are in Lim et al. (2008) and Kobayashi et al. (2018).

The scoring task (i) removes special non-alphanumeric characters (e.g., #, ?, ;, and %) from the logs and (ii) implements a *stop-word list* to avoid conventional terms, such as names of days/months. More important, the scoring task replaces each line with the corresponding pattern, leveraging a *replacements base*, which is populated offline (*offline tuning* in Fig. 2) with the patterns automatically inferred from the logs generated by the system under normal activity.[8] Each line is replaced with the corresponding pattern, if available in the base; otherwise it is left unchanged.

3.2. Computation of the score

Each chunk C_i sampled from L_i is "prepared" as described above. We then extract the pairs *terms-counts*, i.e., the number of occurrences of each distinct term in the chunk after preparation. Let x_t denote the count of the term t in the chunk C_i, with $1 \leq t \leq W$ and W the total number of distinct terms in C_i. We obtain the log.entropy of C_i as $log.entropy = \sqrt{\sum_{t=1}^{W}(e_t \cdot \log_2(1 + x_t))^2}$, where log.entropy ≥ 0. Please note that e_t is the **entropy** of the term t over a set of M prepared chunks (*term-by-chunk matrix* in Fig. 2), i.e., C_i plus (M-1) baseline chunks stored in a database of *normal logs* produced *offline* by microservices under normal operations. In detail, let $x_{t,k}$ denote the count of the term t in the chunk k, with $1 \leq k \leq M$ and $x_{t,M} = x_t$ (i.e., the count extracted from C_i). The entropy is computed as follows:

$$e_t = 1 + \frac{1}{\log_2(M)} \sum_{k=1}^{M} p_{t,k} \log_2(p_{t,k}) \quad (1)$$

$$p_{t,k} = \frac{x_{t,k}}{\sum_{k=1}^{M} x_{t,k}} \quad (2)$$

where $p_{t,k}$ represents the probability to have the term t in the chunk k. As it can be inferred from Eqs. (1) and (2), terms that occur regularly across the M chunks have a low score. The result of the scoring task for a given log C_i is a new observation of the score S_i, which is forwarded to the detection task along with the scores computed for all the logs.

[8] Patterns are automatically obtained through log parsing. We implement an iterative token-sanitization algorithm consistent with state-of-the-art approaches (Vaarandi, 2008), which leverages a clustering-based approach to identify variable fields and infer patterns.

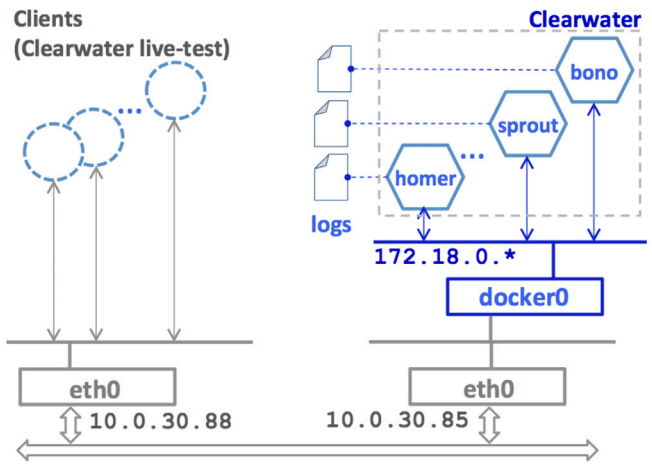


Fig. 3. Experimental setup of Clearwater.

The value of (M-1), i.e., the number of baseline chunks, has been sized in such a way that the normal chunks' log.entropy mean can be characterized with 10% accuracy at 90% confidence, according to procedures in Jain (1991).

4. Experimental setup

We use a **Clearwater** IMS setup as main case study. Clearwater implements the standard IMS architecture, which is adopted by large telcos for IP-based *voice*, *video* and *messaging* services (Clearwater, 2022). Clearwater and IMS-related technologies have been also used in research studies, such as Cotroneo et al. (2017), Di Mauro and Liotta (2019) and Nguyen et al. (2018). We use a version of Clearwater composed by 11 microservices deployed in as many Docker containers on the same virtual machine. We describe some of the microservices referenced by this paper, while a comprehensive architectural view of Clearwater can be found at Clearwater (2022):

- *bono* is the Session Initiation Protocol (SIP) edge proxy providing a WebRTC interface to clients;
- *sprout* is the SIP router that handles client authentication;
- *cassandra* is the database of profile data;
- *homestead* is a RESTful CRUD server that allows retrieving authentication credentials and users profile;
- *homer* is a standard XDMS (XML Document Management Server) used to store MMTEL (MultiMedia TELephony) service settings documents;
- *ellis* represents the provisioning portal.

We use **Clearwater-live-test** (Clearwater test, 2022) to generate a representative load for exercising Clearwater. *Clearwater-live-test* is a well-consolidated suite of *Ruby* test programs. The tests used for our experiments consist in: (i) *test setup*, i.e., registering a certain number of telephone accounts; (ii) issuing the actual sequence of service invocations to Clearwater (the number and type of invocations vary across the tests); (iii) *test finalization*, which deletes the telephone accounts.

The experimental setup is shown in Fig. 3. Clearwater's Docker microservices are hosted by an Ubuntu 16.04.03 LTS OS, Intel Xeon E5-2630L v3 2.9 GHz, 4 Gb RAM server. Microservices are connected through the *docker* LAN `172.18.0.*`, which is bridged with the physical LAN `10.0.30.*`. As such, clients hosted on a different machine, which run instances of the above-mentioned *Clearwater-live-test*, are allowed to reach the microservices.

Overall 13 **log files** by Clearwater's microservices are considered in this setup, as listed in Table 1 by microservice generating at least one log. Each log contains a variety of lines, which encompass received

Table 1
Clearwater's log files by microservice.

Microservice	Log sources	Microservice	Log sources
`astaire`	`astaire.txt`	`bono`	`bono.txt`
`cassandra`	`system.log`	`chronos`	`chronos.txt`
`ellis`	`ellis.txt`	`homer`	`homer.txt`
Microservice	**Log sources**		
`homestead-prov`	`homesteadprov.txt`		
`homestead`	`homestead.txt`, `homesteadaccess.txt`		
`ralf`	`ralf.txt`, `ralfaccess.txt`		
`sprout`	`sprout.txt`, `sproutaccess.txt`		

requests, status messages, errors and exceptions occurred at runtime, and so forth. Examples of collected normal and anomalous log lines are shown in Fig. 4. It is important to note that the anomalous log lines, i.e., the ones from `bono.txt` (lines 6–7) and `sprout.txt` (lines 9–11), have been collected during anomalous events settings (which will be detailed in Section 5), and report occurred error/failure. Logs are periodically sampled to generate the *chunk sets* for our scoring/detection approach as described in Section 3. In our study, the sampling period is set to $T = 30$ s, which represents a tradeoff between the latency of the detection and the need for ensuring a reasonable number of lines per chunk, while the sample size is set to $(M\text{-}1) = 120$.

5. Data collection

Our datasets encompass (i) logs collected by means of direct observations of Clearwater during normal and anomalous events, and (ii) *log variants* that we synthesize by perturbing the content of normal logs for validation purposes.

Please note that – although the experiments are conducted in a controlled environment – we reproduce a mixture of realistic operating scenarios by means of a well-consolidated test suite, i.e., *Clearwater-live-test*. This is done to overcome the lack of publicly-available datasets for the system *in-hand* as mentioned above.

5.1. Normal events and anomalies

Normal logs are obtained by exercising Clearwater with the *live-test* suite. All the collected *chunk sets* are labeled as *normal*, and used to build the replacements base. Fig. 5 shows the scores of 4 out of 13 Clearwater's logs – for the sake of better readability – over around one hour of normal operations; the x-axis is the progressive id of the *chunk set* since the beginning of the collection. It can be noted that the mixture of test cases within *Clearwater-live-test* exposes various normal combinations of the scores. For example, *homesteadprov* is reasonably stable through all the execution if not for few sporadic spikes; *ellis* can be either or not close to *homesteadprov*, as shown in between the *chunk sets* 30 and 70; finally, the score of *homer* appears generally higher than *ellis*. We capitalize on modeling these relationships across the logs for anomaly detection.

Other than normal operations, we also collect logs during further five settings, where each collection setting reproduces an **anomaly**, such as a *bruteforce* authentication, *misuse* of Clearwater's functions or *tampering* with OS processes and resources. All the *chunk sets* collected during a setting are labeled with the related anomaly. Overall, we collected around 387 Mb of data, which account for more than 2 millions of log entries. We reproduce the anomalies during the execution of *Clearwater-live-test* (configured using default values for its parameters in both normal and anomalous events settings), so that these events are *overlapped* with regular operations as it would occur in production. The choice of the events is inspired by previous studies, such as the security attack phases in Ruiu (1999) and Sharma et al. (2011). Table 2 summarizes anomalies; each anomaly is mapped to the attack phase and it is accompanied by a description that explains how the anomaly

Table 2
Anomalous events settings in Clearwater.

Anomaly	**Phase** as in Ruiu (1999) and Sharma et al. (2011)
Bruteforce authentication (AUTH)	**Penetration**
Description: The attacker attempts to gain unauthorized access to the system; this is emulated through bruteforce login attempts.	
Log deletion (DEL)	**Control**
Description: The attacker attempts to cover his/her traces by deleting logs; this is emulated by removing the log of *homer*.	
Registration (REG)	**Embedding/Attack relay**
Description: The attacker attempts ensuring that he/she can retain control of the system even if him/her actions are discovered; this is emulated by registering multiple accounts in the system - a new account is registered every 5 s.	
Denial of service (DoS)	**Attack relay**
Description: The attacker overloads the system in order to slow down the performance; the system is exercised with 10 concurrent clients, with each client running an instance of Clearwater-live-test.	
Kill container (KILL)	**Attack relay**
Description: The attacker attempts to block the system operation by killing some services; this is emulated through the kill of the *bono* container, which is the anchor point for the client's connection to the Clearwater system.	

```
1   *** NORMAL log line from "homer.txt" ***
2  05-10-2018 07:36:54.357 UTC INFO base.py:259: Received request from
3   localhost - PUT http://http_homer/users/sip3A650555003440example.com/
4   simservs.xml
5
6   *** ANOMALOUS log line from "bono.txt" ***
7  6-11-2018 14:18:32.644 UTC Error bono.cpp:1337: Route header flow
8   identifier failed to correlate
9
10  *** ANOMALOUS log line from "sprout.txt" ***
11 10-10-2018 15:44:26.725 UTC Warning httpclient.cpp:620: reg-data failed
12  at server 172.18.0.6 : Timeout was reached (28) : fatal
```

Fig. 4. Example of normal and anomalous log lines.

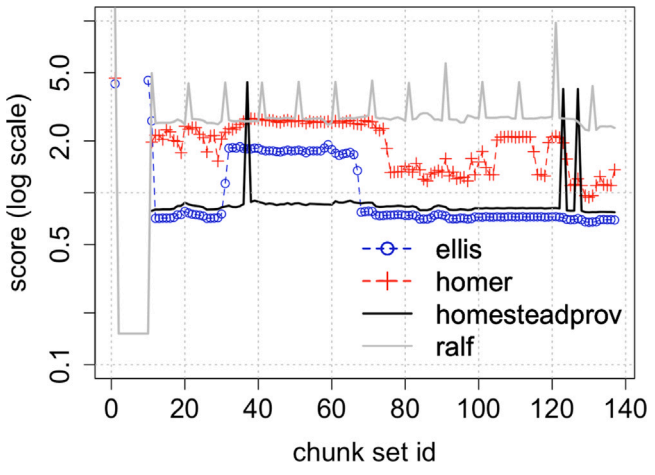


Fig. 5. Scores of four logs over around one hour of normal operations (a chunk corresponds to 30 s).

is accomplished. *These settings are not meant to be exhaustive for all "real-life" anomalies that may occur in practice*. However, they aim to resemble the stages of potential security attacks and to show how system misuses may generate suspicious log entries, which can be caught through the proposal. While we use them in some of the experiments hereinafter, the key findings of our paper are supported by a model consisting of log variants, as follows.

5.2. Log variants

Log variants are perturbations of normal logs. The idea of using such variants arises from the consideration that it is hard to foresee how **real-life events** will practically reflect across the logs. Overall, the inherent *uncertainty* of real operational logs in reporting anomalies is a threat for obtaining accurate detectors. In fact, *poor* training logs (i.e., logs that fail in covering an exhaustive number of real operating conditions) will likely return an ineffective detector. As such, we are aware that the *sole* settings shown in Table 2 – although useful for some of the considerations later on in this study – cannot provide a comprehensive picture on the accuracy that can be expected with our approach. Please note that the lack of data points from real-life anomalies is a challenge in many domains. For example, in Sharma et al. (2011) is shown that many security incident classes, e.g., *spam* and *infected hosts*, had very few occurrences over a very long time frame.

We propose a **systematic procedure** to *model* the effects that anomalies may have in the logs through *log variants*. This is done by means of an experimental design and controllable factors (Jain, 1991). Let us introduce the procedure for generating *one* variant, beforehand; we then discuss the generation of the *dataset* of variants for our experiments.

A log variant is generated by randomly selecting a normal *chunk set*, at first. Within the *chunk set*, we then randomly select a number of chunks n ($1 \leq n \leq N$) to perturb. For each selected chunk, perturbation consists in: (i) modifying the *size*, i.e., the number of lines, of the chunk by either clearing the chunk, or leaving the size unchanged, or doubling it by duplicating each line of the chunk; (ii) appending a certain *amount* (i.e., 1%, 5% or 10% of the size of the chunk) of anomalous lines to the chunk. An anomalous line is a sequence of *unknown* terms, i.e., terms

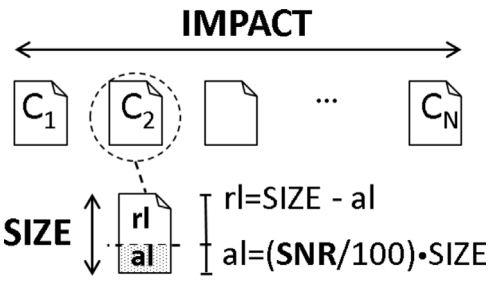


Fig. 6. Representation of the factors.

Table 3
Factors and levels used for log variants.

Factor	Level		
	LOW	*MEDIUM*	*HIGH*
IMPACT	1	3	6
SIZE	×0 (chunk cleared)	×1 (size unchanged)	×2 (size doubled)
SNR	1%	5%	10%

that never occur in the database of normal logs; each unknown term is obtained by concatenating a random string to a given normal term.

More formally, let C_i with $1 \leq i \leq$ N denote the chunks in a *chunk set*. The factors accounted for generating the variants are indicated in the following and depicted in Fig. 6:

- **impact** ($IMPACT$): number of chunks – out of total N – that are targeted by the perturbation;
- **size** ($SIZE$): number of lines of the chunk;
- **signal-to-noise-ratio** (SNR): percentage of *anomalous lines* (al) with respect to the size of the chunk.

Given the value of $SIZE$ and SNR, the number of anomalous lines is al = $((SIZE/100) \cdot SNR)$, while the number of regular lines (rl) is $(SIZE - al)$ as shown in Fig. 6. As usually done in empirical assessments, we categorize the factors through a smaller number of *levels* (Jain, 1991), e.g., *LOW*, *MEDIUM*, *HIGH*. Table 3 summarizes the value of the levels in our study. We generate log variants for all the combinations of values in $IMPACT \times SIZE \times SNR$; the only exception occurs with $SIZE = \times 0$, where SNR does not apply because the variant is obtained by clearing a number of chunks equal to *IMPACT*. As such, we obtain total 21 rather than 27 combinations of the levels. Given that random choices are performed in constructing the variants, we generate around 120 variants (which account for one hour of anomalous operations) for each combination of the levels. It is worth to note that the proposed procedure does not aim to produce log variants that resemble the logs generated during anomalies; rather, it aims to mimic the effects that anomalies may have on the logs.

5.3. Available datasets

We collect total 5 datasets of logs – and thus scores – which are summarized in Table 4. Each dataset is assigned an *ID* that will be used through the rest of the paper for the sake of clarity. *D1* and *D2* contain `NORMAL` vectors of scores, which are collected by means of independent runs of *Clearwater-live-test*; on the other hand, *D3* contains `ANOMALOUS` scores obtained by emulating the settings in Table 2, which are emulated in independent experiments. Noteworthy, scores in *D1*, *D2* and *D3* are computed from logs obtained by direct observations of Clearwater – i.e., *real*, in the "Nature" column of Table 4 – while *D4* and *D5* contain vectors of `ANOMALOUS` scores obtained with log variants. Therefore, they are *synthetic*. We made a comprehensive sample of logs collected during normal and anomalous events publicly available.[9]

Table 4
Summary of the datasets from Clearwater.

Dataset ID	Events type	Nature	# Chunk sets
D1	`NORMAL`	Real	2757
D2	`NORMAL`	Real	127
D3	`ANOMALOUS`	Real	81
D4	`ANOMALOUS`	Synthetic	2632
D5	`ANOMALOUS`	Synthetic	129

Table 5
Metrics obtained with the test set (FNN denotes a feedforward neural network).

	Precision	Recall	F-measure	Accuracy
AdaBoost.M1	0.974	0.973	0.973	97.3%
Random tree	0.973	0.973	0.973	97.3%
Autoencoder	0.969	0.953	0.961	96.1%
Decision tree	0.960	0.957	0.957	95.7%
FNN (4 hidden layers)	0.947	0.961	0.954	95.3%
FNN (1 hidden layer)	0.935	0.934	0.934	93.4%
Bayesian network	0.915	0.906	0.906	90.6%

6. Micro2vec: Validation and practical implications

Numeric representations produced by Micro2vec can be conveniently used to infer models to discriminate normal from anomalous events. Models can be obtained through well-consolidated machine and deep learning techniques by the data mining community. It is worth noting that Micro2vec *does not* mandate a specific mining technique to handle the representations: in order to demonstrate the validity of our approach we use a variety of techniques ranging from decision trees to deep neural networks. For each technique assessed – given the datasets in Table 4 – $D1 \cup D4$ serves as **training** set, while $D2 \cup D5$ is the **test** set; a subset of 20% data points of the training set serves as **validation** set, which is used to select and tune the hyperparameters of the techniques in hand. For the assessment of the techniques we refrain from using *D3* (i.e., the dataset of real anomalies) to avoid any bias that could be caused by the settings in Table 2.

The results are presented in Table 5; the leftmost column shows the techniques assessed. We compute the typical metrics of *precision*, *recall*, *F-measure* and *accuracy* (Makhoul et al., 1999) to quantify the effectiveness of the models at discriminating normal from anomalous events. Results are based on WEKA[10] and Keras[11] implementations of the autoencoder and the multilayer feedforward neural network (FNN). It can be noted that all the techniques achieve remarkable classification figures, which means the numeric representations generated by Micro2vec can be used in conjunction with different techniques. Most notably, the results indicate that deep learning models can be used to handle the scores. For example, in case of the autoencoder we use a semi-supervised approach to learn a model of the normal data points, beforehand; given the points of the test set, anomaly detection is pursued by measuring the distance – also known as reconstruction error – of the points with respect to the model. The autoencoder achieves 0.969 precision and 0.953 recall, which are among the top performing. As for the FNN with 4 hidden layers, we obtain 0.947 precision and 0.961 recall. As a further remark, in order to check if our data are affected by class imbalance issues, we used the *AdaBoost.M1* (Freund and Schapire, 1997) classifier. It is important to note that AdaBoost.M1 is not a mean to check if a dataset is imbalanced or not, but is usually adopted to classify imbalanced datasets (Yuan and Abouelenien, 2015).

[9] www.dessert.unina.it/JNCA2022EventLogs.zip.
[10] https://www.cs.waikato.ac.nz/ml/weka/.
[11] https://keras.io/.

Table 6
Results of ANOVA (SST = 0.0107; R-square = 0.97; imp = importance).

	SS	imp	F-stat	p-value
IMPACT	0.0098	91.6%	86.85	**0.0005**
SIZE	≈0	0.0%	0.36	0.5822
SNR	0.0003	2.8%	2.99	0.1622
IMPACT·SIZE	≈0	0.0%	0.17	0.8470
IMPACT·SNR	0.0003	2.8%	1.45	0.3644
SIZE·SNR	≈0	0.0%	0.24	0.7940
Total	0.0104	97.2%	–	–

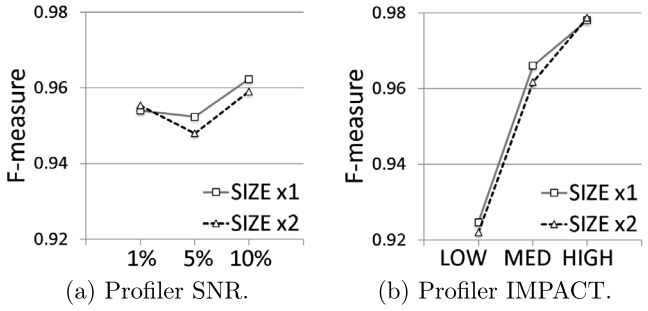
Therefore, since AdaBoost.M1 performance is comparable with the best performing classifier, i.e., random tree, it can be reasonably stated that our data are not affected by the class imbalance issue.

Without loss of generality, we will focus on the **decision tree**: (i) it performs reasonably well in our context, as shown in Table 5, and (ii) much more important, it produces an output that is compact, comprehensible and useful to practitioners, which is crucial to the objective of automating the generation of detection rules that can complement existing SIEM tools. Even if decision trees can be affected by the instability issue (Li and Belford, 2002), this does not happen in our settings as we achieve similar results with both original and perturbed datasets, i.e., considering log variants as it will be shown later on in the paper. We aim to assess how the availability of measurements from different logs is beneficial to the detection of anomalies by means of the decision tree. The key advantage in having log variants obtained under controllable factors consists in the possibility to conduct an **ANalysis Of VAriance (ANOVA)** (Jain, 1991) to gain insights into the statistical significance of factors – and their interactions – on a *response variable*. In this study we use the F-measure as **response variable** because precision and recall of the classifiers are reasonably close.

ANOVA is done as follows. We split the variants into *disjoint groups* based on the levels of the factors that were used to obtain them[12]; for each group of variants we compute the F-measure obtained by the decision tree at classifying the variants belonging to a certain group with respect to the normal logs. Results are summarized in Table 6. The total variability of the F-measure across the groups of variants, i.e., *sum of squares total* (SST) in ANOVA, is 0.0107. For each factor and interaction, Table 6 shows the *sum of squares* (SS), i.e., the portion of SST explained by the factor/interaction. We observe that *IMPACT* is the most important: it explains 91.6% of the variability and it is statistically significant (i.e., *p-value* < 0.05).

Practical implications. Based on the results of ANOVA, the detector is sensitive to small percentages of anomalous lines. In fact, having more than 1% anomalous lines – e.g., 5%, 10% *SNR* – does not necessarily reflect into higher F-measure, such as shown in Fig. 7(a). This means that the detector does not depend on the verbosity of anomalous events in the logs, which is strongly desirable in practice. More important, the observation that *IMPACT* explains most of the variability, indicates that the detector is able to capitalize on the availability of multiple measurements from different microservices; we also note that, differently from the *SNR*, F-measure improves as *IMPACT* increases (Fig. 7(b)). This is another important outcome, given the distributed nature of microservice systems and related security implications. Having *diverse* points of observation, i.e., log sources, is beneficial for automated anomaly detection and reflects into higher F-measure.

[12] We exclude from this analysis the variants obtained with *SIZE* = ×0 because the *SNR* does not apply, such as explained in Section 5.1.



(a) Profiler SNR. (b) Profiler IMPACT.

Fig. 7. Profiler for *SNR* and *IMPACT* by *SIZE*.

Table 7
Usage of the datasets.

Section	Training set	Test set	Notes
7.1	*Not applicable*	*D2* ∪ *D3*	*K*-fold cross validation
7.2	*D1*	*D2* ∪ *D3*	One-class model
7.3	*D1* ∪ *D4*	*D2* ∪ *D3*	Variant-*based* model

7. Results

We present the results concerning anomaly detection. Our aim is to investigate both (i) the relationships across microservices' logs for the system in hand, and (ii) the effectiveness of our variant-*based* approach as a workaround to learn a model in those datasets that lack real-life anomalous data points, which is increasingly common in practice. Our analysis is supplemented by a comparison with one-class classifiers, which can be conveniently used in those settings where only *one* class can be learned accurately.

For the sake of clarity, Table 7 indicates how the datasets described in Section 5.3 have been used to obtain the results presented through the following subsections. For all the cases, the **test set** consists of normal/anomalous scores computed from the logs obtained by means of direct collections from Clearwater, i.e., *D2* ∪ *D3*.

7.1. Relationships across the logs

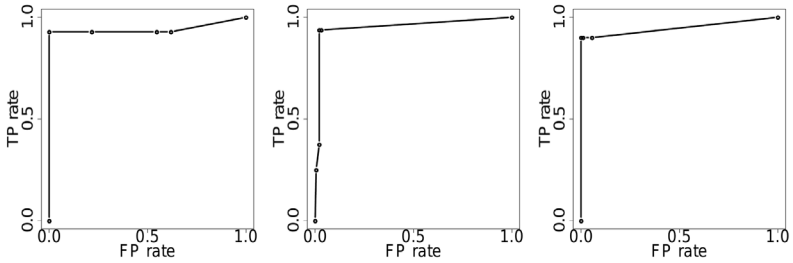
Our proposal develops around the intuition that relationships across microservices' logs can be leveraged for detection purposes. In this experiment we use a *decision tree* and a *K*-fold cross validation approach to model the scores from both normal and anomalous events, i.e., *D2* ∪ *D3*. In this case *D2* ∪ *D3* is split into 10 disjoint folds F (with $1 \leq F \leq 10$). For each fold f, we train a decision tree with 9 folds ($f \neq F$) and test it with the held-out fold (f==F); thus, "training set", is not applicable – as stated in Table 7 – because training/test are inherently intertwined. In this first experiment we use a *K*-fold approach in order to leverage *all* the vectors in *D2* ∪ *D3*: in fact, we aim to compute an upper bound for the recall and precision metrics that will serve as reference to assess the one-class and variant-*based* models.

Fig. 8 shows the relationships by means of the obtained decision tree (please refer to Table 2 for the acronyms of the events); bold characters highlight the *path* that catches the vast majority of normal vectors of scores (*line 12*). Regarding anomalous events, we found out that around 82% vectors from the AUTH setting reflect into abnormal activity by *ellis* (*line 17* in Fig. 8); similarly, 90% vectors obtained during the DoS setting are characterized by abnormal activity in *homesteadaccess* (*line 18*). The reader might further inspect Fig. 8 to see other interesting correlations.

These correlations across the logs are hard to be caught by human experts, who are typically responsible for manually crafting security compromise indicators, e.g., the *correlation rules* of up-to-date SIEM tools (Bhatt et al., 2014). This practice closely resembles traditional business intelligence, where humans must know what they wish to

```
homer <= 0
|   chronos <= 0.054403: KILL (90%)
|   chronos > 0.054403: DEL (93%)
homer > 0
|   homesteadaccess <= 0.836042
|   |   ellis <= 4.256036
|   |   |   homesteadprov <= 0.918568
|   |   |   |   sproutaccess <= 2.480235
|   |   |   |   |   homestead <= 7.301185
|   |   |   |   |   |   bono <= 0.589198: AUTH (18%)
|   |   |   |   |   |   bono > 0.589198: NORM (1%)
|   |   |   |   |   homestead > 7.301185: NORM (99%)
|   |   |   |   sproutaccess > 2.480235
|   |   |   |   |   bono <= 0.649348: DoS (6%)
|   |   |   |   |   bono > 0.649348: REG (25%)
|   |   |   homesteadprov > 0.918568: REG (75%)
|   |   ellis > 4.256036: AUTH (82%)
|   homesteadaccess > 0.836042: DoS (90%)
```

Fig. 8. Decision tree model and relationships among the scores of normal (NORM) and anomalous events.



(a) DEL (b) REG (c) KILL

Fig. 9. Worst three ROC curves for decision tree.

Table 8
Metrics of the classifier trained with real logs.

Precision	Recall	F-measure	ROC area	Accuracy
0.948	0.942	0.944	0.965	94.2%
Recall/ROC area by anomaly				
DoS	KILL	DEL	REG	AUTH
0.90/0.98	0.90/0.95	0.93/0.94	0.94/0.95	0.82/0.99

look for, *beforehand*. Differently, our work puts forth the concept of data discovery to complement the task of finding effective compromise detection rules.

We note that both normal and anomalous events are characterized by a precise signature *in the relationships of the scores*. Table 8 provides overall precision and recall of the decision tree, both higher than 0.94. Moreover, the bottom row of the table also shows the breakdown of the recall and ROC area by anomaly, which are in the range 0.82 (AUTH) - 0.94 (REG) and 0.94 (DEL) - 0.99 (AUTH), respectively—thus reasonably high. The good results are also highlighted by the ROC curves depicted in Fig. 9, which are reasonably near to the optimal curve (as suggested by the ROC area values in Table 8), albeit they are the worst three obtained ones. These figures are used as a reference in assessing the one-class and variant-*based* models.

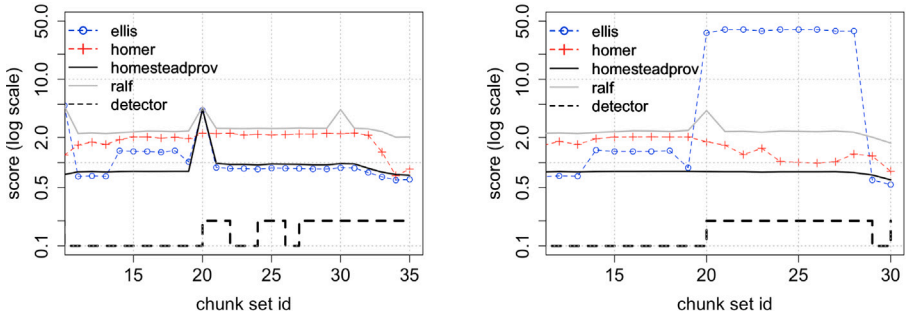
7.2. Detection through one-class classification

One-class classifiers represent a viable solution to cope with the lack of real anomalies. One-class approaches leverage the assumption that only one class – named *target* class – is well characterized in a training set. Any arbitrary data point that does not resemble the only class learned during training is deemed as *outlier*. We explore two one-class classification approaches: (i) the approach proposed in Hempstalk et al. (2008) and (ii) the one-class Support Vector Machine (SVM) (Schölkopf et al., 2000) implemented in Chang and Lin (2011). Both approaches are reference work in one-class classification studies as they have been largely used for comparison purposes (Anderka et al., 2012; Barbhuiya et al., 2018).

Table 9
Metrics of the one-class classifiers.

	Precision	Recall	F-measure	Accuracy
Approach in Hempstalk et al. (2008)				
Random tree	0.553	0.953	0.699	54.8%
Decision tree	0.552	1.000	0.711	55.2%
Bayesian net.	0.897	0.890	0.893	88.3%
Multilayer perc.	0.724	0.929	0.814	76.5%
AdaBoost.M1	0.552	0.949	0.696	64.9%
Approach in Schölkopf et al. (2000)				
One-class SVM	0.750	0.520	0.614	63.9%

In this study the scores computed from *normal* logs are the **target** class: only vectors of scores representing normal events are used to train the one-class classifiers. Anomalies are expected to be classified as **outlier**. We use the dataset *D1* to train the classifiers with real normal scores, and test it against $D2 \cup D3$, as shown in Table 7. Table 9 shows the obtained results. The approach in Hempstalk et al. (2008), after having transformed the problem into a binary classification, can be used in conjunction with any existing classifier: Table 9 presents the results obtained with the most promising we found after sensitivity analyses. Results indicate that the *Bayesian network* reaches the maximum overall metrics, with an F-measure of 0.893 and an accuracy of 88.3%. It can be noted that, in spite of the quite high recall for almost all the classifiers, only the Bayesian one obtains acceptable precision and recall, which are however lower if compared to the corresponding



(a) Registration event. (b) Authentication event.

Fig. 10. Scores of four Clearwater's logs and detector's output in two anomalous events.

Table 10
Metrics of Hempstalk et al. (2008) with the Bayesian network.

	Precision	Recall	F-measure	Accuracy
	0.897	0.890	0.893	88.3%
Recall by anomaly				
DoS	KILL	DEL	REG	AUTH
0.87	1.00	1.00	0.62	1.00

Table 11
Metrics of the classifier trained with log variants.

Precision	Recall	F-measure	ROC area	Accuracy
0.913	0.901	0.907	0.917	90.7%
Recall by anomaly				
DoS	KILL	DEL	REG	AUTH
0.97	0.90	0.93	0.75	0.91

reference metrics in Table 8. This is explained by the breakdown of the recall over the anomalies collected by direct observations of Clearwater in Table 10. The one-class classifier achieves a recall of 1.00 for KILL, DEL and AUTH; however, it drops sharply to 0.87 and 0.62 for DoS and REG, respectively, which means that one-class classification is rather ineffective in some anomalous scenarios.

7.3. Detection by means of the log variants

We aim to address the following question: *can the variants be used to train a model for detecting **real** anomalies?* This is strongly relevant because, as stated above, a training database of real logs will never cover all the anomalies that can occur in production; on the contrary, variants emulate a wide spectrum of settings *by experimental design*, although they are not directly related to anomalies. To explore this proposition we train a decision tree with $D1 \cup D4$ and test it on $D2 \cup D3$, as shown in Table 7, i.e., we attempt to classify scores computed from **real** normal/anomalous logs with a model learned from variants.

Results are shown in Table 11. Both recall and precision are higher than 0.90, which is reasonably satisfactory. In Fig. 10 we plot the output of the detector during the progression of two anomalous events, i.e., *registration* and *authentication*, respectively. The detector is superimposed as a dotted line to the scores of four microservices' logs, where the transition *low-high* denotes that the anomalous event is detected; in both cases the anomalous event starts around the *chunk set* id 20. For example, in the registration setting the detector takes few cycles before it stabilizes at *high*.

With respect to Table 8 – i.e., training done with real logs – we note a loss in recall and precision, which drop roughly by 0.05. Such a drop is expected because, differently from recent contributions in this area, such as Cao et al. (2016) and Xie et al. (2018), our variants *are not* generated by perturbing the anomalous events. This is done to avoid biasing the results towards a specific anomalous setting and to investigate a more general research direction, which does not rely on the availability of labeled data for generating the variants. Interestingly, the loss is caused by only one anomalous event type (i.e., REG, as it can be noted by comparing the bottom rows of Tables 8 and 11), which means that – based on the findings of our study – variants obtained with **no knowledge** of real anomalous events are potentially useful to support the detection. More importantly, the variant-*based* model improves over one-class classification, whose results were in Table 10.

We closely look into the models in order to interpret the improvement obtained with variants over the one-class approach. Fig. 11 shows the model obtained by means of Hempstalk et al. (2008) in conjunction with a decision tree (we discuss the decision tree *in lieu of* the Bayesian network for the sake of visual comparison); similarly, Fig. 12 shows the path of the variant-*based* model catching the majority of normal vectors of scores (*line 18*). Fig. 11 indicates that the model obtained with the one-class approach leverages only 2 out of 13 log sources, i.e., logs of *cassandra* and *ralfaccess*. Please note that negative scores in Fig. 11 result from the internal mathematical transformations of Hempstalk et al. (2008), since – by construction – our log.entropy is always ≥ 0. On the other hand, the model obtained with log variants capitalizes on many more log sources, as shown in Fig. 12, which provides finer-grain leaves to discriminate normal from anomalous events. Interestingly, most of the logs used for the decision are the same as in Fig. 8, which were obtained from *real* logs.

8. Application to the ATC case study

In order to assess the proposal with another case study, we apply it to a real-world critical information system from the Air Traffic Control (ATC) domain. The system installation has been made available by a top-leading industrial company in electronic and information technologies for defense and aerospace, in the context of an academia-industry project. The system encompasses a number of nodes running distributed applications, which cooperate to implement ATC-related capabilities, such as aircraft trajectory monitoring and collision prevention/detection. The cooperation is enabled by means of message passing between applications, according to the publish–subscribe paradigm, widely adopted in microservice architectures. More important, the system encompasses many heterogeneous log sources, with applications generating multiple logs per node, similarly to the Clearwater case. Overall, 16 log files sources are available, encompassing legacy logs implemented by means of a variety of custom formats; the collected log files are listed in Table 12 by originating node.

As done for Clearwater, we collect logs encompassing both *normal* and *anomalous* events. **Normal logs** are collected by exercising the system with test suites, which are used by the industry provider to emulate the system usage by real ATC operators. The workload contains several operations, ranging from Flight Data Plan creation/update, which contains the data of a given flight (such as expected route,

```
1 cassandra <= 0
2 |    cassandra <= -0.013819: outlier (41%)
3 |    cassandra > -0.013819: NORM (96%)
4 cassandra > 0
5 |    ralfaccess <= 0.477326: NORM (4%)
6 |    ralfaccess > 0.477326: outlier (59%)
```

Fig. 11. Model obtained with Hempstalk et al. (2008) and the decision tree.

```
1 omitted
2 homestead <= 8.50964
3 |    homestead <= 0: ANOM (0.7%)
4 |    homestead > 0
5 |    |    ralf <= 0: ANOM (0.8%)
6 |    |    ralf > 0
7 |    |    |    homer <= 0: ANOM (0.6%)
8 |    |    |    homer > 0
9 |    |    |    |    sproutaccess <= 0: ANOM (0.5%)
10 |    |    |    |    sproutaccess > 0
11 |    |    |    |    |    homesteadprov <= 0: ANOM (0.4%)
12 |    |    |    |    |    homesteadprov > 0
13 |    |    |    |    |    |    ralfaccess <= 0: ANOM (0.3%)
14 |    |    |    |    |    |    ralfaccess > 0
15 |    |    |    |    |    |    |    homesteadaccess<=0:ANOM(0.3%)
16 |    |    |    |    |    |    |    homesteadaccess > 0
17 |    |    |    |    |    |    |    |    ellis <= 0: ANOM (0.3%)
18 |    |    |    |    |    |    |    |    ellis > 0: NORM (94.4%)
19 homestead > 8.50964: ANOM (1.7%)
20 omitted
```

Fig. 12. Extract of the decision tree model obtained from variants and relationships among the scores of normal (NORM) and anomalous events (ANOM).

Table 12
Log files by ATC system node.

Node	Name of the log file
D02	D02PAN, D02msg
DB1	DB1PAN, DB1msg
FP1	FP1PAN, FP1NTN, FP1LNR, FP1AFS, FP1msg
MN1	MN1PAN, MN1MNA, MN1msg
MS1	MS1PAN, MS1msg
SFN	SFNPAN, SFNmsg

Table 13
Metrics of the decision tree trained with real logs from the ATC case study.

Precision	Recall	F-measure	ROC area	Accuracy
0.959	0.959	0.959	0.980	95.9%
Recall/ROC area by anomaly				
DoS	KILL	DEL	RER	AUTH
0.90/0.96	0.99/0.99	0.97/0.99	0.96/0.98	1.00/1.00

trajectory), to Flight re-routing, which allows to modify the trajectory of a flight. Normal logs are also used to build the replacements base[13]
Anomalous events are collected by emulating different settings that resemble operations occurring under different attack phases, as done for Clearwater. We emulate 5 settings: (i) *Denial of Service* (DoS), where the attacker starts misusing the system creating a new flight every second to slow down the system response: (ii) *Kill service* (KILL), where the attacker attempts to block the system operation by killing the database service; (iii) *Log Deletion* (DEL), where some application logs are deleted by an attacker with the aim to cover his/her traces, (iv) *Flight re-routing* (RER), where the attacker modifies the data in the system by re-routing a flight every five seconds towards the same destination; (v) *Brute-force authentication* (AUTH), where an attacker attempts to gain unauthorized access to the system through brute-force login attempts at the D02 node, i.e., the front-end of the ATC system. We reproduce these scenarios during the execution of the normal workload of the system, so that anomalous events are overlapped with regular operations as it would occur in production.

The collected logs are periodically sampled to generate the chunk sets; the sampling period is set to $T = 10$ s, while the sample size is set to $(M\text{-}1) = 120$. We collect total 3 datasets, i.e., (i) *D1-ATC* and (ii) *D2-ATC*, containing NORMAL scores, and (iii) *D3-ATC*, containing ANOMALOUS scores. As done for the Clearwater case study, we present the results concerning the detection of anomalous events with the ATC system involving both *multi-class* and *one-class* classifiers.

Multi-class classification. We use a *decision tree*[14] and a K-fold cross validation approach (with $K = 10$) to model the scores from both normal and anomalous events contained in the *D2-ATC* and *D3-ATC*, respectively. As seen for the Clearwater case study, we note that both normal and anomalous events are characterized by a *precise signature in the relationships* of the scores. Fig. 13 shows the relationships by means of the obtained decision tree. It can be noted that around 93% vectors from the DEL setting reflect into abnormal activity by the *D02PAN* log (*line 17*), which also involves other log files, such as *D02msg*, *FP1PAN*, *MN1MNA*, *SFNPAN*, *FP1AFS*, as indicated by the bold characters. Similarly, we found out that around 99% vectors obtained during the KILL setting are characterized by abnormal activity in *FP1PAN* log files (*line 2*). It is important to note that in this case the abnormal activity is represented by the absence of events in the log, which is coherent with the emulated KILL scenario. Table 13 provides overall precision and recall of the decision tree, both higher than 0.95, as well as the overall ROC area value, i.e., 0.980. The high

[13] Patterns included in the replacements base have been made publicly available -www.dessert.unina.it/JNCA2022ATCLogs-Regex.zip.

[14] We use the same classifier of the first case study for comparison purposes.

```
1 D02msg <= 3.244645
2 |   FP1PAN <= 0: KILL (99%)
3 |   FP1PAN > 0
4 |   |   MN1MNA <= 0
5 |   |   |   SFNPAN <= 0.240435
6 |   |   |   |   MS1PAN <= 0.216978
7 |   |   |   |   |   D02PAN <= 0.595386: DoS (1%)
8 |   |   |   |   |   D02PAN > 0.595386: NORM (4%)
9 |   |   |   |   MS1PAN > 0.216978: DEL (1%)
10 |   |   |   SFNPAN > 0.240435
11 |   |   |   |   FP1AFS <= 1.344
12 |   |   |   |   |   SFNPAN <= 1.302572
13 |   |   |   |   |   |   FP1PAN <= 1.380284: DEL (7%)
14 |   |   |   |   |   |   FP1PAN > 1.380284: KILL (1%)
15 |   |   |   |   |   SFNPAN > 1.302572: NORM (3%)
16 |   |   |   |   FP1AFS > 1.344
17 |   |   |   |   |   D02PAN <= 11.853783: DEL (93%)
18 |   |   |   |   |   D02PAN > 11.853783
19 |   |   |   |   |   |   D02PAN <= 12.828658: DEL (1%)
20 |   |   |   |   |   |   D02PAN > 12.828658: DoS (1%)
21 omitted
```

Fig. 13. Extract of the decision tree model and relationships among the scores of normal (NORM) and anomalous events from the ATC system.

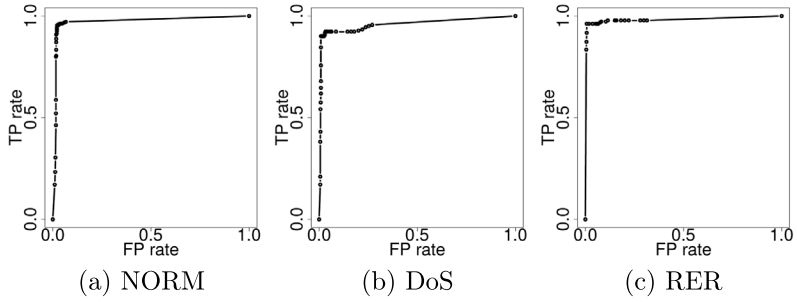


Fig. 14. Worst three ROC curves for decision tree—ATC.

Table 14
Metrics of one-class SVM—ATC.

	Precision	Recall	F-measure	Accuracy
	0.925	0.408	0.566	85.8%
Recall by anomaly				
DoS	KILL	DEL	RER	AUTH
0.99	0.93	0.90	1.00	1.00

ROC area value is explained by the ROC curves depicted in Fig. 14, which are reasonably near to the optimal curve albeit they are the worst three obtained ones, as suggested by the ROC area values in Table 13. Interestingly, achieved results are in line with the ones obtained in the Clearwater case. Let us add that also for this case study we evaluate the AdaBoost.M1 performance, which performs similarly to the decision tree, confirming that also the ATC dataset is not affected by the class imbalance issue.

One-class classification. In order to apply one-class classification approaches, the scores computed from normal logs are considered as the *target* class and they are used for training, as done in the Clearwater case study. We use the dataset *D1-ATC* to train each one-class classifier used for Clearwater with real normal scores, and test it against *D2-ATC* ∪ *D3-ATC*. Results indicate that – with respect to our data – the *one-class SVM* classifier reaches the maximum overall metrics, with an F-measure of 0.566 and an accuracy of 85.8%. Moreover, differently from the other classifiers, the one-class SVM classifier is the only one exhibiting an acceptable precision. Table 14 summarizes the metrics of the one-class SVM classifier. Overall, it achieves precision and recall of 0.925 and 0.408, which are lower if compared to the corresponding reference metrics shown in Table 13. This is explained by the breakdown of the recall over anomaly in Table 14.

9. Discussion and threats to validity

The detection figures that we obtain by mining the numeric representations produced by Micro2vec with commonly-used machine and deep learning algorithms are up to 0.97 and 0.96 recall, precision and F-measure, for the microservices and ATC case study, respectively. The performance achieved is inline with existing work in the area. For example, the Authors in Meng et al. (2019) propose a method, LogAnomaly, based on *template2Vec*, i.e., a template representation method to extract semantic and syntax information from log templates inspired by word embedding. The F-measure of our technique is similar to LogAnomaly applied to the BG/L log benchmark, i.e., 0.96. As for other approaches that imply sequential embedding, the work Guo et al. (2021) applies BERT, i.e., the Bidirectional Encoder Representations from Transformers developed by Google, and achieves 0.92 recall and 0.89 precision with the BG/L log: both the figures are lower than Micro2vec. As for other related approaches using BERT, it is worth mentioning (Hirakawa et al., 2020), which achieves 0.89 F-measure with BG/L, again lower than Micro2vec. SwissLog (Li et al., 2020), which is applied in conjunction with Long short-term memory (LSTM), Bidirectional LSTM (BiLSTM) and Attention-based BiLSTM, achieves 0.95, 0.96 and 0.99 F-measure in BG/L, respectively: the F-measure of Micro2vec is in line with the first two SwissLog variants, although lower than the latter. It should be noted that SwissLog is conceived for a specific fault model of log sequence order changes and log time interval changes: most notably, Micro2vec is not constrained by specific types

of faults. Finally, the recall and precision of Micro2vec are in the range of AutoLog (Catillo et al., 2022), which achieves a value of the recall between 0.96 and 0.99, and precision within 0.93 and 0.98 applied to the logs of four systems.

As for any measurement study, there may be concerns regarding validity and generalizability of the proposal and results. We briefly discuss them, based on the aspects in Wohlin et al. (2000).

Construct validity. Our work builds on the intuition that relationships across microservices' logs can be leveraged for detection purposes. This is pursued by instantiating our experiments in the context of two systems, i.e., Clearwater IMS, which well represents an important category of microservice systems, and the ATC critical information system, which leverages microservices-related communication technologies. We devoted special attention to exercise both systems according to representative operations. Both systems are: (i) deployed according to the settings provided by the developers, (ii) fed with realistic input data, and (iii) exercised with the test suite used by developers to emulate their nominal usage. Regarding the anomalies, we rely on events that resemble the security attack phases in Ruiu (1999) and Sharma et al. (2011). The considered events are not meant to be exhaustive for all anomalies that may occur in practice. However, although they cannot provide a comprehensive picture on the accuracy that can be expected with our approach, the considered anomalies allow pointing out the potential of the approach. Finally, we rely on well-funded log analysis methods for extracting quantitative metrics from logs; findings have been inferred with an approach encompassing design of experiments and assessment of the measurement error.

Internal and conclusion validity. We use a mixture of datasets consisting of real logs and variants to provide evidence of the actual relationships across the metrics. Experiments include multi-class and one-class classification, and classification done on the top of log variants. We assess different classifiers and generate the variants under different configurations of the key factors. Metrics of precision, recall and F-measure are computed with both real logs and variants with the aim of avoiding any favorable setting to our proposal. Overall, this mitigates internal validity threats and provides a reasonable level of confidence on the conclusions.

External validity. Results observed on two case studies are not statistically generalizable. However, the reported findings, which are strongly supported by data, are still useful to get an overall understanding on the potential of the proposal. Our proposal should be easily applicable to other similar systems. We require no microservices/applications modifications. Practitioners are not expected to spend any effort in using our proposal since it is inherently non intrusive; moreover, we do not embed knowledge of the application. The details provided should reasonably support the replication of our study by other researchers. Noteworthy, we made a comprehensive sample of logs/scores publicly available for research purposes.

10. Conclusion

This paper presented our study on mining logs for anomaly detection in the context of systems leveraging microservices-related technologies. We propose Micro2vec, a novel approach to mine numeric representations of computer logs, which allow inferring "actionable" relationships for anomaly detection. The approach embeds no application knowledge, makes no assumptions on the format/semantics of underlying logs, and requires no catalogues of anomalies symptoms. We conduct our experiments with a Clearwater IMS installation and with a real-world ATC critical information system.

We believe that the obtained findings should be extremely useful to practitioners and to drive future research directions. Results indicate that analyzing metrics inferred by different logs facilitates the detection of anomalies, which are characterized by signature involving multiple logs; it also allows inferring explicable detection rules that are hard to be caught by human experts. In addition, log variants obtained from normal logs can support detecting real anomalies, and they improve over one-class classifiers.

In the future we aim to extend our experiments to production systems and to different application domains. In fact, the proposal is not tailored for a specific mining approach or system. In addition, future work will also be devoted to inspect other methods for log variants. Further, we will investigate the actions that malicious users might take to evade detection as well as the use of retraining mechanisms to update the normal model when the system changes. Finally, the impact of both log quality and parsing on the proposal will be also assessed, since they can potentially affect the effectiveness of the scoring task and then of the anomaly detection.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data from one dataset is available via the link provided in the manuscript.

Acknowledgments

This work has been partially supported from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 871342.

References

Ahuja, N., Singal, G., Mukhopadhyay, D., Kumar, N., 2021. Automated DDOS attack detection in software defined networking. J. Netw. Comput. Appl. 187, 103108.

Anderka, M., Stein, B., Lipka, N., 2012. Predicting quality flaws in user-generated content: The case of wikipedia. In: Proc. of the 35th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval. ACM, New York, NY, USA, pp. 981–990.

Bakar, N., Belaton, B., Samsudin, A., 2005. False positives reduction via intrusion alert quality framework. In: 2005 13th IEEE Intl. Conference on Networks Jointly Held with the 2005 IEEE 7th Malaysia Intl. Conference on Communications. p. 6.

Barbhuiya, S., Papazachos, Z., Kilpatrick, P., Nikolopoulos, D.S., 2018. RADS: Real-time anomaly detection system for cloud data centres. CoRR, abs/1811.04481 arXiv:1811.04481.

Bass, L., Weber, I., Zhu, L., 2015. DevOps: A Software Architect's Perspective. Addison-Wesley Professional.

Behal, S., Kumar, K., Sachdeva, M., 2018. D-FACE: An anomaly based distributed approach for early detection of DDoS attacks and flash events. J. Netw. Comput. Appl. 111, 49–63.

Benjamin, H.S., et al., 2010. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. Technical Report, Google, Inc..

Bertero, C., Roy, M., Sauvanaud, C., Trédan, G., 2017. Experience Report: Log mining using natural language processing and application to anomaly detection. In: IEEE 28th Intl. Symposium on Software Reliability Engineering. pp. 351–360.

Bhatt, S., Manadhata, P.K., Zomlot, L., 2014. The operational role of security information and event management systems. IEEE Secur. Priv. 12 (5), 35–41.

Brandón, Á., et al., 2020. Graph-based root cause analysis for service-oriented and microservice architectures. J. Syst. Softw. 159, 110432.

Campos, J.R., Vieira, M., Costa, E., 2018. Exploratory study of machine learning techniques for supporting failure prediction. In: 14th European Dependable Computing Conference. pp. 9–16.

Cao, P., Badger, E.C., Kalbarczyk, Z.T., Iyer, R.K., 2016. A framework for generation, replay, and analysis of real-world attack variants. In: Proc. of the Symposium and Bootcamp on the Science of Security. ACM, NY, USA, pp. 28–37.

Cardenas, A.A., et al., 2013. Big Data Analytics for Security Intelligence. Technical Report, Cloud Security Alliance, URL: https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Big_Data_Analytics_for_Security_Intelligence.pdf.

Catillo, M., Pecchia, A., Villano, U., 2022. AutoLog: Anomaly detection by deep autoencoding of system logs. Expert Syst. Appl. 191, 116263.

Chang, C.-C., Lin, C.-J., 2011. LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. (TIST) 2, 27:1–27:27.

Cinque, M., Cotroneo, D., Della Corte, R., Pecchia, A., 2016. Characterizing direct monitoring techniques in software systems. IEEE Trans. Reliab. 65 (4), 1665–1681.

Cinque, M., Della Corte, R., Pecchia, A., 2017. Entropy-based security analytics: Measurements from a critical information system. In: 47th Annual IEEE/IFIP Intl. Conf. on Dependable Systems and Networks. pp. 379–390.

Cinque, M., Della Corte, R., Pecchia, A., 2020a. Contextual filtering and prioritization of computer application logs for security situational awareness. Future Gener. Comput. Syst. 111, 668–680.

Cinque, M., Della Corte, R., Pecchia, A., 2020b. An empirical analysis of error propagation in critical software systems. Empir. Softw. Eng. 25 (4), 2450–2484.

Cinque, M., Della Corte, R., Pecchia, A., 2022. Microservices monitoring with event logs and black box execution tracing. IEEE Trans. Serv. Comput. 15 (1), 294–307.

2022. Clearwater. https://www.metaswitch.com/products/clearwater-core-ims.

2022. Clearwater test. https://github.com/Metaswitch/clearwater-live-test/.

Cotroneo, D., Natella, R., Rosiello, S., 2017. NFV-Throttle: An overload control framework for network function virtualization. IEEE Trans. Netw. Serv. Manag. 14 (4), 949–963.

D'Amico, A., Whitley, K., 2008. The real work of computer network defense analysts. In: VizSEC 2007. Springer, pp. 19–37.

De Koninck, P., vanden Broucke, S., De Weerdt, J., 2018. Act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (Eds.), Business Process Management. Springer International Publishing, Cham, pp. 305–321.

Di Mauro, M., Liotta, A., 2019. Statistical assessment of IP multimedia subsystem in a softwarized environment: A queueing networks approach. IEEE Trans. Netw. Serv. Manag. 16 (4), 1493–1506.

Dragoni, N., et al., 2017. Microservices: Yesterday, today, and tomorrow. In: Present and Ulterior Software Engineering. Springer International Publishing, Cham, pp. 195–216.

Du, M., Li, F., Zheng, G., Srikumar, V., 2017. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In: Proc. of the ACM SIGSAC Conference on Computer and Communications Security. ACM, NY, USA, pp. 1285–1298.

Fadda, E., Plebani, P., Vitali, M., 2016. Optimizing monitorability of multi-cloud applications. In: Advanced Information Systems Engineering. Springer International Publishing, pp. 411–426.

Farshchi, M., Schneider, J.-G., Weber, I., Grundy, J., 2018. Metric selection and anomaly detection for cloud operations using log and metric correlation analysis. J. Syst. Softw. 137, 531–549.

Freund, Y., Schapire, R.E., 1997. A Decision-Theoretic generalization of on-line learning and an application to boosting. J. Comput. System Sci. 55 (1), 119–139.

Fu, X., Shi, J., Xie, L., 2010. A novel data mining-based method for alert reduction and analysis. J. Netw. 5 (1), 88.

Gan, Y., Dev, S., Lo, D., Delimitrou, C., 2020. Sage: Leveraging ML to diagnose unpredictable performance in cloud microservices. ML Comput. Archit. Syst..

Guo, H., Yuan, S., Wu, X., 2021. LogBERT: Log anomaly detection via BERT. arXiv: 2103.04475.

Hassan, W.U., Noureddine, M.A., Datta, P., Bates, A., 2020. Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis. In: Network and Distributed System Security Symposium.

Hempstalk, K., Frank, E., Witten, I.H., 2008. One-class classification by combining density and class probability estimation. In: Daelemans, W., Goethals, B., Morik, K. (Eds.), Machine Learning and Knowledge Discovery in Databases. Berlin, Heidelberg, pp. 505–519.

Hirakawa, R., Tominaga, K., Nakatoh, Y., 2020. Software log anomaly detection through one class clustering of transformer encoder representation. In: Stephanidis, C., Antona, M. (Eds.), Proc. HCI International - Posters. Springer, pp. 655–661.

Ibidunmoye, O., Rezaie, A., Elmroth, E., 2018. Adaptive anomaly detection in performance metric streams. IEEE Trans. Netw. Serv. Manag. 15 (1), 217–231.

Jain, R., 1991. The Art of Computer Systems Performance Analysis. John Wiley & Sons New York.

Julisch, K., Dacier, M., 2002. Mining intrusion detection alarms for actionable knowledge. In: Proc. of the Eighth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining. ACM, New York, NY, USA, pp. 366–375.

Kavanagh, K.M., et al., 2016. Magic Quadrant for Security Information and Event Management. Technical Report, Gartner Reasearch.

Kobayashi, S., Otomo, K., Fukuda, K., Esaki, H., 2018. Mining causality of network events in log data. IEEE Trans. Netw. Serv. Manag. 15 (1), 53–67.

Li, R.-H., Belford, G.G., 2002. Instability of decision tree classification algorithms. In: Proc. of the 8th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining. In: KDD 2002, Association for Computing Machinery, New York, USA, pp. 570–575. http://dx.doi.org/10.1145/775047.775131.

Li, X., Chen, P., Jing, L., He, Z., Yu, G., 2020. SwissLog: Robust and unified deep learning based log anomaly detection for diverse faults. In: Proc. International Symposium on Software Reliability Engineering. IEEE, pp. 92–103. http://dx.doi.org/10.1109/ISSRE5003.2020.00018.

Lim, C., Singh, N., Yajnik, S., 2008. A log mining approach to failure analysis of enterprise telephony systems. In: Proc. Intl. Conf. on Dependable Systems and Networks.

Liu, F., et al., 2019. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In: Proc. of the ACM SIGSAC Conference on Computer and Communications Security. Association for Computing Machinery, pp. 1777–1794.

Luo, Z., Hou, T., Nguyen, T.T., Zeng, H., Lu, Z., 2020. Log analytics in HPC: A data-driven reinforcement learning framework. In: IEEE Conf. on Computer Communications Workshops. pp. 550–555.

Makhoul, J., Kubala, F., Schwartz, R., Weischedel, R., 1999. Performance measures for information extraction. In: In Proceedings of DARPA Broadcast News Workshop. pp. 249–252.

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., Zhou, R., 2019. LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In: Proc. International Joint Conf. on Artificial Intelligence. International Joint Conferences on Artificial Intelligence Organization, pp. 4739–4745.

Meng, Y., Qin, T., Liu, Y., He, C., 2018. High threat alarms mining for effective security management: Modeling, experiment and application. In: IEEE Symposium on Computers and Communications.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013. Distributed representations of words and phrases and their compositionality. Adv. Neural Inf. Process. Syst. 26.

2022. Netflix hystrix. https://github.com/Netflix/Hystrix.

Nguyen, D.T., Nguyen, K.K., Cheriet, M., 2018. NFV-based architecture for the interworking between WebRTC and IMS. IEEE Trans. Netw. Serv. Manag. 15 (4), 1363–1377.

Noor, A., et al., 2019. A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments. In: IEEE 12th Intl. Conf. on Cloud Computing. pp. 156–163.

Oliner, A., Ganapathi, A., Xu, W., 2012. Advances and challenges in log analysis. Commun. ACM 55 (2), 55–61.

Ruiu, D., 1999. Cautionary tales: stealth coordinated attack how to. URL: http://www.ouah.org/stealthhowto.html.

Santana, M., Sampaio Jr., A., Andrade, M., Rosa, N.S., 2019. Transparent tracing of microservice-based applications. In: Proc. of the 34th ACM/SIGAPP Symposium on Applied Computing. New York, NY, USA, pp. 1252–1259.

Schölkopf, B., Williamson, R.C., Smola, A., Shawe-Taylor, J., Platt, J., 2000. Support vector method for novelty detection. In: Advances in Neural Information Processing Systems.

Sharma, A., Kalbarczyk, Z., Barlow, J., Iyer, R., 2011. Analysis of security data from a large computing organization. In: The 41st IEEE/IFIP Intl. Conf. on Dependable Systems and Networks.

Spathoulas, G.P., Katsikas, S.K., 2010. Reducing false positives in intrusion detection systems. Comput. Secur. 29 (1), 35–44.

Srirama, S.N., Adhikari, M., Paul, S., 2020. Application deployment using containers with auto-scaling for microservices in cloud environment. J. Netw. Comput. Appl. 160, 102629.

Stearley, J., Oliner, A.J., 2008. Bad words: Finding faults in spirit's syslogs. In: IEEE Intl. Symposium on Cluster Computing and the Grid. pp. 765–770.

Sun, Y., Nanda, S., Jaeger, T., 2015. Security-as-a-service for microservices-based cloud applications. In: IEEE 7th Intl. Conf. on Cloud Computing Technology and Science.

2022. Sysdig. https://sysdig.com/opensource/.

Thalheim, J., et al., 2017. Sieve: Actionable insights from monitored metrics in distributed systems. In: Proc. of the 18th ACM Middleware Conference. Association for Computing Machinery, pp. 14–27.

Vaarandi, R., 2008. Mining event logs with SLCT and LogHound. In: IEEE Network Operations and Management Symposium.

Valeur, F., Vigna, G., Kruegel, C., Kemmerer, R.A., 2004. Comprehensive approach to intrusion detection alert correlation. IEEE Trans. Dependable Secure Comput. 1 (3).

Wohlin, C., et al., 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic.

Wolff, E., 2016. Microservices: Flexible Software Architecture. Addison-Wesley Professional.

Wu, L., Tordsson, J., Elmroth, E., Kao, O., 2020. MicroRCA: Root cause localization of performance issues in microservices. In: IEEE/IFIP Network Operations and Management Symposium. pp. 1–9.

Xie, H., Lv, K., Hu, C., 2018. An effective method to generate simulated attack data based on generative adversarial nets. In: 17th IEEE Intl. Conf. on Trust, Security and Privacy in Computing and Communications. pp. 1777–1784.

Xu, W., Huang, L., Fox, A., Patterson, D.A., Jordan, M.I., 2008. Mining console logs for large-scale system problem detection. SysML 8, 4.

Yang, R., Qu, D., Gao, Y., Qian, Y., Tang, Y., 2019. nLSALog: An anomaly detection framework for log sequence in security management. IEEE Access 7, 181152–181164.

Yuan, X., Abouelenien, M., 2015. A multi-class boosting method for learning from imbalanced data. Int. J. Granul. Comput. Rough Sets Intell. Syst. 4 (1), 13–29.
Yuan, Y., et al., 2020. ADA: Adaptive deep log anomaly detector. In: IEEE Conf. on Computer Communications. pp. 2449–2458.
2022. Zipkin. https://github.com/openzipkin/zipkin.
Zoppi, T., Ceccarelli, A., Bondavalli, A., 2016. Context-awareness to improve anomaly detection in dynamic service oriented architectures. In: Computer Safety, Reliability, and Security. Springer Int'l Publishing, pp. 145–158.
Zuo, Y., Wu, Y., Min, G., Huang, C., Pei, K., 2020. An intelligent anomaly detection scheme for micro-services architectures with temporal and spatial data analysis. IEEE Trans. Cogn. Commun. Netw. 6 (2), 548–561.

Marcello Cinque received the Graduate (Hons.) degree and Ph.D. degree from the Federico II University of Naples, Italy, in 2003 and 2006, respectively. He is currently an Associate Professor at the Department of Computer and Systems Engineering, Federico II University of Naples, Italy. His interests include field failure data analysis of distributed systems, and middleware solutions for mobile ubiquitous systems. Dr. Cinque is the Chair and/or TPC member of several technical conferences and workshops on dependable, mobile, and pervasive systems, including IEEE Personal, Indoor and Mobile Radio Communications and ACM International Conference Proceeding Series.

Raffaele Della Corte received the B.S. and M.S. degrees in computer engineering and Ph.D. degree from the Federico II University of Naples, Italy in 2009, 2012, and 2016, respectively. He is currently an Assistant Professor with the Department of Electrical Engineering and Information Technologies, Federico II University of Naples. His research interests include data-driven failure analysis, on-line monitoring of software systems, and security. Dr. Della Corte serves as Reviewer in several dependability conferences and workshops, and he is involved in industrial projects developing techniques for the analysis and monitoring of critical systems.

Antonio Pecchia received the B.S. (2005), M.S. (2008) and Ph.D. (2011) in Computer Engineering from the Federico II University of Naples, Italy. He is now an Assistant Professor of data science at the University of Sannio, Italy. He is a co-founder of the Critiware spin-off company (www.critiware.com). He serves as TPC member and reviewer in conferences and workshops on software engineering and dependability, such as ISSRE, EDCC, LADC and ICTSS. His research interests include deep learning, log analysis, empirical software engineering, intrusion detection, dependable and secure distributed systems.