Master's Programme in Security and Cloud Computing

Anomaly Detection of Web-Based Attacks in Microservices

Master's Thesis

Eljon Harlicaj

Aalto University
School of Science

EURECOM

MASTER'S
THESIS

Aalto University - EURECOM
MASTER'S THESIS 2021

Anomaly Detection of Web-Based Attacks in Microservices

Anomaly Detection of Web-Based Attacks in Microservices

Eljon Harlicaj

This thesis is a public document and does not contain any confidential information.

Cette thèse est un document public et ne contient aucun information confidentielle.

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.
Espoo, 16 July 2021

Supervisor: Prof. Mario Di Francesco, Aalto University
Co-Supervisor: Prof. Davide Balzarotti, EURECOM

Aalto University - School of Science
EURECOM

Master's Programme in
Security and Cloud Computing

Aalto University
School of Science

EURECOM
Sophia Antipolis

Abstract

Author
Eljon Harlicaj

Title
Anomaly Detection of Web-Based Attacks in Microservices

School School of Science

Degree programme Master of Science

Major Security and Cloud Computing (SECCLO) **Code** SCI3084

Supervisor Prof. Mario Di Francesco, Aalto University
Prof. Davide Balzarotti, EURECOM

Level Master's thesis **Date** 16 July 2021 **Pages** 55 **Language** English

Abstract

Cybercriminals exploit vulnerabilities in web applications by leveraging different attacks to gain unauthorized access to sensitive resources in web servers. Security researchers have extensively investigated anomaly detection of web-based attacks; however, the cloud-native paradigm shift combined with the increasing usage of microservices introduces new challenges and opportunities.

This thesis studies relevant research in anomaly detection of web-based attacks and proposes new methods for modeling regular web requests and the inter-service communication patterns in modern web applications. Specifically, we present a solution that leverages service meshes for collecting web logs in cloud environments without accessing the source code of the applications. First, we present the design and implementation of a method to abstract from web logs to Log-Keys sequences for performing anomaly detection with Long Short-Term Memory Recurrent Neural Networks. Second, we implement Autoencoders to detect anomalies in the content of web requests. Finally, we create two datasets and conduct experiments to analyze and evaluate our solution.

We perform an extensive analysis of the parameter space and the related impact on the anomaly detection performance. By an appropriate choice of these parameters, our solution is able to detect 91% of the anomalies in the considered dataset with only a 0.11% false positive rate.

Keywords: anomaly detection, artificial intelligence, cloud security, deep learning, machine learning, microservices, web security

Aalto University
School of Science

EURECOM
Sophia Antipolis

Abstrait

Auteur
Eljon Harlicaj

Titre
Anomaly Detection of Web-Based Attacks in Microservices

Programme d'Études Double Diplôme de Master

Filière d'Attachement Security and Cloud Computing (SECCLO)

Encadrants Académiques Prof. Mario Di Francesco, Aalto University
Prof. Davide Balzarotti, EURECOM

Categorie La Thèse de Master **Date** 16 July 2021 **Pages** 55 **Langue** Anglais

Abstrait

Les cybercriminels exploitent les vulnérabilités des applications web en recourant à différentes cyberattaques afin d'obtenir des accès non légitimes à des ressources critiques présentes sur les serveurs web. Les chercheurs en sécurité ont intensémment étudié le sujet de la détection d'anomalies liées aux attaques en ligne, cependant le changement de paradigme en cloud-native combiné à l'utilisation croissante des microservices introduit de nouveaux défis et de nouvelles opportunités.

Cette thèse étudie les recherches pertinentes en matière de détection d'anomalies d'attaques basées sur le web et propose de nouvelles méthodes pour modéliser les requêtes web regulières et les patterns de communication inter-services dans les applications web modernes. Plus précisément, nous présentons une solution qui exploite les service meshes pour collecter les web logs des environnements cloud sans directement accéder au code source des applications. Dans un premier temps nous présenterons la conception et la mise en oeuvre d'une méthode permettant d'extraire des séquences clé depuis les web logs dans le but d'effectuer la détection d'anomalies grâce à un Long Short-Term Memory Recurrent Neural Networks. Ensuite, nous mettons en œuvre des autoencoders pour détecter des anomalies dans le contenu des requêtes Web. Enfin, nous créons deux datasets et menons des expériences pour analyser notre solution.

Nous effectuons une analyse approfondie de l'espace des paramètres et de leur impact sur la performance dans le cadre de la détection des anomalies. Grâce à un choix approprié de ces paramètres, notre solution est capable de détecter 91% des anomalies dans le dataset considéré avec un taux de faux positifs de seulement 0.11%.

Keywords: anomaly detection, artificial intelligence, cloud security, deep learning, machine learning, microservices, web security

To Enver, Merushe and Erald,
thank you for everything.

Acknowledgement

I would first like to thank my supervisors, **Professor Mario Di Francesco** and **Professor Davide Balzarotti**, for allowing me to perform research following my intuitions and for guiding me all along this journey. Moreover, thank **Professor Giovanni Vigna** and **Professor Cristopher Kruegel**, I truly appreciate your confidence in me.

I am grateful to my family and friends for their continuous support throughout my studies. Last but not least, thank you, **Hsin-Yi Chen**, for your lovely smile that brightened even the darkest days in Finnish winter.

SECCLO
Master's Programme in
Security and Cloud Computing

Aalto University
School of Science

EURECOM
Sophia Antipolis

Contents

List of Tables

List of Figures

Abbreviations

AI	Artificial Intelligence
BCE	Binary Cross-Entropy
CAE	Convolutional Autoencoders
CFS	Correlation Feature Selection
CNN	Convolutional Neural Network
DL	Deep Learning
EFK	Elasticsearch, Fluentd, and Kibana
IDS	Intrusion Detection Systems
JSON	JavaScript Object Notation
K8S	Kubernetes
LSTM	Long Short-Term Memory
ML	Machine Learning
mRMR	minimal-Redundancy-Maximal-Relevance
NN	Neural Network
OS	Operating System
RE	Regular Expressions
RNN	Recurrent Neural Networks
SM	Service Mesh
SQLI	SQL Injection
SRS	Stan's Robot Shop
VEGP	Vanishing Gradients and Exploding Gradient Problem
VM	Virtual Machines

Chapter 1

Introduction

"Prima di essere ingegneri voi siete uomini"

"Bevor ihr Ingenieure seid, seid ihr vor allem Menschen"

Francesco de Sanctis (1817-1883)

1.1 Motivation

Cybercriminals exploit vulnerabilities in the source code of web applications for unauthorized access to databases and resources in web servers by leveraging different web-based attacks (e.g., Cross-Site Scripting and SQL Injection) to accomplish their malicious goals.

Although security researchers have extensively investigated anomaly detection of web-based attacks, the cloud-native paradigm shift combined with the increasing usage of microservices – an architectural pattern that defines an application as a collection of independent services – introduces new challenges and opportunities.

Users expect modern web applications to please their needs fast and reliably regardless of their device, geographical location, or time. Developers exploit the latest technology to meet users' expectations, and in this sense, the use of cloud services and scalable solutions combined with modular (microservices) and distributed architectures are the preferred solutions. A recent survey [50] shows that 96% of respondents are familiar with microservices, and 73% of these have already integrated microservices into their application process. Further, 88% of respondents use REST, 67% uses containers, and 66% use cloud services.

With microservices becoming the fundamental blocks of modern web applications, studying the communications between those elements enables learning the workflow initiated from incoming requests and, consequently, modeling an expected behavior for future requests. This idea, combined with more classic research on features derived from web requests, enables high anomaly detection rates of web-based attacks.

1.2 Problem Statement

This thesis explores anomaly detection of web-based attacks on microservices by modeling regular web requests and the communication behavior between microservices in modern web applications. Moreover, the microservice architecture is one of the newest and widely-used patterns to develop applications; hence, detecting anomalies in microservices is of critical importance.

Existing cloud anomaly detection services mostly come as analysis of time series with metrics related to the infrastructure (CPU usage, memory usage). After reviewing the most recent research in web-based attacks' anomaly detection, we found a gap between the techniques applied to monolithic applications and microservices.

This thesis explores and proposes anomaly detection techniques of web-based attacks in microservices architecture, filling the current research gap.

1.3 Contribution

This thesis aims to design and develop an effective method to detect web-based attacks in a microservices architecture. At the moment, most research on anomaly detection of web-based attacks is targeting monolithic web applications without adequate consideration of the paradigm shift towards cloud computing.

This thesis contributions are the following.

- It designs and proposes a Kubernetes-based deployment that leverages service mesh software to collect and store web logs effortlessly.
- Two datasets consisting of web-requests logs of an e-commerce application are collected.
- A Long Short-Term Memory (LSTM) [26], an artificial Recurrent Neural Networks (RNN), to detect anomalies on sequences of log-keys generated from collected web logs.
- A set of Autoencoders, a type of Neural Networks (NNs), specialized in API endpoints for detecting anomalies in web requests.

1.4 Structure of the Thesis

The rest of the thesis is organized as follows:

The second chapter introduces the key cloud-based technologies considered in this thesis.

The third chapter introduces anomaly detection and reviews the most relevant literature in the context of web-based attacks, describing challenges and proposed solutions.

The fourth chapter presents this thesis's main contributions introducing the designed log collection method and the Long Short-Term Memory anomaly detection model for Log-Key Sequences.

The fifth chapter presents relevant features of web requests and the usage of Autoencoders for anomaly detection.

The sixth chapter presents the experiment setup, the analysis of the proposed solution, and a performance evaluation.

The seventh chapter concludes the thesis by summarizing our contribution and proposing future works related to this thesis's research.

Chapter 2

Background

This chapter overviews the technologies and concepts that are relevant to the thesis work.

It first introduces the microservices architecture and relevant use cases. Next, it discussed Kubernetes (K8S) and leveraging K8S to accomplish this thesis' goals. Finally, it defines service mesh software by overviewing different functions and usage scenarios.

2.1 Microservices

We define microservices as tiny applications independently deployable, scalable, testable, and with a single responsibility leading to loosely-coupled design [47]. Loose coupling is a design pattern (or paradigm) describing systems built on various components detached from each other. In comparison, monolithic architectures are software built as one large system, hard-to-scale on-demand, large codebase, and tightly-coupled design. Tight coupling is another design pattern describing systems built with a specific purpose and components bound to each other.

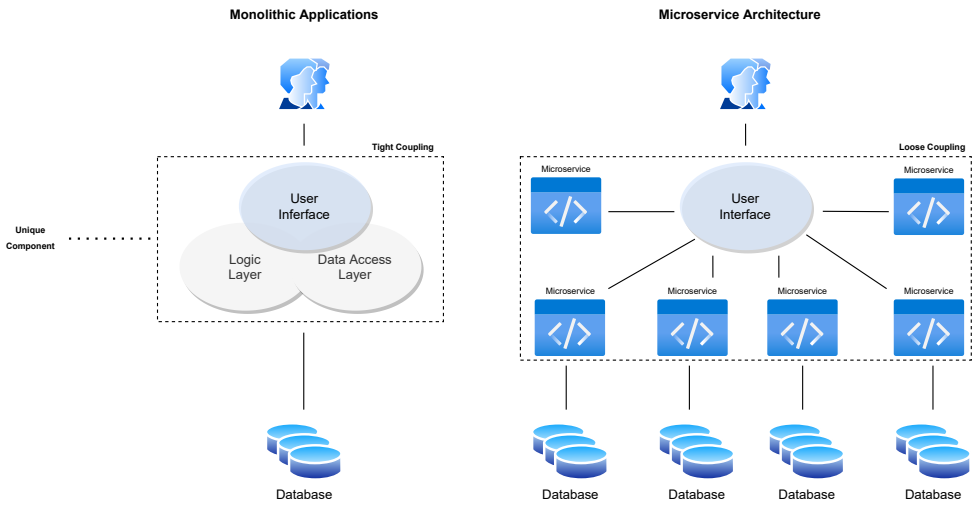


Figure 2.1. High-level view of monolithic and microservice architectures

The use of microservice architecture – and consequently loose coupling – has

been steadily increasing over the years and became the de-facto best practice for designing software. The growing adoption of microservice follows with the increasing adaption of representational state transfer application programming interfaces (REST APIs). REST are architectural constraints, and API are definitions – and protocols – for building and integrating application software such as web applications and microservices. In other words, REST APIs help communication following predefined rules and provide a lightweight communication mechanism between web components. Microservices implementing REST APIs can be rapidly updated, deployed, and scaled.

This thesis focuses on web applications developed following microservice architecture with HTTP REST APIs for communication between components. Figure 2.2 gives a high-level overview of monolithic and microservice architectures.

2.2 Kubernetes

Containers, such as Docker[1], are the enabling technologies behind the current paradigm shift towards Cloud Computing [45]. Docker is an open-source project aiming to simplify and automate as much as possible the deployment of applications [9]. On a higher level, Kubernetes[2] poses itself as a cluster manager for containers and aims to provide automated deployment, scaling, and management of containerized applications.

To better understand the need for K8S, we must discuss how software deployment evolved over time[3]:

- **Traditional deployment era:** an application would run on a physical machine, without real boundaries on resources. This leads to resource allocation issues, which could only be solved through dedicated physical machines for each application.
- **Virtualized deployment era:** allows to run multiple Operating Systems (OSes) on a physical machine as Virtual Machines (VM). A hypervisor then provides mechanisms for resource allocation and isolations between applications running in different VMs. Each VM runs its own OS on top of the virtualized hardware.
- **Container deployment era:** containers are similar to VMs but share the underlying OS. As a consequence, containers are lighter than VMs, even though containers have their own file system, share of CPU, memory, and other resources.

[1] https://www.docker.com/
[2] https://kubernetes.io/
[3] https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/#going-back-in-time

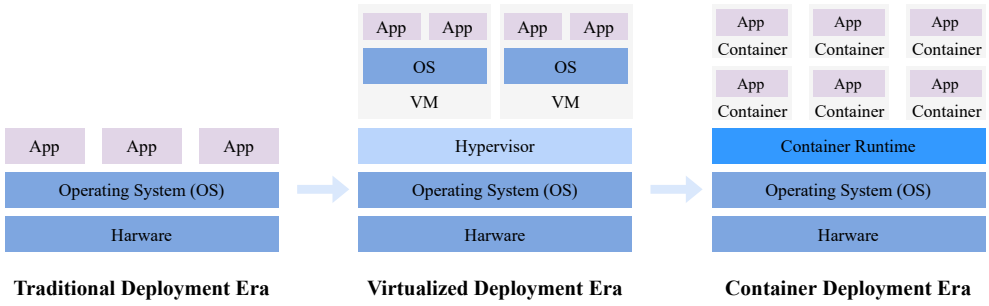


Figure 2.2. Evolution of deployment philosophy

The microservice architecture heavily relies on container deployment for agile application creation and development. In addition to loosely coupled applications, performance isolation, and resource utilization, microservices allow sysadmins to focus on application deployment by raising the level of abstraction from managing the OS and the hardware to managing virtual resources. Furthermore, container deployment brings environmental consistency and portability. Moreover, it simplifies the deployment cycle introducing a clear separation between development and IT operations.

Containers are an efficient solution for bundling and running applications but introduce an additional level of indirection and abstraction. Currently, real-world applications are composed of hundreds of microservices and suffer from classical challenges such as failures. Considering the number of microservices and complexity of applications, automated systems to manage container deployment, scaling, and updates are vital. Kubernetes are efficient solutions to those challenges and aim to ease the automated deployment of containerized applications by offering service discovery and load balancing, storage orchestration, automated rollouts and rollbacks, automatic bin packing, self-healing, secret and configuration management[4]. K8S is complex computer software with several components. A K8S cluster has multiple components (Figure 2.3) performing different tasks:

- **Control Plane:** The Control Plane acts as a container for other components. From a high-level point of view, it is responsible for making global decisions as well as detecting and responding to events at the cluster level.
  - `kube-apiserver`: Control plane component exposing the K8S control plane API.
  - `etcd`: Control plane component, acting as a key-value store of cluster information such as configuration.
  - `kube-scheduler`: Control plane component, acting as a manager of Pods and nodes. It continuously checks for newly created Pods with no

[4] `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/#why-you-need-kubernetes-and-what-can-it-do`

assigned node and assigns Pods to nodes for them to run.

- `kube-controller-menager`: Control plane component running controller processes. Controller processes have different types and tasks. Responsibilities of controllers are noticing and responding to nodes going down, creating Pods to run one-off tasks, creating account and API access tokens for new namespaces, and others.
- `cloud-control-manager`: Control plane component handling cloud-specific control logic. In other words, it is the manager enabling the linking of clusters into the cloud provider's API by separating components interacting with the cloud platform and those that only interact with the K8S cluster.

- **Node Components:** On the other hand, node components run on every node and maintain healthy running environments:
  - `kubelet`: Node component agent that controls if containers are running in a healthy state.
  - `kube-proxy`: Node component that works as a proxy for each node by maintaining network rules on nodes.
  - `Container runtime`: Software responsible for running containers (e.g., Docker)

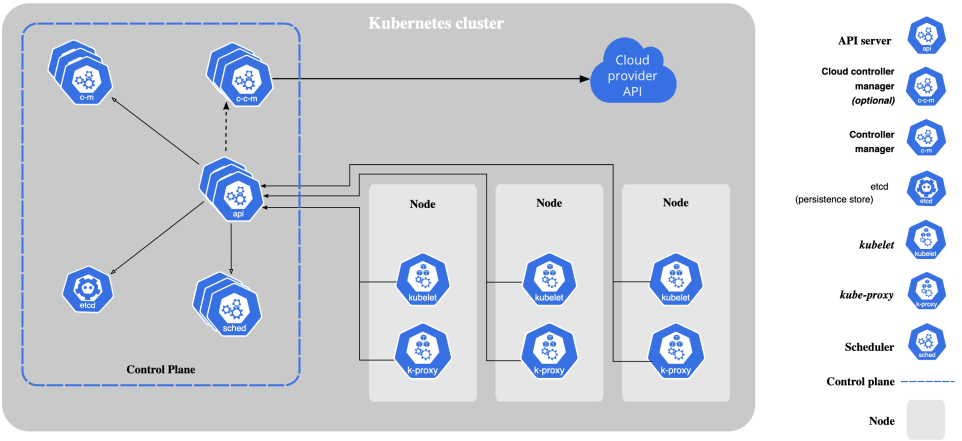


Figure 2.3. Diagram of a Kubernetes cluster[5]

This thesis focuses on a single node cluster with multiple Pods, each running a microservice. Pods are the minor deployable units of computing that can be managed in Kubernetes[6]. A Pod, depending on the application, can run a single or multiple containers. Each Pod can have one or multiple volumes for storage. Pod's resources, such as network and volume, are shared between containers running in the Pod. Pod's content is located within the same context and scheduled

[6] `https://kubernetes.io/docs/concepts/workloads/pods/`

simultaneously. In other words, a Pod models application-specific logic in a relatively tightly coupled manner. Figure 2.4 shows the diagram of a Kubernetes node.

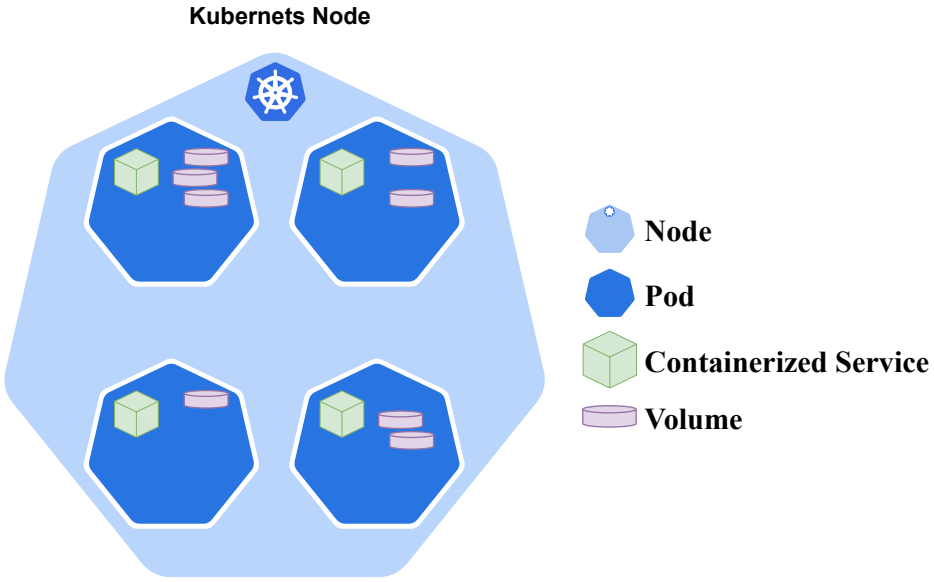


Figure 2.4. Diagram of a Kubernetes Node

2.3 Service Mesh

In a microservice-based application, a service is responsible for a specific task and has specific logic. Often, service relies on each other to perform actions and some services have higher load than others. The loose coupling of microservice-based applications introduces new challenges related to the observability of the systems and understanding their work and data flow. Recently, to address those challenges, Service Meshe (SM) solutions have been developed.

A SM is a software that gives observability on how different services communicate and share data. In other words, SMs introduce a dedicated infrastructure layer that reports how different components of an application interact. Such a layer helps sysadmins find and understand communications patterns, bottlenecks, data-flow, and locate performance issues, thus avoiding downtime as an application scale. With the growing complexity and the size of components in applications, SMs have become of vital importance. SMs free the developer from introducing reporting functionality by explicitly instrumenting application code. They take the logic governing service-to-service communication out of individual services and abstract it to a layer infrastructure[7]. As an analogy, SMs work similarly to a proxy. The SMs are attached to an app as a web of network proxies. The SMs

[7] `https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh`

embeds proxy-like components, called sidecars, alongside each component of an application. In other words, each service resides alongside a sidecar proxy as shown in Figure 2.5.

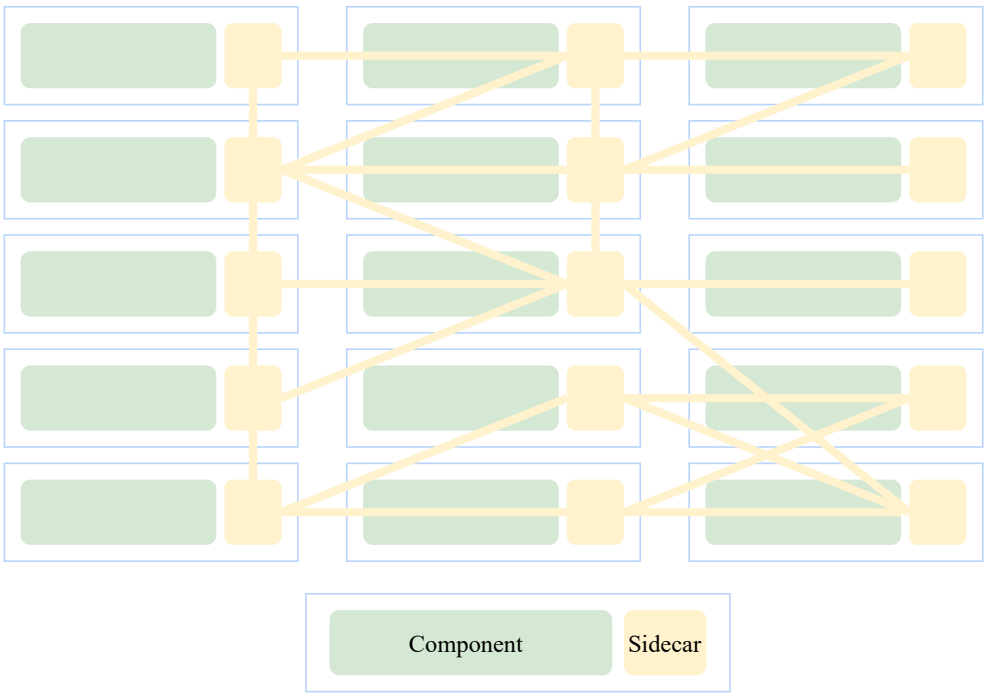


Figure 2.5. Service Mesh proxy network with sidecars

With the fast-growing pace of applications, the communication environment becomes increasingly complex, introduces possible failures, and, given their loosely coupled nature, difficulty understanding the source of problems. The SMs comes as a solution for identifying the source of problems by introducing means of capturing multiple aspects of inter-service communications and making SM software critical to any application base on microservices.

There are different solutions for SMs. Among them, Istio is an open-source project software with a vibrant community with interesting key features:

- Ease of deployment and use with K8S.
- Envoy[8], an embedded high-performance C++ distributed network proxy deployable as a sidecar alongside containers.
- Envoy's scripting and filtering capabilities to intercept and modify content and format of web requests at runtime.

Figure 2.6, illustrates how the architecture of a Kubernetes Node (in Figure 2.4) changes by introducing of Istio. The figure assumes that Istio is deployed in the same node as the target application. We enable Istio's injection capability, and

[8] `https://www.envoy.com`

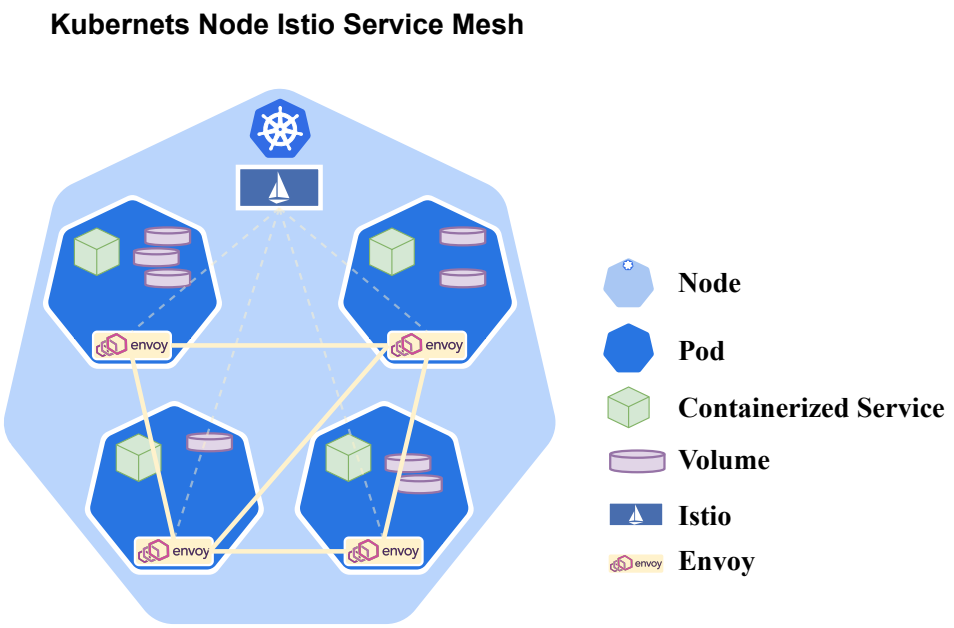


Figure 2.6. Diagram of a Kubernetes Node with Istio Service Mesh

we deploy a sidecar alongside each service (recall that we define as service each component of an application). Each service has an Envoy proxy sidecar. Thus, all the communication between services must go through the proxy. Istio supervises the communication and collects reports from the deployed sidecars.

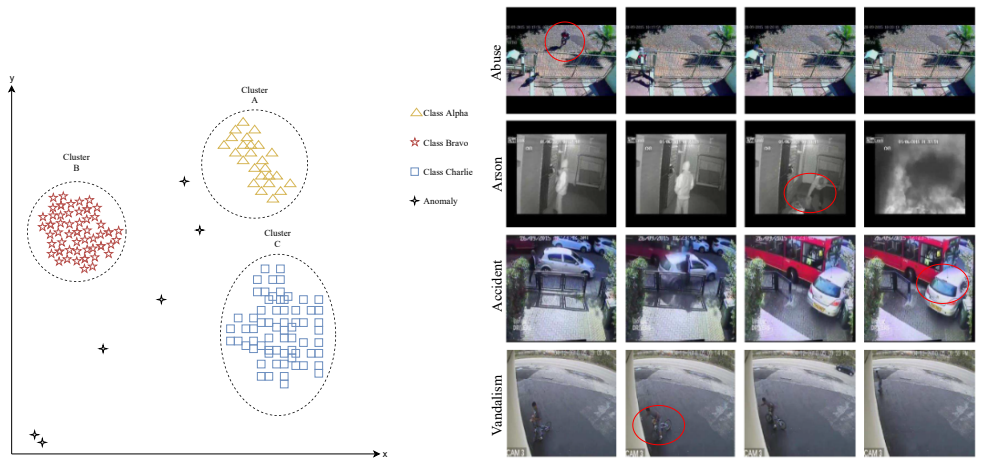
Chapter 3

Anomaly Detection

This chapter gives an overview of the large field of anomaly detection and the use cases. Next, it comprehensively illustrates the significant challenges identified in scientific cross-field surveys. Finally, it summarizes and analyzes papers related explicitly to anomaly detection for web applications.

3.1 Definition

Anomaly detection (AD) is an exciting problem across multiple disciplines that has attracted significant research over the years. From a high-level perspective, anomaly detection aims to find and model patterns in a dataset and subsequently locate nonconforming data elements in a data-driven fashion [13]. Nonconforming data elements are usually referred to as anomalies, novelties, and outliers. From an abstract level, anomalies are defined as patterns in data not conforming to expected normal behavior [13]. Anomaly detection is of relevance in situations where identifying outliers is critical to the system. As an example, anomaly detection is largely used in network intrusion detection at the packet-level communication [19, 36, 37], detection of fraudulent credit card transactions [4], behavior-based malware detection [10], structural health monitoring [6] and others.



(a) AD with clustering

(b) DL for AD in surveillance videos [46]

Figure 3.1. Examples of anomaly detection uses cases

Given the challenges with data heterogeneity between disciplines, anomaly detection methods are often specific to certain problems. On the other hand, sharing the underlying techniques (e.g., algorithms) across disciplines is frequent. More recently, the increasing availability of computation power is making Artificial Intelligence (AI) approaches based on Machine Learning (ML) more widespread, as opposed to traditional techniques based on data density [31], correlation [34], subspaces [33], deviation rules [30], fuzzy logic [48], and cluster analysis [11]. Deep Learning (DL), a field of ML, is profitably applied to anomaly detection in diverse fields. Thanks to scientific advancements, DL approaches are outperforming many traditional AD methods [12, 28, 42].

3.2 Challenges

Anomalies are defined as patterns in data non conforming to normal behavior. As a consequence, detecting anomalies requires recognizing and modeling expected behaviors, therefore identifying data with behavior that the constructed models cannot explain. This simple approach is very challenging in reality. The following describes some of the most critical challenges identified by the surveys [13, 12] and in the recent work of Peng et al. [40].

- **Unkownness:** Anomalies are often associated with novelties. Therefore they remain unknown until they occur. Moreover, anomalies are related to unknown behaviors, failures and distributions.
- **Heterogeneity:** By nature, anomalies are irregular. The intrinsic heterogeneity between anomalies makes detection problematic since one class of anomalies is entirely different from another.
- **Scarcity:** Anomalies happen very infrequently. Therefore it is challenging to collect a considerable number of anomalies for the analysis purpose.
- **Class imbalance:** The scarcity of anomalous instances results in problematic datasets with imbalanced classes between normal and abnormal instances.
- **Recall rate:** The combination of scarcity and heterogeneity results in a low probability to detect anomalous data. Moreover, this condition results in a high false-positive rate by incorrectly detecting regular instances as anomalies and a high false-negative rate by missing to identify anomalies on instances with sophisticated features.
- **High-dimensional data**: Detection of anomalies in low-dimensional spaces have straightforward solutions since the abnormal characteristics of the data are easy to model. On the other hand, those characteristics become hidden and often unnoticeable in high-dimensional data, thereby making the problem much more challenging [51].

- **Data dependencies:** Detecting anomalies in instances somehow related to each other requires different approaches from detecting anomalies in unrelated data. Detecting anomalies from instances dependent on each other is a known, challenging problem [1].
- **Data-efficient learning:** Collecting clean datasets has a high cost, and labeling instances as normal or abnormal is difficult. Unsupervised anomaly detection is vastly in use. Unsupervised AD approaches do not require labeled datasets and do not have a prior definition of anomalies. That is, unsupervised approaches heavily rely on the data distribution and assumptions learned during the training of the model.
- **Noise-resilience:** Supervised AD methods require data to be labeled. The issue resides in the assumption that data labeled data sets are clean. In fact, datasets can contain noise i.e., instances wrongly labeled. Therefore, supervised AD could learn from instances with noise and subsequently perform poorly. The main challenge resides in the irregular distribution of such noisy instances in a dataset.
- **Complexity:** Most of the existing AD methods are designed to detect abnormal data as single instances. Anomalies in complex relationships and dependencies between instances are challenging problems. One of the exciting challenges here is to integrate the concept of conditional and group anomalies into AD models. Moreover, it is challenging to consider input data from different sources and develop AD approaches to perform detection with incoming data from multiple data sources. An example is given by anomalies in a video by considering image frames, audio, text, and the relation between the elements in the video.
- **Anomaly explanation:** AD systems are often used as black-box models. Models, such as AD systems, could be responsible for algorithmic bias towards minority groups underrepresented in the training dataset. In other cases, the explainability of an anomaly is not possible. Deriving anomaly explanation from specific detection methods is still a largely unsolved problem, especially for complex models [40].

New DL methods can partially address challenges related to unkownness, heterogeneity, and data dependencies. On the other hand, approaches that effectively address the rarity of anomalies, complexity, and anomaly explanation are still open problems and mainly tackled with heuristics based on specialized know-how.

3.3 Web-Based Scenarios

Anomaly Detection of Web-Based Attacks is the scenario that is most relevant to this thesis. As a consequence, the rest of this section introduces related

work and methods related to our research and motivates the need for new approaches specifically designed for cloud applications developed following the microservice paradigm.

The nature of web applications is to be open to the network. The widespread use of such applications gives malicious entities a vast attack surface and researchers the puzzling problem of detecting and preventing attacks. To detect known attacks, misuse detection systems based on signatures are employed. From a high-level point of view, a signature is a sequence of bytes modeling well-known attacks with the end goal of detecting them by matching their signature to incoming web traffic. Signature-based intrusion detection systems (IDS) are leveraged in legacy systems. Due to their nature, they are unable to detect unknown attacks. Moreover, IDSs are time-consuming to maintain, given the speed at which new unseen attacks are created and detected in the wild. To address these significant drawbacks, AD systems based on other techniques have been developed.

Krugel and Vigna [35] developed an IDS that effectively applies several different anomaly detection methods to address the challenges. The authors start by analyzing and modeling HTTP requests as logged by standard web servers. To this end, they extract URIs from successful requests, the related path to the desired resource, path information, and query strings. A query string is an optional part of the URI composed of parameters and values. Subsequently, processed data goes into a pipeline composed of several detection models. Each model outputs a probability value in a defined Anomaly Score equation. The first model relies on attribute length to approximate the unknown distribution of the lengths of these values. The second relies on *Idealized Character Distribution*, following the intuition that characters across attributes occur with different frequencies. The third model relies on Bayesian inference to derive a Markov model and create a probabilistic grammar describing attributes to detect attacks respecting normal character distribution and therefore evade the second model. The subsequent model is responsible for learning if a particular attribute is drawn from a set of known elements to detect attribute enumeration. The last model analyzes the attribute order in a query string by creating directed graphs with the intuition that the order of attributes should not change across different requests. Each model outputs an anomaly probability value, and all scores are finally part of the *Anomaly Score* equation. The authors' work shows the effectiveness of combining different detection models based on statistics and know-how by developing a solution that delivers a low number of false positives.

Cho and Cha [14] proposed a web session anomaly detection based on parameter estimation. A web session is a sequence of web pages requested by a

user. The authors' work shows that Bayesian estimation effectively determines anomalous web sessions without knowledge of web request characteristics in advance. Unfortunately, the proposed method also shows a high false-positive rate that prevents its in real-world scenarios.

Nguyen et al. [39] developed GeFS, a series of techniques based on generic feature selection measures for web intrusion detection. They propose The Correlation Feature Selection (CFS) Measure and The minimal-Redundancy-Maximal-Relevance (mRMR) Measure. CFS linearly characterized the relevance of features and their relationship. mRMR considers non-linear relationships in features by studying mutual information between them. The authors started by identifying 30 possible features (Table 3.1) in real large-scale web requests datasets. The authors' approach shows that most of the features are either linearly or non-linearly correlated. They claim that not all the features are required for effective anomaly detection and proof they result by running detection techniques on derived important features.

Table 3.1. Web request features identified by Nguyen et al. [39]. The symbol ⋆ marks the most important features.

Feature Name	**Feature Name**
Length of the request ⋆	Length of the path ⋆
Length of the arguments ⋆	Length of the header ⋆ "Accept"
Length of the header "Accept-Encoding"	Length of the header "Accept-Charset"
Length of the header "Accept-Language"	Length of the header "Cookie"
Length of the header "Content-Length"	Length of the header "Content-Type"
Length of the Host	Length of the header "Referer"
Length of the header "User-Agent"	Method identifier
Number of arguments ⋆	Number of letters in the arguments ⋆
Number of digits in the arguments ⋆	Number of 'special' char in the arguments ⋆
Number of other char in the arguments	Number of letters char in the path ⋆
Number of digits in the path ⋆	Number of 'special' char in the path ⋆
Number of other char in path	Number of cookies
Minimum byte value in the request	Maximum byte value in the request ⋆
Number of distinct bytes	Entropy
Number of keywords in the path	Number of keywords in the arguments

Fan and Guo [16] introduced an approach relying on the normalization of web request URLs and HTTP requests. Firstly, destination URLs are extracted from web logs. Subsequently, the results are partitioned based on request method types and other standard features such as host, date, and IP address. By analyzing the resulting partitions, the authors built multiple detection models based on hidden Markov models and decide whether an unseen request is normal or an anomaly. Their work demonstrates the capabilities of adaptive models reporting low false-positive alerts in the order of 0.5% between different datasets.

Zolotukhin et al. [52] considers the analysis of HTTP requests for the detection of network intrusions. First, they collected a dataset of web requests without

known anomalies. Second, they trained different machine learning models based on n-grams and clustering to detect anomalies. *These models are then used to detect network attacks as deviations from the computed norms*.

Kozik et al. [32] propose a different approach by modeling HTTP Requests with Regular Expressions (RE) for detecting web attacks. They start by modeling normal requests sent from the client to the server so as to find REs able to group similar HTTP requests together. To this end, they analyze and represent URLs as graphs, whose vertices represent HTTP request parameters. The challenge of building REs to model normal behavior starting from graphs can be formalized as a graph segmentation problem, and they tackle this by using a similar algorithm to the one proposed in [17]. The shown results are promising and outperforming previously described methods such as [39, 35] in the CSIC-2010 [21] dataset (Table 3.2).

Table 3.2. Results obtained by Kozik et al. [32] on CSIC-10 [21] dataset.

Method	**Detection Rate**	**False Positive Rate**
Kozik et al. [32]	94.46%	4.34%
Nguyen et al. (avg.) [39]	93.65%	6.9%
ICD [35]	78.50 %	11.9%
SCALP GET+POST[9]	19.00%	0.17%
SCALP GET only	9.16%	0.09%

Following the increasing availability of affordable computation power ML frameworks, Althubiti et al. [5] experimented with several ML techniques on real large-scale datasets. The authors' work begins by ranking nine HTTP features used in [39] by using attribute evaluator methods proposed in [25] and selecting the best five in their applications (Table 3.3). The study shows how different sets of features could be effective with different ML approaches for anomaly detection on HTTP requests. The authors claim to achieve higher accuracy rates than [39] and the similar work conducted by Pham et al. [43].

Park et al. [41] argued that anomaly detection methods selecting features based on heuristics result in limited performance given the weak understanding of HTTP messages. They propose a method based on Convolutional Autoencoders (CAE) with character-level binary image transformation. In other words, HTTP requests messages are transformed into images and given as input to the CAE. The CAE consists of an encoder and decoder with a convolutional neural network (CNN) structure. In the first phase, the encoder takes an image as input and transforms it into a latent representation. In the second phase, the latent representation is input to the decoder, and the output is another image. Finally, the CAE is trained to minimize the binary cross-entropy (BCE) between input and output. For nonanomalous HTTP messages, the model produces outputs similar

Table 3.3. Names of 9 features that are considered relevant for the detection of Web attacks. ⋆ marks the most important features identified by Althubiti et al. [5].

Feature Name
Length of the request ⋆
Length of the arguments ⋆
Number of arguments ⋆
Number of digits in the arguments
Length of the path ⋆
Number of letters in the arguments
Number of letter chars in the path
Number of “special” chars in the path ⋆
Maximum byte value in the request

to the inputs with low BCE. If a message is anomalous, the model's ability to produce similar outputs is weak, resulting in a high BCE. By carefully selecting a threshold value for the BCE, anomalies can be detected [2].

Anomaly detection of web-based attacks has been a hot topic in the last 20 years. The academic community proposed different methods ranging from pure statistic-based solutions to artificial intelligence with machine learning algorithms and, most recently, deep learning. With the current paradigm shift to cloud computing and the increased use of microservices, previously proposed techniques and methods must be adapted and combined to serve their purpose. To the best of our knowledge, the existing literature has not yet considered AD in the context of web-based attacks targetting microservices. That is, this thesis's goal is to introduce methods for anomaly detection of web-based attacks in microservice architecture.

Chapter 4

Log-Key Anomaly Detection

This chapter overviews log collection and the related infrastructure offered by Kubernetes (K8S). Next, it introduces the **E**lasticsearch[10], **F**luentd[11], and **K**ibana[12] (EFK) stack for log collection, describes its architecture and presents our log collection method based on service mesh and the EFK stack. Finally, the chapter discusses our solution for Log-Key creation by abstracting HTTP requests into discrete sequences, then presents our Log-Key anomaly detection method based on Long Short-Term Memory (LSTM) [26].

4.1 Log Collection

K8S is the de-facto industry standard for container orchestration and is characterized by highly distributed environments with tens of machines, hundreds of containers, and different actions such as deployment, update, termination, restart and reschedule. Logging poses unique challenges in such a dynamic environment, and is essential to gain observability on the system. The following introduces the basic concepts on the K8S logging infrastructure, additional tools, and combining those with service mesh software to collect useful logs in web applications. The rest of the discussion is primarily based on the material in [27, 29, 3].

4.1.1 Kubernetes Logging Infrastructure

There are two main methods to accomplish log collection in K8S: **kubelet** and **sidecar**. Moreover, K8S offers system component logging. K8S' system components are services enabling the correct functioning of nodes and clusters. System component logging is beyond the scope of this thesis, but we will briefly introduce it here for completeness.

4.1.1.1 Kubelet

K8S offers out-of-the-box native logging through the **kubelet** service present at each node. *Kubelet* service works by redirecting applications' output to their re-

[10] `www.elastic.co`
[11] `https://www.fluentd.org/`
[12] `https://www.elastic.co/kibana`

spective pod `stdout` and `stderr` streams. Moreover, *kubectl* is a command-line tool that allows retrieving logs on a pod. Retrieving logs from all pods and aggregate them in a single place is not natively supported by K8S, but custom scripts such as **kubetail**[13] can help in accomplishing this.

4.1.1.2 Sidecar

The **sidecar** pattern (see also Section 2.3 and Figure 4.1) allows to collect logs in a systematic and scalable fashion. A Pod is the basic atomic unit of deployment in k8s. Pods contain one or more containers and share volume and network. A sidecar is nothing more than an additional lightweight container in the Pod. In our use case, and following the separation of concerns principle, sidecars allow the collection and shipping of application logs to an external log aggregator.

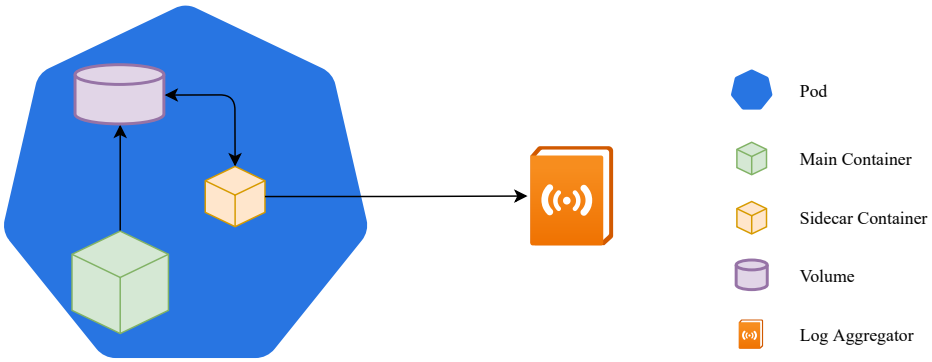


Figure 4.1. Sidecar pattern for log collection and shipping to an aggregator.

4.1.1.3 Kubernetes System Component Logging

In addition to node services (such as **kubelet**), K8S offer logging capabilities at the cluster level for system components. The main system components are `kube-apiserver`, `kube-scheduler` and `etcd`. Let us recall that:

- `kube-apiserver` acts as the main access point to the cluster;
- `kube-scheduler` is the component responsible to determines into which Pod a container has to be deployed;
- `etcd` is the standard key-value pair store system used cluster configuration storage.

Additionally, there are other system components, some run in a container in the cluster, but primarily they run on the operating system level as system services. Furthermore, K8S also supports additional data types for logging, such as *events* and *audit logs*. *Events* can indicate and report about resource states, thereby being critical to investigate performance issues. Finally, *audits* logs are helpful for compliance by recording all actions taking place in the system.

[13] `https://github.com/johanhaleby/kubetail`

4.1.2 EFK Stack

The **E**lasticsearch, **F**luentd, and **K**ibana (EFK) stack is a centralized logging solution that helps to collect, sort, and analyze a large volume of data produced by your application. **Elasticsearch** is a real-time distributed, free, open-source and analytics engine for data. Elasticsearch search engine is built on top of Lucene library[14], and it is famous for providing simple search engine REST APIs, lightning-fast search, scalability, and fine-tuned relevancy. **Fluentd** is a streaming data collector that introduces logging on a unified layer. Fluentd allows data collection, transformation, and ingestion into data sinks. We use Fluentd as a unified logging solution to tail container logs and deliver them to our Elasticsearch cluster. Finally, **Kibana** is a web application commonly combined with Elasticsearch as a data analytics platform. Kibana enhances data querying, visualization, and navigation into Elasticsearch.

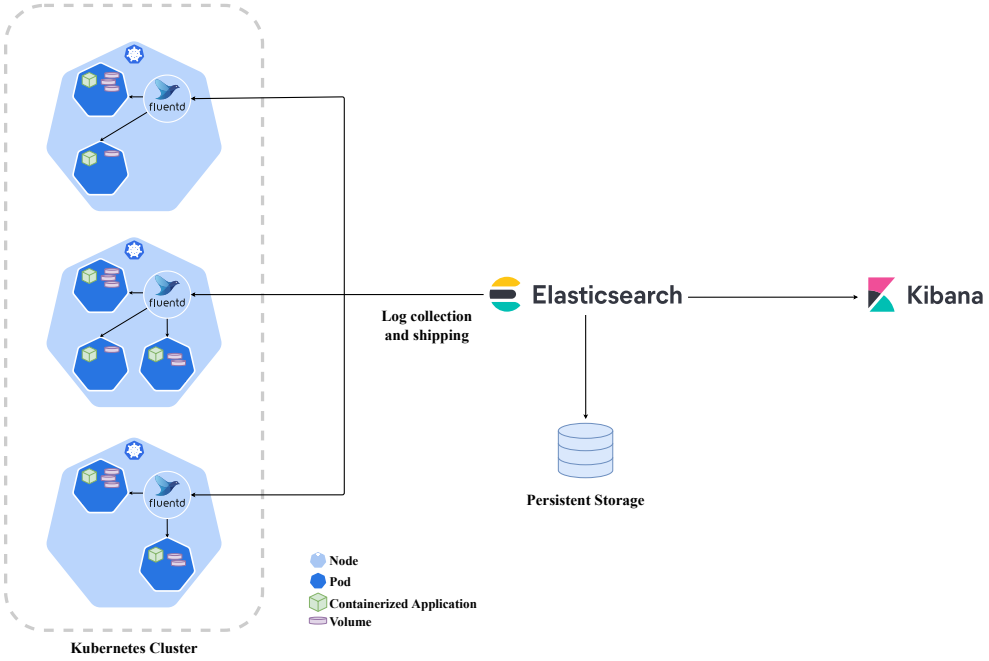


Figure 4.2. EFK Stack

4.1.3 Proposed solution

To tackle the challenges related to observability and log collection in K8S, we propose a solution based on Istio service mesh and EFK stack. We leverage Istio to extend K8S and establish a programmable, application-aware network using Envoy as a sidecar proxy deployed alongside each microservice in the Pods.

The process begins by deploying Istio into our cluster and activating the

[14] https://lucene.apache.org/core/

injection function that enables the deployment of Envoy in the Pods. To gain more insights, we configure Istio/Envoy filters to log incoming network requests to the `stdout` and `stderr` streams. The second step of our process is the deployment of an EFK stack into the cluster according to three main phases. In phase one, Fluentd is responsible for collecting all the logs from the pods, specifically those produced from Envoy's sidecars. Recall that Envoy behaves like a network proxy, and all communications to and from the Pod must go through it. During phase two, Fluentd performs data wrangling (transforming raw data into ingestable data) and starts log ingestion into Elastisearch. In phase three, Elasticsearch performs indexing and data optimization to provide full-text search on collected logs. In the last part of the process, Kibana allows performing queries based on the information we want to extract from the logs. Figure 4.2 illustrates the architecture of the proposed solution.

It is worth noting that neither Istio nor Envoy provide capabilities to log complete HTTP communication. However, it is possible to obtain the same by leveraging Istio log formatting and on how the build-in version of Istio's Envoy handles log formatting policies. This solution will be open-sourced in a second phase.

4.2 Log-Key Sequence abstraction

The basic format of web requests contains a finite number of entries. Classical entries are HTTP methods, hosts, and response codes. Thus we can abstract single web requests to log keys and model those as discrete sequences over time. In other words, we sort classic web requests based on timestamps and process them. Because the number of HTTP methods, microservice and response codes is bounded, we can define a set of keys $\mathcal{K}$ and $|\mathcal{K}| \leq |\mathcal{M}| \cdot |\mathcal{S}| \cdot |\mathcal{T}| \leq \max(|\mathcal{M}|, |\mathcal{S}|, |\mathcal{T}|)^3$ where $\mathcal{M}$ are the methods and m_i a HTTP method (e.g., `GET` or `POST`), $\mathcal{S}$ the microservices and s_i a microservice name (e.g., `microservice-1`), $\mathcal{R}$ the response codes and r_i a response code (e.g,. `200` or `404`). Each k_i represents a Log-Key entry in $\mathcal{K}$, and with this simple abstraction, we can define a injective surjective function that maps log entries to integers such that $f(m_i, s_i, r_i) = k_i$. Table 4.1 illustrates the process of Log-Keys creation.

Table 4.1. Log-Key creation example with methods, microservice names and response codes.

Time	Web Request	Log-Key
t_1	GET /example.html microservice-1 200	1
t_2	POST /test microservice-2 200	2
t_3	PUT /test microservice-1 200	3
t_4	GET /example.html microservice-1 200	1
...	...	...

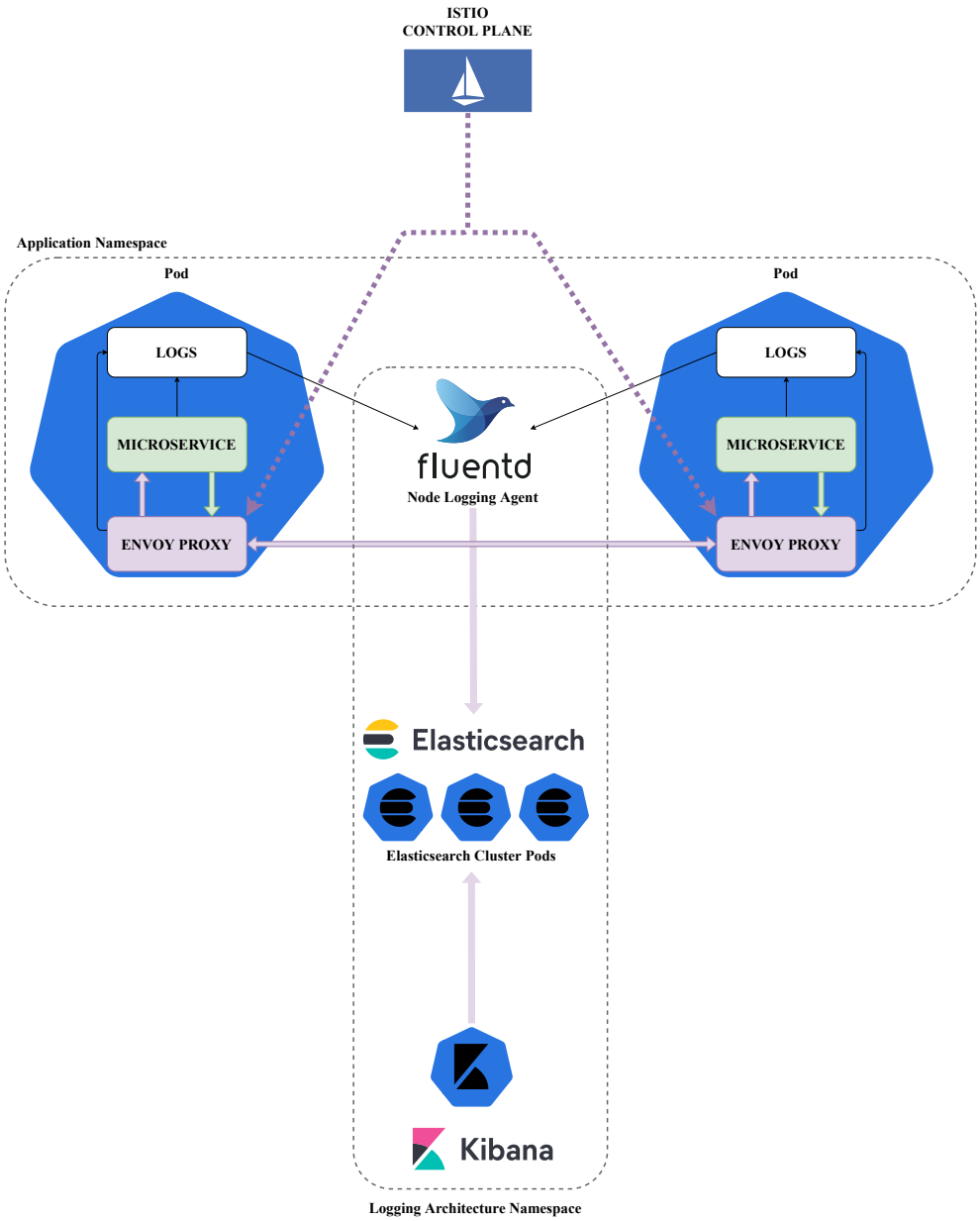


Figure 4.3. Proposed architecture for web log collection

4.3 Long Short-Term Memory for Log-Key Sequences Anomaly Detection

Neural Networks (NNs) are computing systems inspired by the biological neural networks that constitute animal brains. The goal of NNs is to simulate the human brain and help computer programs recognize patterns and solve artificial intelligence (AI) problems [49, 44]. NNs are composed of layers: an input layer, hidden layers, and an output layer. Each layer is composed of neurons. Neurons are the fundamental component of NNs. They are connected to other neurons with weighted edges, and each edge can transmit some information. The information flow between neurons and layers gives NNs memory, since prior inputs

affect current input and output. Standard NNs (one single input, hidden and output layers) cannot capture sequential information in the input data (Figure 4.4). Therefore, their ability to perform well with sequential data — such as those in machine translation and speech recognition — is minimal.

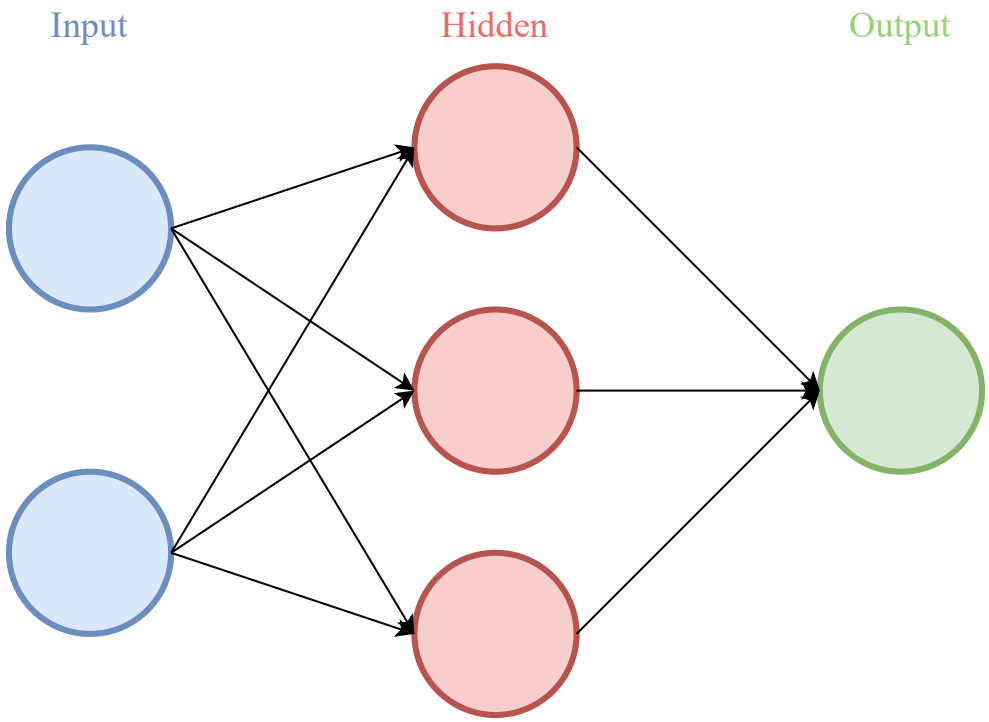


Figure 4.4. High-level view of a Neural Network.

Recurrent Neural Networks (RNNs) are a type of NNs with a self-loop in the connections on neurons in the hidden layers. Their architecture offers an improved ability to learn dependencies in sequential data, which is a challenging and critical problem. Bengio et al. [8] define three requirements for an RNN to learn long-term dependencies: storing information for a specific time; resistance to noise in the input data; and the ability of the system to have trainable parameters. Addressing those requirements introduces a problem known as *vanishing gradient and exploding gradient*. In other words, classic RNNs suffer from the impact of a given input on the hidden layers and, therefore, the output either decays or amplifies [23]. Figure 4.5 illustrates the architecture of a standard RNN and relative unfolding. Unfolding shows how the node's self-loop impacts the output based on current and previous inputs.

Long Short-Term Memory (LSTM) is a specific type of Recurrent Neural Network (RNN) able to learn ordered dependencies in sequence prediction problems. The success of LSTMs is being the first implemented RNNs addressing

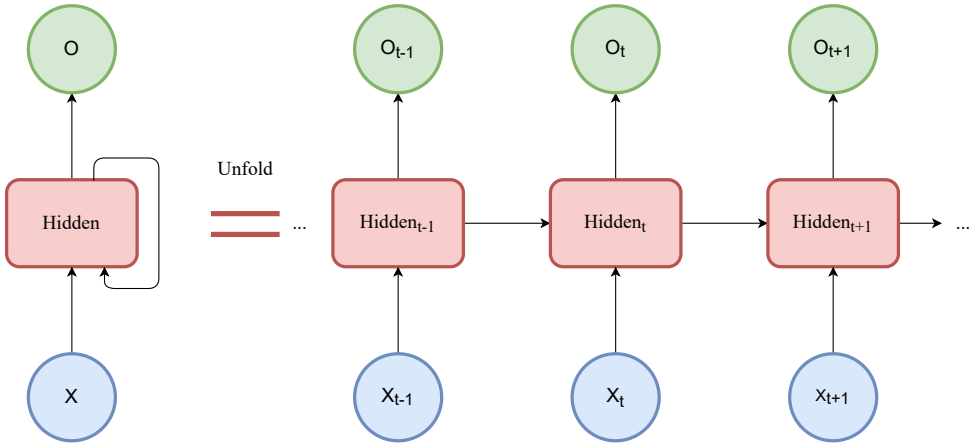


Figure 4.5. High level view of a Recurrent Neural Networks and unfolding representation.

the requirements defined by Bengio et al. Moreover, LSTMs are very different from standard RNNs. LSTMs, like RNNs, are composed of three layers: one input layer, one hidden layer, and one output layer. The hidden layer contains cells and corresponding gate units. Cells are the fundamental units of LSTMs and act as a transportation path for information to the sequence chain. The cells are designed to act as memory, and the cell state can carry information during the processing of a sequence. During the processing of sequences, the information is added or removed from the cells by the gates. Gates are different networks inside a cell and decide which information is allowed in the cell state. LSTM can learn dependencies in sequential data and where the first architecture that addressed the *vanishing gradients and exploding gradient problem* (VEGP) [20]. LSTM addresses the problem with the use of the forget gates by deciding which information should not be forgotten or allowed in the cell's state. This approach makes LSTM resistant to the VEGP. However, both phenomena are still mathematically possible [24]. Figure 4.6 illustrates a standard LSTM cell.

4.3.1 Proposed solution

The Log-Key creation method proposed in Section in 4.2 allows to handle HTTP requests as a sequence. The intuition is that communication between microservices must have some order and rules. In some sense, communication between microservices can be modeled as a language. Inspired by the work in [15], we leverage LSTMs' ability to learn order dependence in sequence prediction problems.

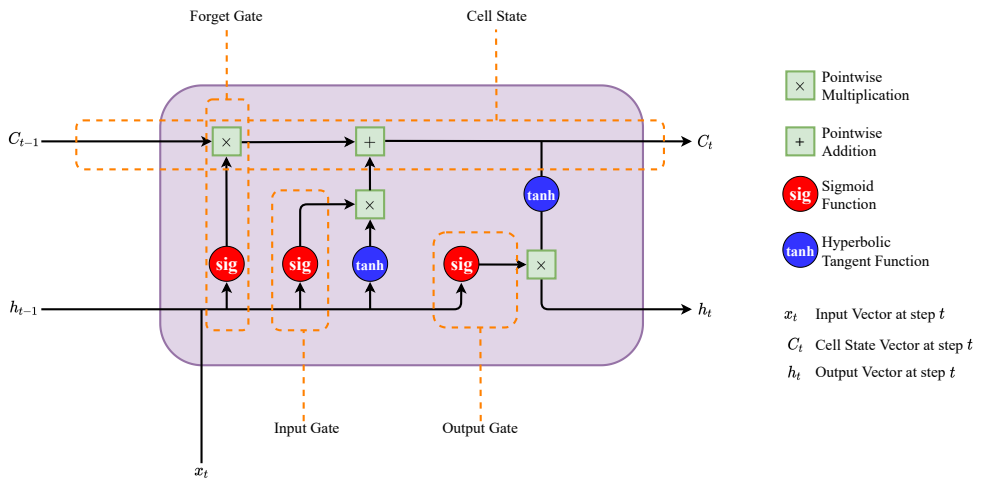


Figure 4.6. High level view of a LSTM Cell.

4.3.1.1 Architecture

The proposed architecture has two main parts: the Log-Key creation and the Log-Key anomaly detection model based on LSTM (Figure 4.7).

4.3.1.2 Training stage

Let w be a window of size l in the sequence and s_i a Log-Key value in K. Clearly, s_i could be any value in K. Moreover, let q be the Log-Key yet to appear. Then, the training input for the model is a window $w = \{s_{q-l}, \ldots, s_{q-2}, s_{q-1}\}$. The training output is a model of the conditional probabilities $Pr[s_q = k_i|w]$. For instance, given the sequence $\{k_{27}, k_{18}, k_{11}, k_{26}, k_{15}, k_{26}\}$ and $l = 3$, we train the model with inputs $\{k_{27}, k_{18}, k_{11} \rightarrow k_{26}\}$, $\{k_{18}, k_{11}, k_{26} \rightarrow k_{15}\}$, $\{k_{11}, k_{26}, k_{15} \rightarrow k_{26}\}$.

4.3.1.3 Detection stage

To test a new incoming Log-Key, we input to the model the previous recent Log-Keys. Let z_q be the incoming Log-Key. The input will be $w = \{z_q, .., z_{q-2}, z_{q-1}\}$ and the output a normalized probability distribution $Pr[z_q|w] = \{k_1 : p_1, k_2 : p_2, ..., k_n : p_n\}$ describing the probability for each Log-Key in K to appear as the next Log-Key value given the history. Finally, probabilities are sorted, and the incoming Log-Key is flagged as anomaly if it is not found in the top g candidates [15].

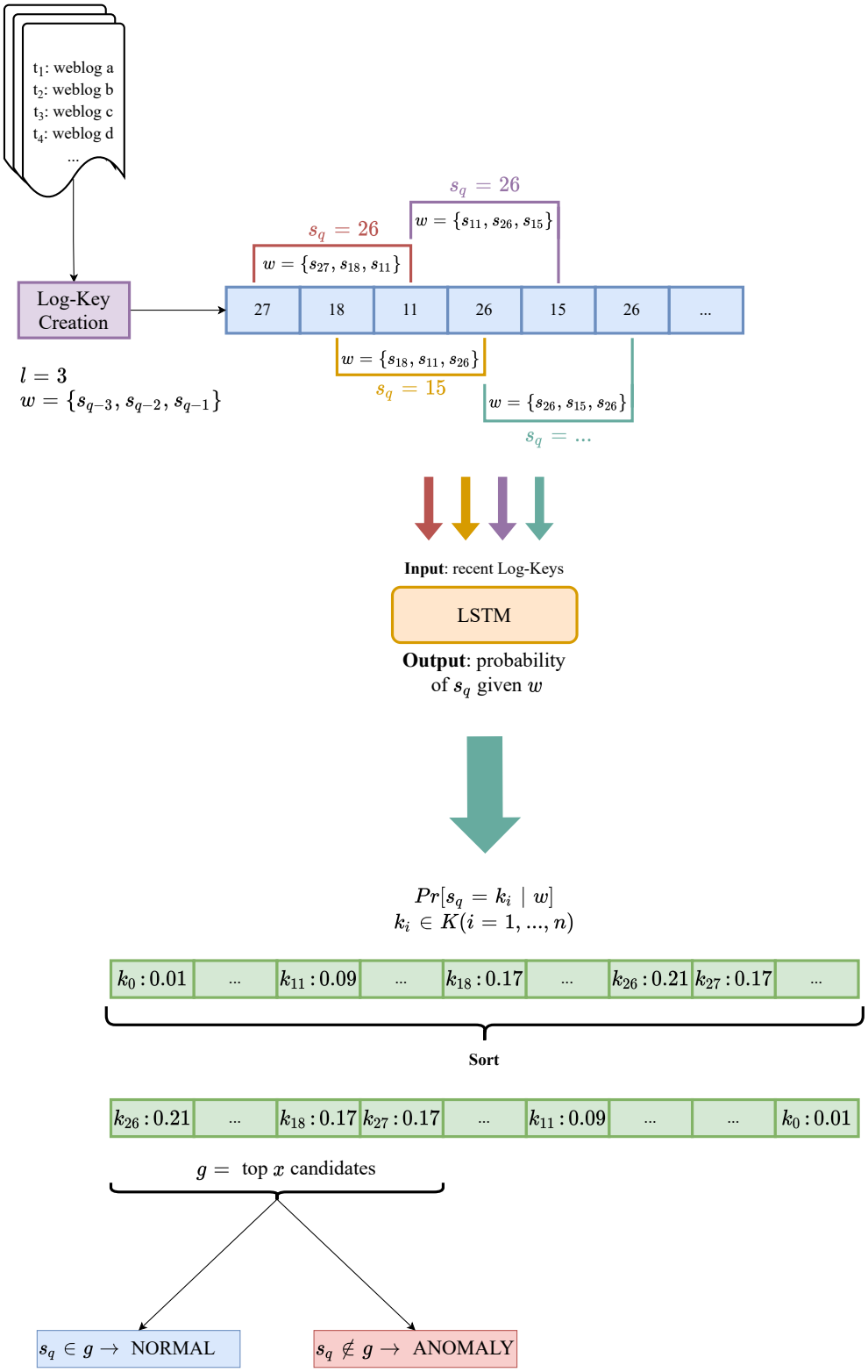


Figure 4.7. Log-Key Anomaly Detection process.

Chapter 5

Web Request Anomaly Detection

This chapter overviews how standard web logs are clustered based on services and REST APIs. Next, it describes the features selection process of HTTP requests. Finally, it introduces our proposed solution for AD with Autoencoders.

5.1 REST APIs extraction and clustering

Performing anomaly detection of web-based attacks is challenging given the distributed and loosely-coupled architecture of microservices. Moreover, automatic scaling is commonly used in cloud environments for performance reasons. The same service can be simultaneously deployed in multiple containers in different clusters or locations. Furthermore, inter-service communication can be efficiently achieved by leveraging RESTful APIs. To efficiently model and distinguish regular requests from abnormal ones, it is critical to model these requests based on specific target service APIs.

Clustering requests based on APIs becomes a challenging problem under the assumptions that APIs are not documented and source code is not available. We experimented with approaches similar to [32, 38] for automatically creating Regular Expressions (REs) for APIs extraction, but with inconclusive results in the clustering phase due to REs ambiguities (e.g., API endpoint matched by multiple REs). Furthermore, Bartoli et al. [7] propose a method based on genetic programming to learn REs from example strings and conduct a large-scale experiment comparing their solutions to user's solutions. The findings are remarkable and show that the quality of automatically-constructed solutions is similar to those constructed by the most skilled group of users. On the other hand, the time for automatic construction was similar to the time required by human users.

The log collection method proposed in Section 4.1 allows clustering requests based on target services. The log contains the path, and the path points to resources provided by the service. In RESTful APIs, the path combines strings

separated by a forward slash (/) symbol. We analyze paths, infer regular expressions for path matching, and perform clustering of web requests.

Table 5.1. Example of Regular Expression matching.

	`^\/cart\/[\w-]*$`	`^\/check\/[\w-]*$`	`^\/order\/[\w-]*$`
`/cart/example`	✓		
`/cart/test-01`	✓		
`/check/example`		✓	
`/order/example`			✓
`/order/test-01`			✓

For instance, Table 5.1 shows five paths and three REs. Each RE represents an endpoint in the service. The first RE matches the first two paths, the second RE matches the third path, and the last RE matches the third and fourth. Finally, Figure 5.1 illustrates the process of clustering web requests based on service and endpoint.

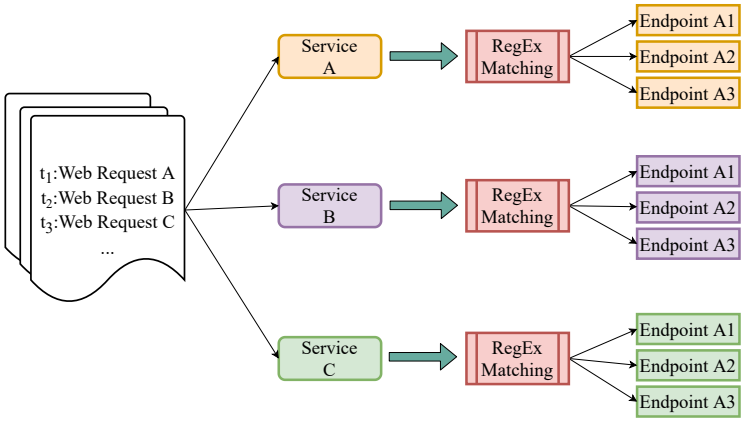


Figure 5.1. Illustration of endpoint matching based on Regular Expressions.

5.2 Web Request Features

The Hypertext Transfer Protocol HTTP is an application-level protocol for information systems. HTTP is generic and stateless, therefore allows systems to be built independently of the data being transferred, facilitating communication and data exchanges. In the client-server computing model, HTTP behaves as a request-response protocol; the client may be the web browser and the server an application. The client initiates a connection by submitting an HTTP request to the server. On the other hand, the server answers the request by providing desired file resources or performing other actions on behalf of the client [18]. We perform analysis on such requests and response logs collected with the methods described in Section 4.1. Furthermore, the analysis focuses on web applications developed with microservice architecture communicating through restful APIs

and JavaScript Object Notation (JSON)[15]. Following the analysis of the related work in anomaly detection of web-based attacks (Section 3.3), we selected features that may be relevant to identify anomalies (Table 5.2).

Table 5.2. Selected features for Web Request modelling

Feature Name
Request Method
Number of bytes in the request received by the server
Number of bytes in the request sent by the client
Request Path length
Number of parameters in the JSON body
Number of special characters in the JSON body
Lowest byte value in the request body
Highest byte value in the request body

5.3 Autoencoders

An autoencoder is a particular type of neural network trained to imitate its input to its output. From an abstract level, autoencoders can be seen as a form of lossy compression, which enables the reconstruction of an approximated version of the original data. Autoencoders are composed of two parts: an encoder and a decoder. In its simplest form, an encoder has a function ϕ transforming the input data $\mathcal{X}$ into a latent representation $\mathcal{F}$ (also referred to as code or latent variables) Eq. (5.1). The decoder ψ transforms the latent representation $\mathcal{F}$ into the output $\mathcal{X}$ Eq. (5.2). We choose ϕ and ψ such that the difference between the input and the output is minimized Eq. (5.3). Thus, we recreate the original input following a generalized non-linear compression.

$$\phi : \mathcal{X} \mapsto \mathcal{F} \tag{5.1}$$

$$\psi : \mathcal{F} \mapsto \mathcal{X} \tag{5.2}$$

$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \|\mathcal{X} - (\psi \circ \phi)\mathcal{X}\|^2 \tag{5.3}$$

Let d be the number of nodes in the input and output layer, and p the number of nodes in the hidden layer h Eqs. (5.4) and (5.5). Then, the encoding stage takes as input x and maps it into h Eq. (5.6). Where h is the latent representation, σ an activation function, W a weight matrix and b the bias vector updated during training through backpropagation [22]. The decoding stage takes as input h and maps it to a reconstruction x' Eq. (5.6) and the output of the decoder is X' Eq. (5.9). Finally, autoencoders are trained to minimize the reconstruction errors such as

[15]However, it is straightforward to consider other data-interchange formats such as *xml*, *x-wbe+xml*, *x-www-form-urlencoded* or *form-data*

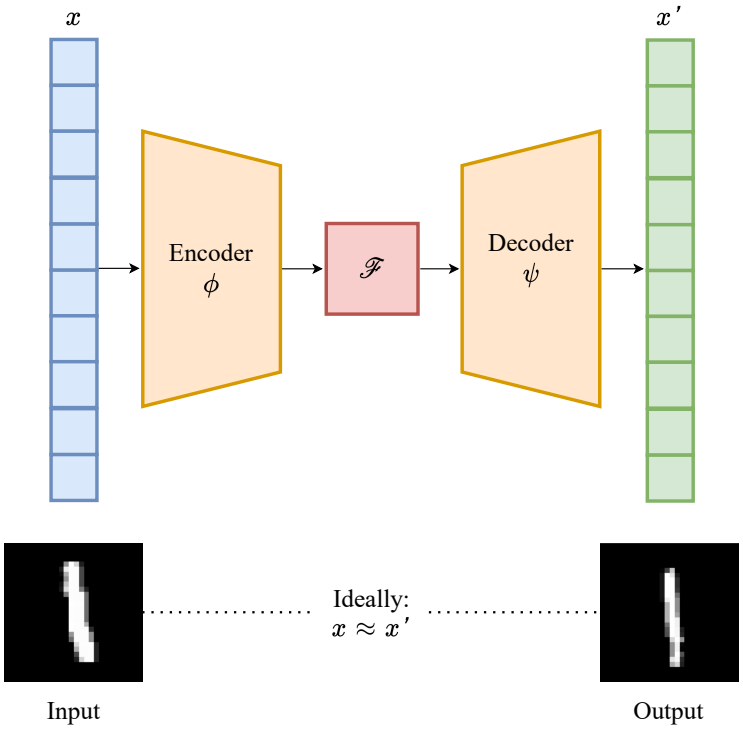


Figure 5.2. Architecture of a basic Autoencoder.

squared errors, Eq. (5.7) between input and output $\mathcal{L}(x, x')$.

$$\mathbf{x} \in \mathbb{R}^d = X \tag{5.4}$$

$$\mathbf{h} \in \mathbb{R}^p = F \tag{5.5}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{5.6}$$

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{h} + \mathbf{b}') \tag{5.7}$$

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2 \tag{5.8}$$

$$\mathbf{x}' \in \mathbb{R}^d = X' \tag{5.9}$$

5.4 Proposed solution

We propose a solution for web requests anomaly detection composed of three main phases:

(A) Parsing and REs matching.

(B) Feature computing.

(C) Anomaly detection with Autoencoder models.

Phase **(A)** begins by parsing HTTP requests from web log entries. Next, web requests are clustered based on target service and matched against a list of REs representing single endpoints in services (Figure 5.1). Phase **(B)** extracts the

target data from web requests and computes the features described in Section 5.2. Finally, phase **(C)** performs the actual anomaly detection and has two modes: training and detection.

5.4.1 Training:

We leverage the web request modeling in Section 5.2 and the Autoencoder model proposed above for anomaly detection of web-based attacks. The intuition is that an Autoencoder model trained with sufficient normal web requests has the ability to imitate its input to its input with minimal reconstruction error. In other words, we expect the model to have a high reconstruction error if the input is an anomalous web request. During training, the inputs to the autoencoder model are the web request features computed in phase **(B)**. The model is trained to minimize reconstruction error Eq. (5.8). Note that each service has multiple endpoints, and each endpoint has a specialized anomaly detection model. In other words, there are as many autoencoder models as the number of endpoints in the service. Thus, each model is highly specialized in reconstructing requests for a specific endpoint since requests are similar. At the end of the training, a threshold value τ based on the reconstruction error is chosen.

5.4.2 Detection:

During detection, the request features are given as input to the model. If the reconstruction error is higher than the threshold value τ chosen during training, then the web request is labeled as anomalous.

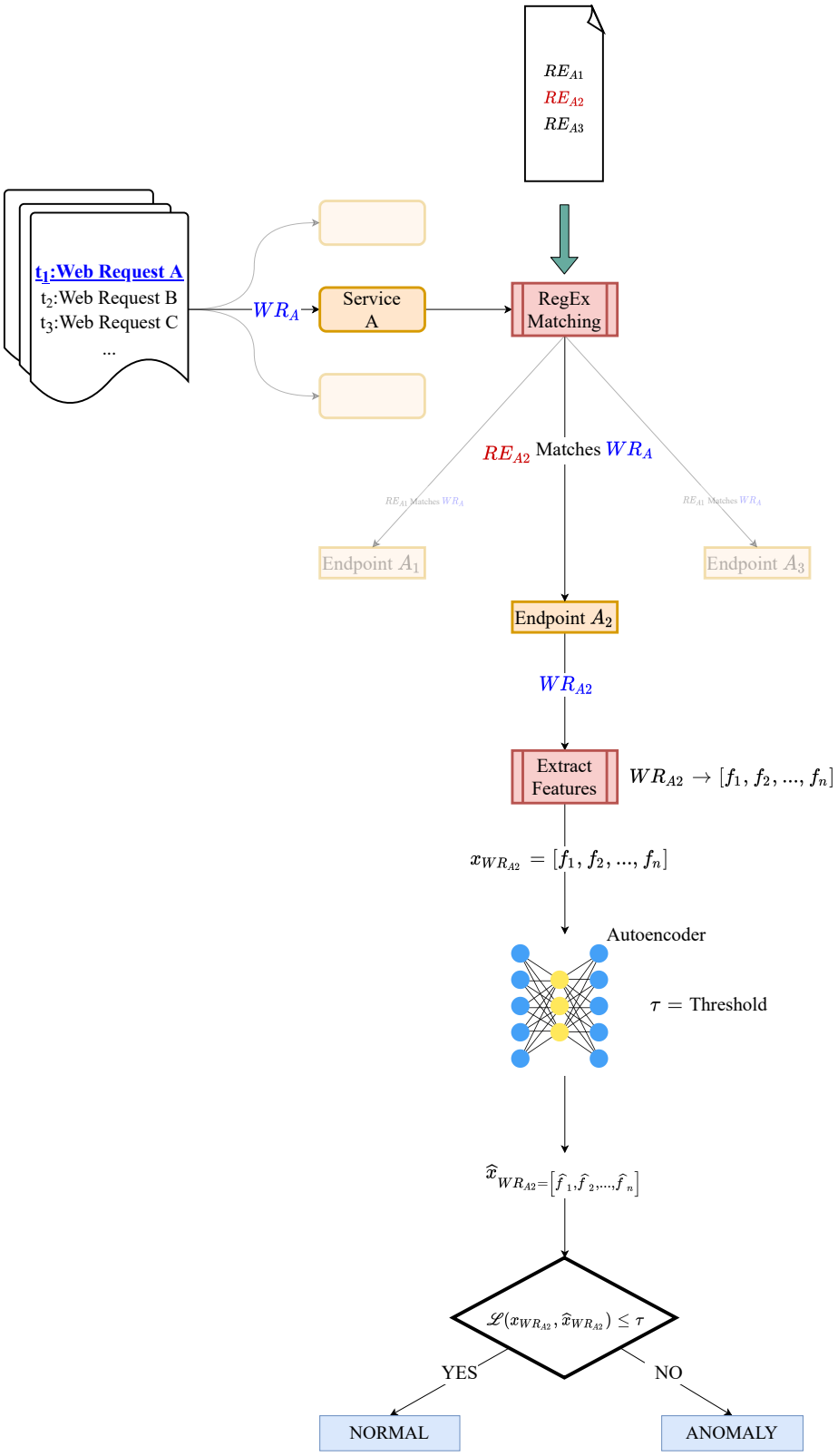


Figure 5.3. Architecture of web request anomaly detection.

Chapter 6

Evaluation

This chapter includes an overview of the experimental setup and methodology.

6.1 Setup and Methodology

6.1.1 Reference Application

Stan's Robot Shop[16] (SRS) is a sample microservice application for learning containerized application orchestration and monitoring techniques. The application uses different technologies and services that resemble an e-commerce web application developed with microservices architecture. We chose SRS since it is one of the few continuously maintained open-source projects that meet microservice architecture requirements.

6.1.2 Methodology

We deployed SRS in a local K8S engine with minikube[17] alongside our proposed method for log collection described in Section 4.1. We divide the creation of the dataset into two phases. In the first phase, we simulate the usage of the application as normal users would, including activities such as casual browsing, user creation, product review, product payment, and product shipping. In the second phase, we perform web attacks such as cross-site scripting, directory traversal, request method tampering, and parameter tampering.

Table 6.1. Collected datasets and anomalies.

Dataset	**Number of logs**	**Anomalies**				
		XSS	Directory Traversal	Method Tampering	Parameter Tampering	Total Anomalies
SRS Dataset	11,220	-	-	-	-	0
SRS Dataset (With Anomalies)	9,923	45	24	18	43	130

The creation of the dataset with anomalies is time-consuming, and performing web attacks requires specific skills. We simulated the attacks by intercepting

[16] https://github.com/instana/robot-shop/
[17] https://minikube.sigs.k8s.io/

web requests on the client-side with BurpSuite, one of the most widely used web application security testing software[18]. The results are two datasets illustrated in Table 6.1. We analyze the performance of the models by using standard metrics such as the number of false positives (FP) and false negatives (FN). Additionally, we compute the Precision=$\frac{true\ postive}{true\ positve+false\ positive}$, Recall=$\frac{true\ positive}{true\ positive+false\ negative}$, and the F-measure=$\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$ (also called F1 Score or harmonic mean).

6.2 Preliminary Analysis

This section studies the impact of parameter tuning on the proposed models and evaluates the performance of our solution on the collected dataset.

6.2.1 Log-Key

The Log-Key Anomaly detection model requires training to detect anomalies. We use the SRS dataset without anomalies to train the proposed LSTM based model and the SRS dataset with anomalies for evaluation. The training dataset contains 140 Log-Keys. If we consider all possible combinations of services, methods, and response codes in the application, the total number of possible Log-Keys is 4221. On the other hand, the SRS dataset with anomalies contains 294 Log-Keys (which anticipates unseen events in the evaluation dataset compared to the training dataset).

The fix parameters, referred to as default values, for the LSTM model are $h = 2$, $\alpha = 128$, $l = 10$, $g = 30$. Recall that h is the number of layers in the LSTM model, α is the number of cells in each layer, l is the length of a window of keys, and g are the top predictions. Given the structure of the proposed solution, we consider an anomaly as detected if our model detects an incoming Log-Key as an anomaly or if any previous l keys are anomalous. That is, we consider an anomaly as detected if it is found in an anomalous Log-Key context of length l.

We study the performance impact of parameters g, l, α, and h. To perform the analysis, we iterate through different parameter values while keeping default values for the others. In Figure 6.1, we compute the cumulative probability of top g keys predictions, and – as expected – we see that the cumulative probability constantly increases with the number of top g predictions. With $g = 23$, the cumulative probability is 99.7% and slowly reaches a plateau 99.8% towards $g = 30$. In other words, with $g = 30$ top predictions, we expect to correctly predict if an incoming Log-Key is anomalous 99.8% of the time. This is not completely correct since the Log-Key Anomaly Detection Model has the ability to detect anomalies in

[18] https://portswigger.net/

the workflow of an application but not attacks that are not interfering with it. For instance, a successful XXS payload will not change the workflow of an application. On the other hand, an XSS payload that crashes a service changes the workflow and is (correctly) detected as an anomaly.

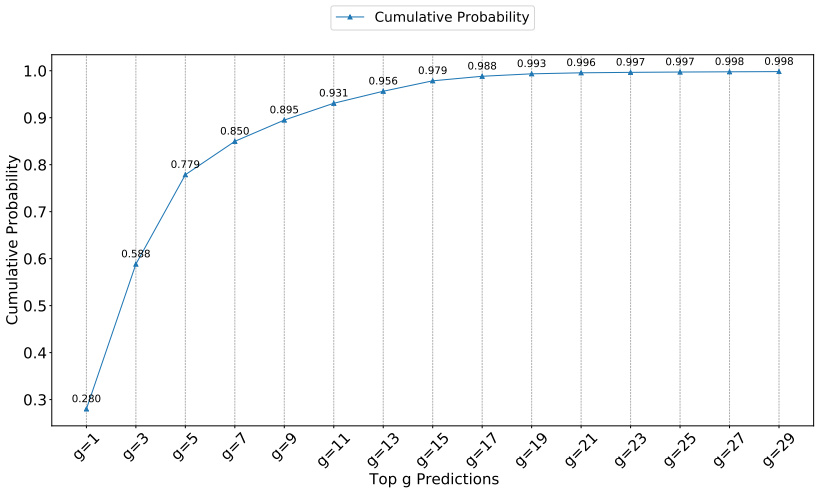


Figure 6.1. Cumulative Probabilities of top g predictions.

The length of the window sequence l has a profound impact on the model performance. Figure 6.2 illustrates how the model evaluation measures varies with l. F-measure, recall, and precision reach peak values with $l = 13$ and they rapidly decrease afterward. Choosing the best value for l is challenging, and it highly depends on the application architecture and inter-service communication patterns. Based on our observation, the l value for an application with hundreds of inter-service communications will be higher than an application with a dozen inter-service communications.

Choosing the correct number of cells α for each layer is another challenging task. If α is too small, the model could underfit during training. On the other hand, if α is too big, the chances to overfit during training are higher. In both cases, the result is poor performance during evaluation. Figure 6.3 clearly illustrates this behaviour. In the first case, with $\alpha \leq 32$, the result is a lightly underfit model with an F-measure approaching 0.50. In the other case, with $\alpha \geq 256$, the result is an overfitting model with performances steadily decreasing.

As for the cells' number, choosing the correct value for the hidden layers h in a deep neural network is challenging, and it is common practice to primarily rely on trial and error and intuitions. Figure 6.4 shows the impact of the number of layers

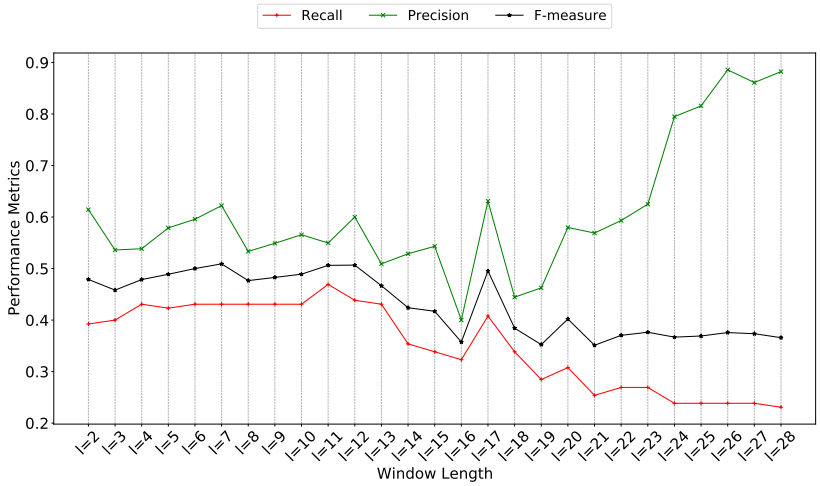


Figure 6.2. Log-Key model's performance by increasing window size l.

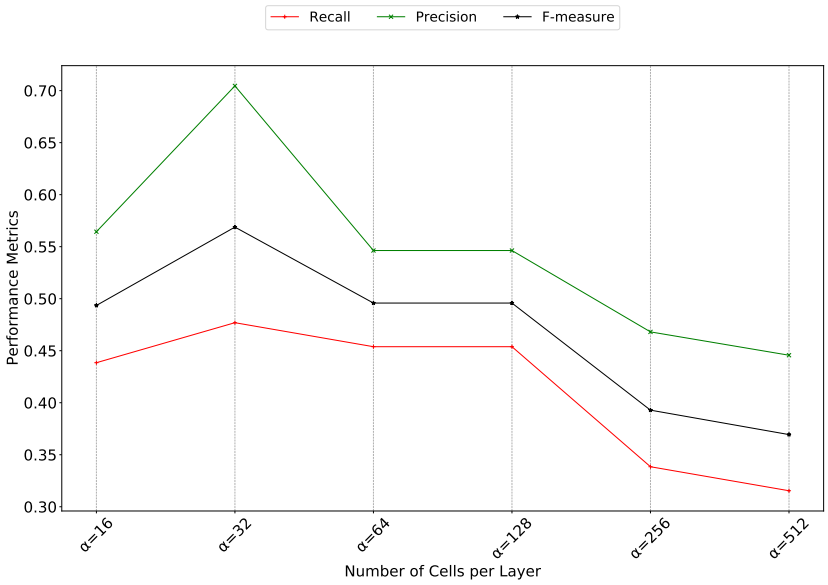


Figure 6.3. Log-Key model's performance by increasing the number of cells α.

on the model performance. We can observe that with few layers $1 \leq h \leq 4$, the result is an underfit model and poor performances. On the other hand, with $h \geq 7$, the model reaches an F-measure of 0.57 but does not overfit. A number of layers higher than needed leads to higher computational overhead during training and evaluation; it is then essential to choose l such that performance is maximized and computational costs are minimized.

6.2.2 Autoencoder

As the Log-Key Sequences Anomaly Detection, the Web Request Anomaly Detection with Autoencodrs requires training. Similar to the previous chapters, we use the SRS dataset without anomalies for training and the SRS dataset with

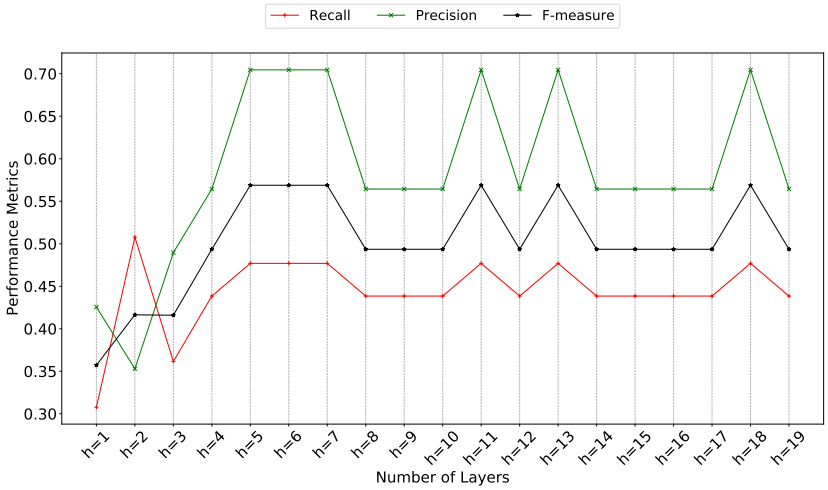


Figure 6.4. Log-Key model's performance by increasing the number of layers h.

anomalies for evaluation.

We evaluate our web requests AD with Autoencoders using the same metrics proposed in the methodology section. The default values for the Autoencoders are encoding layers $el = 3$, decoding layers $dl = 3$. Cells in the encoding layers are $el_1 = 8, el_2 = 4, el_3 = 2$ and in the decoding layer $el_1 = 2, el_2 = 4, el_3 = 8$. Finally, the threshold value τ heavily depends on the training dataset. The threshold τ is set to label as anomalies web logs with reconstruction error above the 0.999 percentile compared to the errors computed during training.

Figure 6.5 illustrates the importance of carefully setting the threshold value τ. In the proposed model, performance suffer with $\tau \leq 0.989$ but increase towards an harmonic mean of 0.80 for $\tau = 0.993$. Moreover, the overall performance slightly decreases and fewer anomalies are detected with the threshold being too strict.

6.3 Evaluation

The proposed solution is to combine the Log-Key Sequences and the Autoencoders capability for AD. On the one hand, we aim at detecting anomalies in the inter-service communication workflow with the Log-Key Sequences. On the other hand, we aim at detecting anomalies in the features of the web requests (e.g., the malicious payload in the request body) with the Autoencoders. The confusion matrix in Figure 6.6 illustrates the performance of our solution. The high accuracy (99.6%) shows that our model performs very well from an overall perspective. The recall (78.8%) describes the occurrence of false negatives. The precision (91.5%) states the confidence of the model on true positives. While dealing with AD models, the use case is vital for tuning the model. For instance, if the use case is a real-world

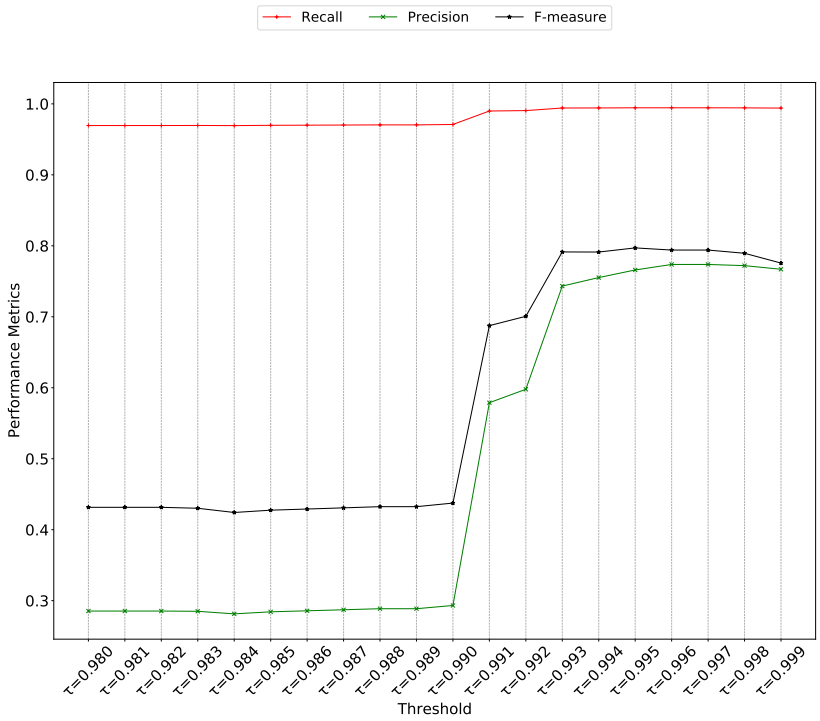


Figure 6.5. Autoencoder's performance by increasing the threshold τ

critical infrastructure, the model must be tuned to achieve a high recall value and avoid false negatives. On the other hand, few false negatives could be acceptable in non-critical systems, but not poor precision since it would lead to multiple false positives and overwhelm system administrators. Moreover, the proposed model detects 100% (45/45) of the XSS anomalies, 83.3% (20/24) of the directory traversal anomalies, 94.4% (17/18) of the method tampering anomalies, and 86.1% (37/43) of the parameter tampering anomalies.

Altogether, our solution obtains an F-measure of 0.847 which — considering limited training dataset and limited parameter tuning — is a remarkable result.

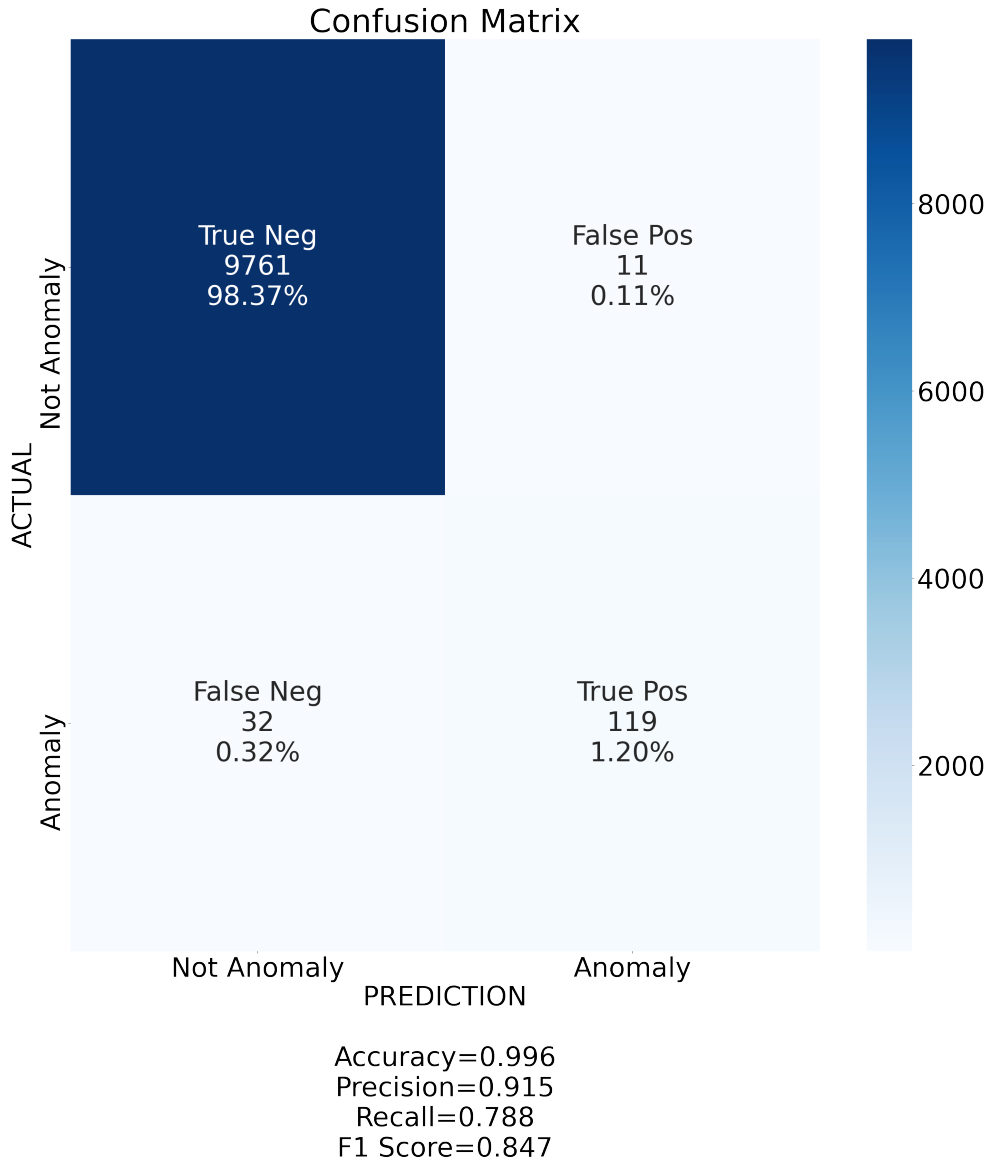


Figure 6.6. Confusion matrix of the proposed solution.

Chapter 7

Conclusion

This thesis explores anomaly detection of web-based attacks in microservices architecture by modeling regular web requests and communication behaviors between microservices in modern web applications. It started by reviewing the concept of anomaly detection in different fields and continued by summarizing the most relevant papers in anomaly detection of web-based attacks. Then, it proposed a web log collection method deployable in K8S that does not require any access to microservices' source code. Next, it proposed a Log-Key Sequences anomaly detection method with the ability to abstract from web logs to Log-Key sequences and performed Anomaly Detection with LSTM on the workflow and the communication patterns between services. Last, it proposed a web log anomaly detection method based on Autoencoders identifying malicious content in web requests. Figure 7.1 summarized our proposed solution.

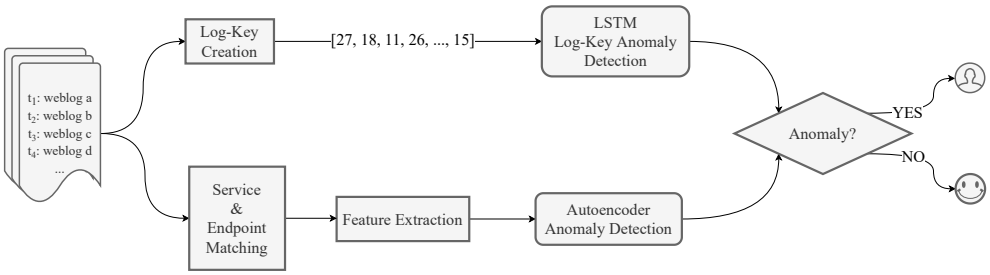


Figure 7.1. Summary of the proposed solution.

The proposed solution approach is cloud-native and easily deployable alongside any web application running in K8S deployment. Moreover, the thesis studied the impact of the models' main parameters on their performance and showed the effectiveness of the proposed solution.

The principal contribution of this thesis is the proposed method for detecting anomalies in the inter-services communication workflow. To the author's best knowledge, the existing literature has not considered AD in the context of microservices communication workflow.

The proposed solution is a first step towards addressing AD of web attacks in microservices. Nevertheless, to make our solution more interesting, two limitations should be addressed in the future. The first one on is to provide the model with online feedback. In this sense, the model should be updated in an online fashion based on human feedback on the model's detected false positives and (when possible) false negatives. The second interesting future work is to infer workflow patterns from Log-Key Sequences and automatically construct rules which can be used to infer inter-service communication policies with a service mesh software.

Bibliography

[1] Charu C Aggarwal. An introduction to outlier analysis. In *Outlier analysis*, pages 1–34. Springer, 2017.

[2] Charu C Aggarwal. Linear models for outlier detection. In *Outlier analysis*, pages 65–109. Springer, 2017.

[3] Mohamed Ahmed. The sidecar pattern, Sep 2019. URL: `https://www.magalix.com/blog/the-sidecar-pattern`.

[4] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md Rafiqul Islam. A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278–288, 2016.

[5] Sara Althubiti, Xiaohong Yuan, and Albert Esterline. Analyzing http requests for web intrusion detection. 2017.

[6] Yuequan Bao, Zhiyi Tang, Hui Li, and Yufeng Zhang. Computer vision and deep learning–based data anomaly detection method for structural health monitoring. *Structural Health Monitoring*, 18(2):401–421, 2019.

[7] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Can a machine replace humans in building regular expressions? a case study. *IEEE Intelligent Systems*, 31(6):15–21, 2016.

[8] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[9] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[10] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26, 2011.

[11] Ricardo JGB Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(1):1–51, 2015.

[12] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[13] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Outlier detection: A survey. *ACM Computing Surveys*, 14:15, 2007.

[14] Sanghyun Cho and Sungdeok Cha. Sad: web session anomaly detection based on parameter estimation. *Computers & Security*, 23(4):312–319, 2004.

[15] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, 2017.

[16] Wen Kai Guo Fan. An adaptive anomaly detection of web-based attacks. In *2012 7th International Conference on Computer Science & Education (ICCSE)*, pages 690–694. IEEE, 2012.

[17] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[18] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1, 1999.

[19] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2):18–28, 2009.

[20] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[21] Carmen Torrano Giménez, Alejandro Pérez Villegas, and Gonzalo Álvarez Marañón. Http data set csic 2010. *Information Security Institute of CSIC (Spanish Research National Council)*, 2010.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 6.5 back-propagation and other differentiation algorithms. *Deep Learning*, pages 200–220, 2016.

[23] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2008.

[24] Roger Grosse. Lecture 15: Exploding and vanishing gradients. *University of Toronto Computer Science*, 2017.

[25] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Dotan Horovits. A practical guide to kubernetes logging, Sep 2020. URL: `https://logz.io/blog/a-practical-guide-to-kubernetes-logging/`.

[28] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 21–26, 2016.

[29] Hanif Jetha. How to set up an elasticsearch, fluentd and kibana (efk) logging stack on kubernetes, Mar 2020. URL: `https://do.co/2SOtZAx`.

[30] Li-Jen Kao and Yo-Ping Huang. Association rules based algorithm for identifying outlier transactions in data stream. In *2012 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 3209–3214. IEEE, 2012.

[31] Edwin M Knorr, Raymond T Ng, and Vladimir Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8(3):237–253, 2000.

[32] Rafał Kozik, Michał Choraś, Rafał Renk, and Witold Hołubowicz. Modelling http requests with regular expressions for detection of cyber attacks targeted at web applications. In *International Joint Conference SOCO'14-CISIS'14-ICEUTE'14*, pages 527–535. Springer, 2014.

[33] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Outlier detection in axis-parallel subspaces of high dimensional data. In *Pacific-asia conference on knowledge discovery and data mining*, pages 831–838. Springer, 2009.

[34] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Outlier detection in arbitrarily oriented subspaces. In *2012 IEEE 12th international conference on data mining*, pages 379–388. IEEE, 2012.

[35] Christopher Kruegel and Giovanni Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261, 2003.

[36] Aleksandar Lazarevic, Levent Ertoz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 SIAM international conference on data mining*, pages 25–36. SIAM, 2003.

[37] Kingsly Leung and Christopher Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, pages 333–342, 2005.

[38] Vladimir Likic. The needleman-wunsch algorithm for sequence alignment. *Lecture given at the 7th Melbourne Bioinformatics Course, Bi021 Molecular Science and Biotechnology Institute, University of Melbourne*, pages 1–46, 2008.

[39] Hai Thanh Nguyen, Carmen Torrano-Gimenez, Gonzalo Alvarez, Slobodan Petrović, and Katrin Franke. Application of the generic feature selection measure in detection of web attacks. In *Computational Intelligence in Security for Information Systems*, pages 25–32. Springer, 2011.

[40] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton van den Hengel. Deep learning for anomaly detection: A review. *arXiv preprint arXiv:2007.02500*, 2020.

[41] Seungyoung Park, Myungjin Kim, and Seokwoo Lee. Anomaly detection for http using convolutional autoencoders. *IEEE Access*, 6:70884–70901, 2018.

[42] Huan-Kai Peng and Radu Marculescu. Multi-scale compositionality: identifying the compositional structures of social dynamics using deep learning. *PloS one*, 10(4):e0118309, 2015.

[43] Truong Son Pham, Tuan Hao Hoang, and Vu Van Canh. Machine learning techniques for web intrusion detection—a comparison. In *2016 Eighth International Conference on Knowledge and Systems Engineering (KSE)*, pages 291–297. IEEE, 2016.

[44] Warren S Sarle. Neural networks and statistical models. 1994.

[45] Ahmed Shawish and Maria Salama. Cloud computing: paradigms and technologies. In *Inter-cooperative collective intelligence: Techniques and applications*, pages 39–67. Springer, 2014.

[46] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6479–6488, 2018.

[47] Johannes Thönes. Microservices. *IEEE software*, 32(1):116–116, 2015.

[48] Deepa Verma, Rakesh Kumar, and Akhilesh Kumar. Survey paper on outlier detection using fuzzy logic based method. *International Journal on Cybernetics &; Informatics (IJCI) 6*, 2017.

[49] Sun-Chong Wang. Artificial neural network. In *Interdisciplinary computing in java programming*, pages 81–100. Springer, 2003.

[50] ZDNet. Microservices worth the hype, 2020. URL: `https://www.zdnet.com/article/survey-microservices-worth-the-hype/`.

[51] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(5):363–387, 2012.

[52] Mikhail Zolotukhin, Timo Hämäläinen, Tero Kokkonen, and Jarmo Siltanen. Analysis of http requests for anomaly detection of web attacks. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pages 406–411. IEEE, 2014.