Security monitoring of microservice-based applications

Security monitoring of microservice-based applications

Wei Zhang
Independent Researcher

ABSTRACT

Microservice-based architecture is a relatively new paradigm of software development that has gained tremendous popularity recently. A microservice-based application comprises multiple relatively small independent functional components (microservices) that interact to accomplish a sophisticated function. Microservices enable agile development and deployment of applications and enhanced scalability and resiliency. However, microservices-based systems have more significant security concerns than traditional monolithic systems. In this article, we discuss the various security loopholes of a microservice-based architecture and how these diverse components increase the attack surface of the aggregate framework. This article also proposes a machine learning (ML) based behavioral analysis framework that analyzes the network traffic and API calls to detect flaws and vulnerabilities in the microservice architecture to overcome these challenges. Prior research has demonstrated the potential of network monitoring to secure microservice-based cloud applications. However, they used hand-designed policies to enforce security compliance. Manually designed policies have their drawbacks. This article discusses these challenges and proposes a novel ML-based pattern recognition to automate the manual definition of policies. ML-based attack detection techniques have achieved state-of-the-art performance in various cybersecurity applications like malware detection and vulnerability detection. However, security monitoring of microservice-based applications is still a developing research field that has not experienced the impact of ML yet. This article proposes implementing supervised ML-based security monitoring of microservice-based applications based on Seeker to detect vulnerability exploits in real-time.

1 INTRODUCTION

Microservices architecture is one of the latest trendsetters in software development. We can broadly define microservices as a service-based architecture composed of a collection of small services. Developers deploy these microservices as independent applications that communicate via application programming interfaces (API) to accomplish sophisticated tasks. Microservices' advantages include enhanced scalability, faster software development and deployment, and easier debugging and testing. Famous examples of microservice-based web platforms include Netflix [2] and eBay [1].

However, the security risks introduced by microservices often overshadow their advantages. The security risks of microservices are manifold. First, the increase in the number of independent components in the network increases the attack surface of the aggregate system. The compromise of any of the microservices will threaten the entire network. Second, the increased number of components in the network leads to an exponential increase in communication between the services, thus making it challenging to monitor the network traffic with an intrusion detection system. Third, the

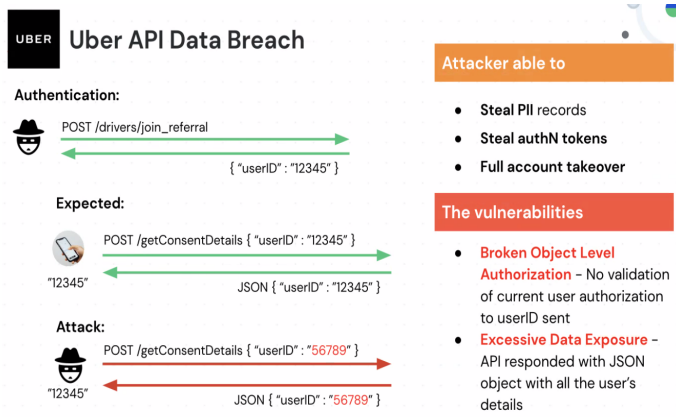

Figure 1: Details of an attack on the microservices-based platform of Uber

diversity of the microservices increases the vulnerability of the aggregate application. The microservices architecture is vulnerable to the traditional web vulnerabilities like SQL injection and cross-site scripting and new classes of vulnerabilities like broken object-level authorization [5]. For example, a recent API-based attack on the Uber application compromised the Uber cab drivers' credentials and personal identifiable information. This vulnerability arose when the developers updated to the latest version of the Uber application. Although the updates enhanced the features of the overall platform, they broke the security compliance of a relatively old feature that was a part of the Uber platform's microservice architecture. This vulnerability allowed hackers to send requests to join Uber as a cab driver and receive the credentials of all the existent cab drivers in their database. We depict the high-level attack flow of this attack in Fig. 1.

Security of microservices-based architecture is a developing field of research [3, 4, 6, 8–10, 18, 19]. Most of the current research has focused on authentication and authorization issues [7, 12, 14]. Unfortunately, there has not been enough research on vulnerability exploit detection on microservice-based software platforms. Torkura et al. [17] discusses a security gateway-based surveillance method to secure cloud-native applications deployed on microservices. Although unique, they rely on manually-defined compliance rules. The manual definition of rules has two significant disadvantages. First, the manual description of rules becomes ineffective and incomplete when the size of the system is large with many microservices. Second, stringent compliance rules may stop benign traffic and hinder the potential of the application. Sun et al. [15] also attempts to address the overarching problem of security monitoring of a microservice-based architecture. They utilize a software-defined network (SDN) to monitor network traffic and enforce compliance rules via the controller of the SDN. However, SDN is a relatively new technology, and most existing services do not employ this technology. Therefore, deploying the proposed technique in currently deployed services will be challenging. Furthermore, they manually

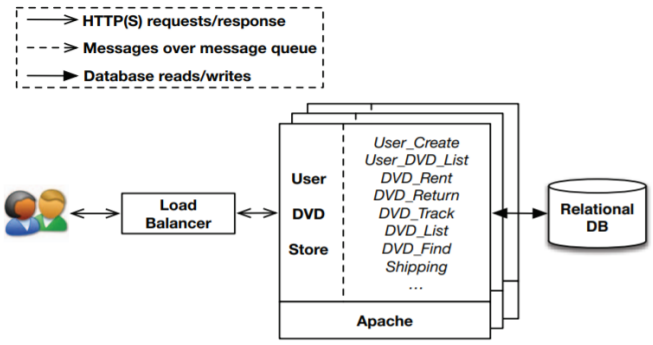

Figure 2: Architecture of an online DVD rental service based on a monolithic architecture [15]

define the compliance rules of the network. We offer a machine learning (ML) approach to identify attack patterns to overcome these drawbacks. We also suggest using distributed tracing instead of SDN to monitor the inter-service communication between microservices. Distributed tracing is relatively easier to deploy than SDN since it can be added to existing framework as a new microservice. However, deploying an SDN requires redesigning the entire application and network architecture. Traceable.ai is a startup that claims to use distributed tracing to perform unsupervised ML to detect anomalies in network traffic. However, unsupervised ML often leads to high false positives, which can be frustrating for the security analyst. We propose a supervised ML technique based on Seeker, which detects and classifies potential exploits and has the potential to explain the reason for its prediction. These details can be highly beneficial and convenient for security analysts, thus making our framework more preferable than existing solutions.

2 BACKGROUND

This section discusses the background of microservices and their security threats.

2.1 Microservices architecture

Software developers have been deploying traditional internet applications as a stand-alone service with a single backend monolithic application handling business logic. For example, we show the backend architecture of an online monolithic DVD rental service in Fig. 2. We observe that the application implements all the business logic of the rental contract in a single executable process. There are multiple instances of this process to handle scalability.

Microservices introduce a new paradigm of software engineering where the business logic is divided into smaller processes. These smaller processes, known as microservices, are deployed independently and are loosely coupled with each other. They communicate with each other through APIs using a lightweight communication protocol. We demonstrate an alternate microservice-based version of the DVD rental service in Fig. 3. In Fig. 3, we see that the business logic of the DVD rental service is segmented into microservices like *User-Create* and *Contract-Update*. This microservice-based architecture enables in overcoming multiple disadvantages of deploying and maintaining a monolithic application.

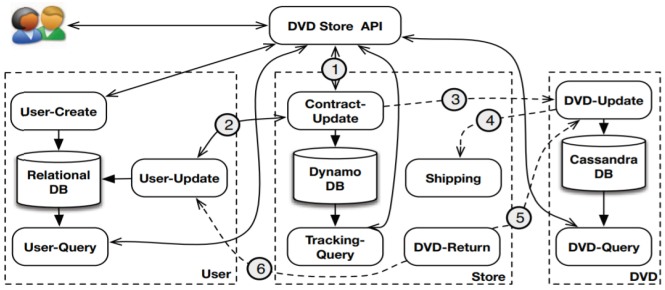

Figure 3: Architecture of an online DVD rental service based on a microservices architecture [15]

The challenges faced in a monolithic application life-cycle are manifold. First, the scaling requirements for different modules may differ. For example, developers may need to scale the *UserCreate* function more frequently than the *Shipping* function. This difference in scaling requirements of services is challenging in a monolithic scenario because the entire application needs to be replicated. However, in microservice-based architectures, the individual microservice can be scaled up or down independently, thus saving time and resources. Next, updating individual components of a monolithic application is complicated because the entire application needs to undergo an end-to-end modification. Since the microservices are independent modules, updating individual microservices does not affect the functionalities of the other microservices. Finally, segregating the DevOps responsibilities also becomes problematic in a monolithic application because all the components are intertwined in a sophisticated fashion. Microservices ameliorate this issue by having dedicated DevOps teams for individual microservices, thus making the life-cycle of the application more agile.

Although microservices enable a variety of advantages throughout the application life-cycle, it introduces a lot of new components and communication channels in the network. This breakdown of a monolithic application into complementary services presents a variety of security threats. First, the introduction of multiple user-facing services expands the attack surface of the aggregate application. The malicious actor has more entry points into the network, thus making it easier for him to compromise the system. Second, the increased network complexity generated by the new components makes it challenging to debug and test the systems. For example, in Fig. 3, updating the *Contract-Update* service not only requires testing the *Contract-Update* module but the entire application at large. Otherwise, the system may fall prey to attacks like the one shown in Fig. 1. Finally, the increased communication on the network makes it challenging to monitor the network traffic.

3 CHALLENGES

Securing microservice-based architectures is a nightmare compared to securing monolithic systems. We demonstrate some of the unique challenges of microservices security below.

- **Larger attack surface:** All communications in monolithic applications occur within a single process. This makes it easy to monitor the activity of the system operations. However, the internal components of a monolithic applications are designed as independent services in the microservice architecture. Thus, the in-process calls among the internal

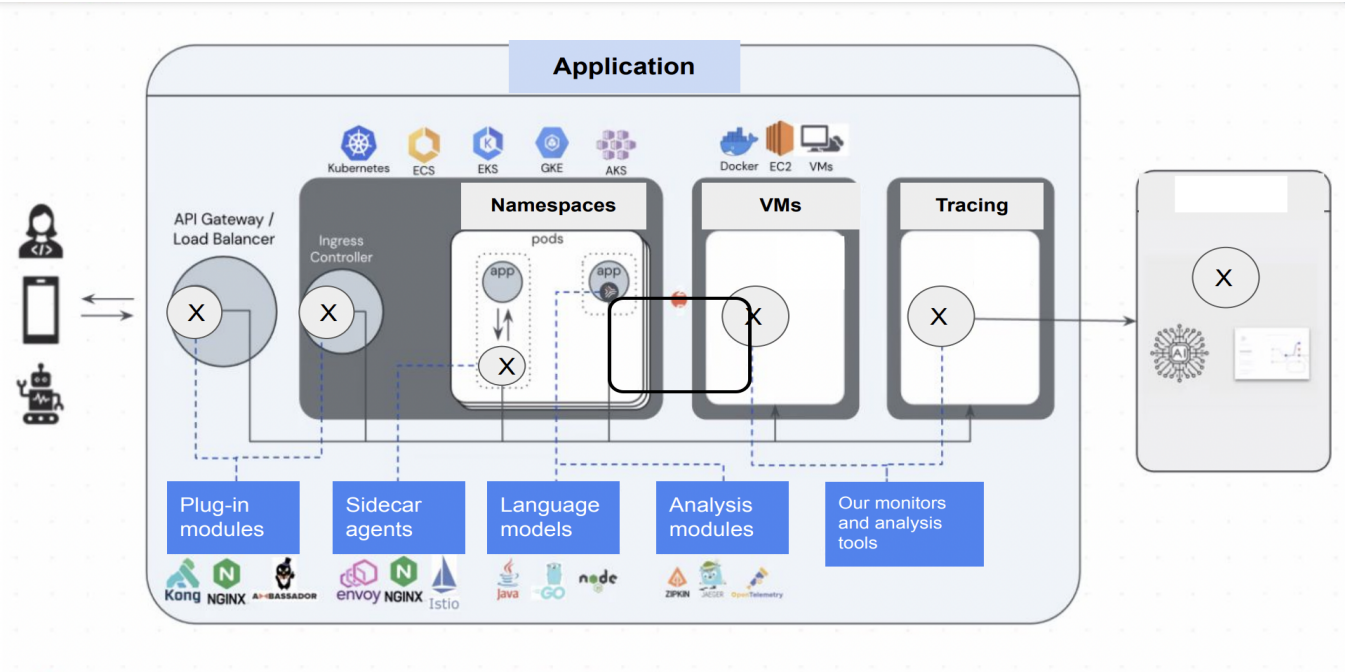

Figure 4: Overview of vulnerability detection and real-time exploit detection with Seeker

components become remote calls. Consequently, this results in more entry points into the system. As the number of entry points in a system increases, the attack surface of the aggregate system broadens. This larger attack surface makes security monitoring of microservices-based architectures challenging.

- **Processes traversing multiple microservices are difficult to trace:** Monolithic applications use logging and tracing of requests to derive metrics of the overall system. For example, the average invalid access requests per second is a metric that enables the system to track a potential DoS attack. Tracing provides a complete overview of the trajectory of the process from the point it enters the system to the point it exits the system. This becomes challenging in a microservice-based deployment because every microservice is an individual system. Moreover, third-party microservice applications further reduces the transparency of the system. Thus, correlating requests among microservices is challenging. Recent research proposes the use of distributed tracing that partially mitigates these challenges.
- **Distributive property of microservices makes sharing user context harder:** In monolithic applications, all internal components share the same web session. This single session conveniences the retrieval of all desired properties of the user. However, this is not possible in a microservice-based system. Since minimal information is shared between microservices, they have explicitly communicate the user context among each other. It is challenging to build trust among these microservices and detect when a user context has been modified.
- **Lack of labeled data:** The distributed traces for normal traffic is abundant. However, obtaining the traces of malicious traffic is difficult [11]. This requires simulating various scenarios in the cloud infrastructure and continuously monitoring the systems. This requires considerable investment and efforts.
- **Long-range data dependencies:** The sequences in malicious network traces may have malicious actions separated by a long series of non-malicious events. Unfortunately, most sequence-based ML models are incapable of handling such long-range dependencies. Furthermore, the models that can effectively capture these dependencies incur a huge computation overhead. Although Seeker addresses these concerns, it needs to be redesigned and retrained for microservices security.
- **Noisy data:** The reaction time of microservices is dependent on the performance of multiple components like network switches, processors, routers, programming languages, and volume of user requests. Therefore, the signal-to-noise ratio of the timing metrics is low. Furthermore, user behavior varies over time. This variation results in multiple modalities in the data. Thus, it is challenging to extract the malicious patterns from underlying data.

4 METHODOLOGY

This section describes the framework for monitoring the security status of a microservice-based application in real-time. First, we show our proposed pipeline at a high level in Fig. 4. The figure shows that we use tracing to monitor the communication over the application network.

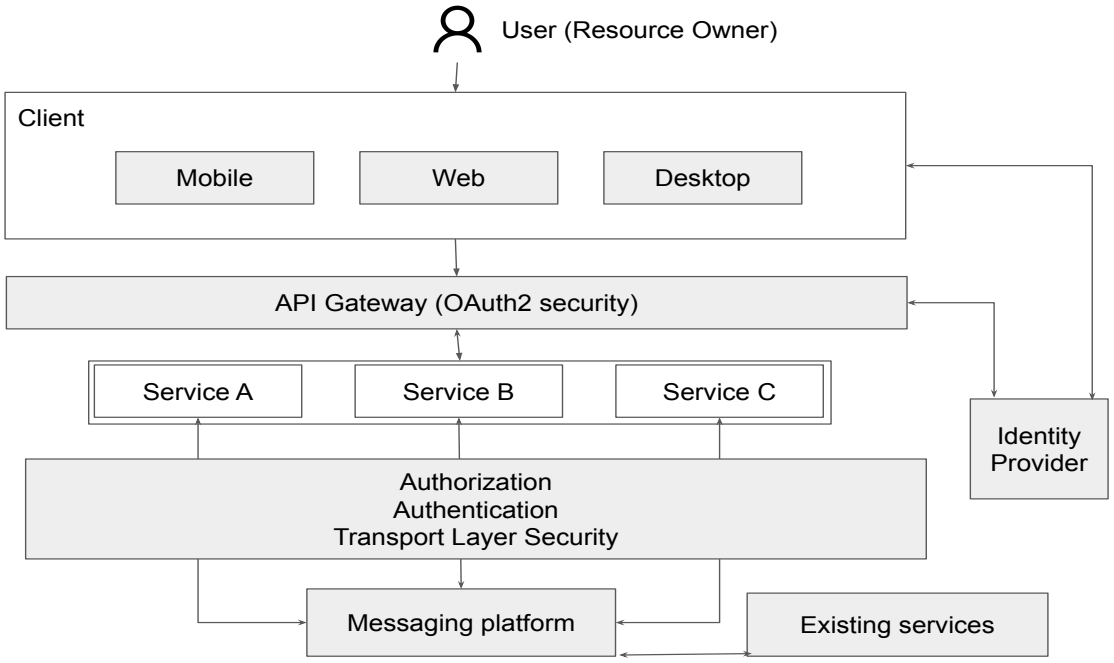

Figure 5: Proposed security architecture for ensuring safe communication between microservices and with external entities

4.1 Proposed Solutions

Now, we describe some of our proposed solutions to tackle the challenges in microservice security mentioned above.

- **API gateway:** We propose the utilization of an API gateway to monitor the traffic between the system and external entities. This is a critical point in the security chain because most of the adversaries are external entities who attempt to gain access privileges to the system. The API gateway will consist of a proxy service that will represent the microservice and implement its security policies.
  A drawback of this approach is that it introduces a monolithic component to the overall architecture. We introduce an identity provider (IDP) which serves as an authorization server microservice. The IDP validates the security credentials of the microservices. In this approach, the API gateway will communicate with the IDP to validate the credentials of the client. Moreover, we propose to implement a common security standard like OAuth 2.0 in the API gateway to enhance the security of the system. We demonstrate the utilization of the API gateway in our architecture in Fig. 5.
- **Defense-in-depth:** Defense-in-depth (DID) refers to the deployment of multiple layers of security to protect a system resource. We employ DID to secure the inter-service communication within the system. When compared to communication with external entities, the security of inter-service communication is different. The consumer and the provider reside in the same internal secured network, thus mitigating the threat scenario. We can implement security for east-west traffic with the following approaches: transport layer security (TLS); TLS with message layer security using authentication; TLS with message layer security using authentication and authorization. We chose to design our security architecture with maximum security because the data present in the system is often highly sensitive. Thus, we opt to have a DID-based system with separate layers for TLS, authentication, and authorization. We demonstrate the DID component in our security architecture in Fig. 5.
  The API gateway and DID help in securing the multitude of entry points in the microservice architecture, thus managing the broad attack surface.
- **Distributed Tracing:** Distributed tracing has been tailor-made to track requests and communication between microservices [13]. Developers usually use distributed tracing for application analytics, where the engineers get real-time information about the health of individual microservices. We propose to use distributed tracing to monitor the communication between microservices. This monitor provides us with a detailed log of requests between microservices. We also deploy distributed tracing on the API gateway. Thus, we can also track the application's communication with external users. This tracking gives a detailed overview of the communication happening over the network.
  Distributed tracing is different from logging and traditional application monitoring. Logging focuses entirely on the operations in a single application. Log data can contain various information ranging from human-interpretable performance metrics to machine-understandable data that can trigger automatic responses. Application monitoring tools aggregate the data from multiple log files to provide insights into application performance metrics such as the overall latency and response rates across various users. However, the system does not have access to the log files of individual microservices in a cloud-native application. Thus, application

monitoring only reports the end-to-end performance of the cloud service provider.

However, distributed tracing can effectively provide fine-grained details about the performance of every microservice operating on the cloud. It observes requests as they travel across distributed cloud environments. The tracer tags an interaction with a unique identifier. This identifier stays with the transaction as it migrates between microservices and virtual machines. The tracer records every request interaction and collects the relevant metadata. Thus, we obtain the series of microservices that a request has accessed. The data includes details of the service name, start and end timestamps, and other metadata. We generate a sequence of interactions from this data, with every element of the series being a multi-dimensional vector. We use these sequences to train Seeker.

- **Self-supervised Learning (SSL):** The scarcity of labelled data warrants the extraction of maximum information from the available data. SSL enables us to extract maximum patterns underlying the dataset. Pre-training our model with SSL will enhance our performance even with kimited data. There is a way to think about SSL within the unified framework of an energy-based model (EBM). An EBM is a trainable system that, given two inputs, x and y, tells us how incompatible they are with each other [16]. For example, x could be a short video clip, and y another proposed video clip. The machine would tell us to what extent y is a good continuation for x. To indicate the incompatibility between x and y, the machine produces a single number, called an energy. If the energy is low, x and y are deemed compatible; if it is high, they are deemed incompatible.
- **Transfer Learning:** Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point. This alleviates the needs for vast compute and time resources required to develop neural network models.

  We have developed a trained ML model for vulnerability detection in Seeker. We can transfer the knowledge from Seeker to identify similar vulnerabilities in cloud-based platforms like Docker and Kubernetes.

5 CONCLUSION

This article proposes the utilization of supervised ML to detect vulnerability exploits on a microservice-based application in real-time. Prior work has used distributed tracing for microservice analytics and unsupervised learning-based anomaly detection. However, we expect our supervised learning framework to have fewer false positives. We also propose a novel idea of introducing the explainability of ML models in the framework. This transparency helps to build the trust of the security analysts and makes it more likely for them to deploy our model in their application.

REFERENCES

[1] 2022. eBay. https://ebay.com/.

[2] 2022. Netflix. https://www.netflix.com/.

[3] Jacob Brown, Tanujay Saha, and Niraj K Jha. 2021. GRAVITAS: Graphical reticulated attack vectors for Internet-of-Things aggregate security. *IEEE Transactions on Emerging Topics in Computing* (2021).

[4] Christian Esposito, Aniello Castiglione, Constantin-Alexandru Tudorica, and Florin Pop. 2017. Security and privacy for cloud-based data management in the health network service chain: a microservice approach. *IEEE Communications Magazine* 55, 9, 102–108.

[5] Muhammad Idris, Iwan Syarif, and Idris Winarno. 2021. Development of vulnerable web application based on OWASP API security risks. In *International Electronics Symposium*. IEEE, 190–194.

[6] Antonio Nehme, Vitor Jesus, Khaled Mahbub, and Ali Abdallah. 2019. Securing microservices. *IT Professional* 21, 1, 42–49.

[7] Anelis Pereira-Vale, Gastón Márquez, Hernán Astudillo, and Eduardo B Fernandez. 2019. Security mechanisms used in microservices-based systems: A systematic mapping. In *XLV Latin American Computing Conference*. IEEE, 01–10.

[8] Tanujay Saha et al. 2022. Machine learning-based efficient and generalizable cybersecurity frameworks. (2022).

[9] Tanujay Saha, Najwa Aaraj, Neel Ajjarapu, and Niraj K Jha. 2021. SHARKS: Smart Hacking Approaches for RisK Scanning in Internet-of-Things and cyber-physical systems based on machine learning. *IEEE Transactions on Emerging Topics in Computing* (2021).

[10] Tanujay Saha, Najwa Aaraj, and Niraj K Jha. 2022. Machine Learning Assisted Security Analysis of 5G-Network-Connected Systems. *IEEE Transactions on Emerging Topics in Computing* (2022).

[11] Tanujay Saha, Najwa Aaraj, and Niraj K Jha. 2022. System and method for security in internet-of-things and cyber-physical systems based on machine learning. (June 23 2022). US Patent App. 17/603,453.

[12] Tanujay Saha and Vikash Sehwag. 2016. TV-PUF: A fast lightweight aging-resistant threshold voltage PUF. *Cryptology ePrint Archive* (2016).

[13] Matheus Santana, Adalberto Sampaio Jr., Marcos Andrade, and Nelson S. Rosa. 2019. Transparent tracing of microservice-based applications. In *34th ACM/SIGAPP Symposium on Applied Computing*. 1252–1259.

[14] Vikash Sehwag and Tanujay Saha. 2016. TV-PUF: a fast lightweight analog physical unclonable function. In *2016 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*. IEEE, 182–186.

[15] Yuqiong Sun, Susanta Nanda, and Trent Jaeger. 2015. Security-as-a-service for microservices-based cloud applications. In *IEEE 7th International Conference on Cloud Computing Technology and Science*. IEEE, 50–57.

[16] Yee Whye Teh, Max Welling, Simon Osindero, and Geoffrey E Hinton. 2003. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research* 4, Dec (2003), 1235–1260.

[17] Kennedy A. Torkura, Muhammad I.H. Sukmana, and Christoph Meinel. 2017. Integrating continuous security assessments in microservices and cloud native applications. In *10th International Conference on Utility and Cloud Computing*. 171–180.

[18] Rongxu Xu, Wenquan Jin, and Dohyeun Kim. 2019. Microservice security agent based on API gateway in edge computing. *Sensors* 19, 22, 4905.

[19] Tetiana Yarygina and Anya Helene Bagge. 2018. Overcoming security challenges in microservice architectures. In *IEEE Symposium on Service-Oriented System Engineering*. IEEE, 11–20.