Received: 12 September 2017 | Revised: 15 November 2017 | Accepted: 3 January 2018
DOI: 10.1002/cpe.4436

SPECIAL ISSUE PAPER

WILEY

A survey on security issues in services communication of Microservices-enabled fog applications

Dongjin Yu[1] | **Yike Jin**[1] | **Yuqun Zhang**[2] | **Xi Zheng**[3]

[1]College of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China
[2]Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China
[3]Department of Computing, Macquarie University, North Ryde, NSW 2109, Australia

Correspondence
Yuqun Zhang, Southern University of Science and Technology, Shenzhen 518055, China.
Email: zhangyq@sustc.edu.cn
Xi Zheng, Macquarie University, North Ryde, NSW 2109, Australia.
Email: james.zheng@mq.edu.au

Summary

Fog computing is used as a popular extension of cloud computing for a variety of emerging applications. To incorporate various design choices and customized policies in fog computing paradigm, Microservices is proposed as a new software architecture, which is easy to modify and quick to deploy fog applications because of its significant features, ie, fine granularity and loose coupling. Unfortunately, the Microservices architecture is vulnerable due to its wildly distributed interfaces that are easily attacked. However, the industry has not been fully aware of its security issues. In this paper, a survey of different security risks that pose a threat to the Microservices-based fog applications is presented. Because a fog application based on Microservices architecture consists of numerous services and communication among services is frequent, we focus on the security issues that arise in services communication of Microservices in four aspects: containers, data, permission, and network. Containers are often used as the deployment and operational environment for Microservices. Data is communicated among services and is vital for every enterprise. Permission is the guarantee of services security. Network security is the foundation for secure communication. Finally, we propose an ideal solution for security issues in services communication of Microservices-based fog applications.

KEYWORDS

container, data, Microservices, network, permission, security

1 | INTRODUCTION

Fog computing is an emerging programming paradigm to resolve latency, bandwidth saturation, mobility support, and location awareness issues in cloud computing.[1-3] Microservices is a latest service-oriented software architecture that contains a number of small independently services. The concept of Microservices arises from the industrial practice to divide large monolithic applications into smaller synergetic services to be more maintainable, scalable, and testable to suit for fog environments.[4,5] Since the architectural goals of both microservices and fog computing are very similar, the development and deployment of fog applications using Microservices architecture are getting increasingly popular.[6-9] Although there are some advantages in the development and deployment of Microservices architecture, the security is always important to the IT community and is the first consideration of customers.[10] We start off by identifying the taxonomy of security issues in Microservices. As shown in Figure 1, we list several security issues[11] of Microservices that fog computing developers should take care of. But instead of taking into account all the problems, in this paper, we focus on the services communication and analyze security vulnerabilities related to the services communication. Specifically, we want to talk about the security issues in four aspects: containers, data, permission, and network.

First, the presence of containers provides a perfect environment for Microservices.[12] The usage of containers to wrap or containerize distributed Microservices has a few advantages. For example, containers remove dependencies on the underlying infrastructure services, which reduce the complexity of dealing with different platforms. Therefore, Microservices architecture could make use of containers (Docker Containers*) to test and deploy single services in separate containers across available network of computers and other computing devices. Moreover, containers can provide

* In this paper, we use containers as a short term for Docker Containers.

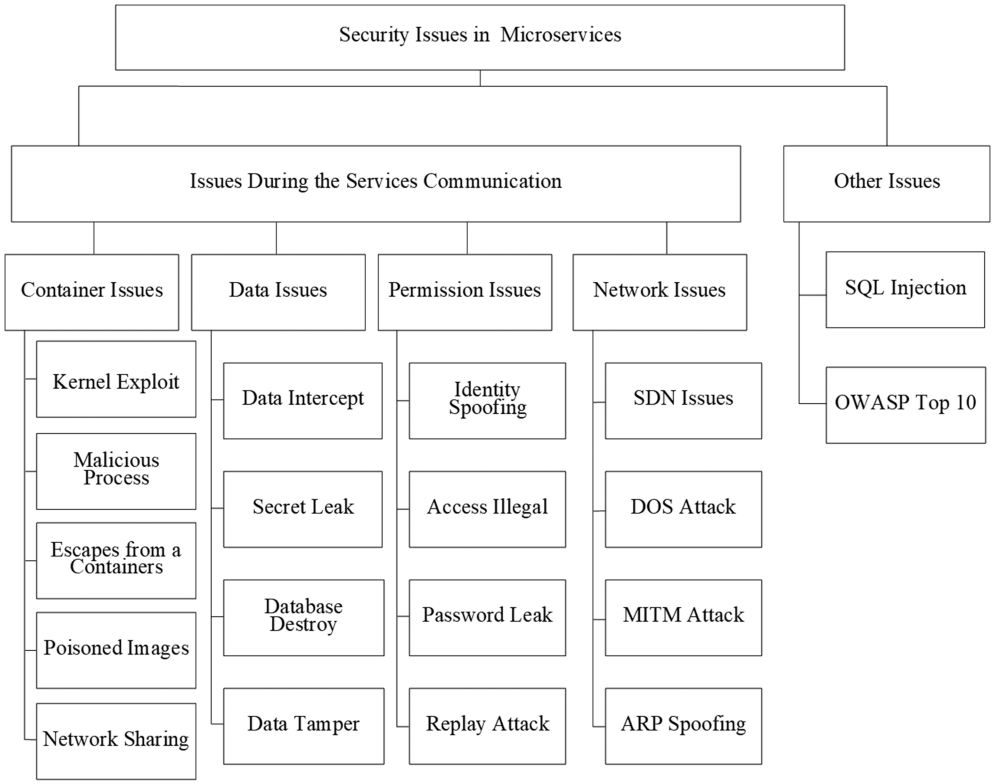

FIGURE 1 A taxonomy of security issues of Microservices

standardized building and continuous integration and delivery. In a nutshell, the existence of the containers is highly relevant to the development of Microservices. They bind together and form an ecosystem. But when the containers has security problems, it will inevitably cause a serious impact on the Microservices. Therefore, we focus on how to ensure the safety of the containers. Since we often compare containers with virtual machine, we would like to discuss and compare the security performance of containers and virtual machines.

Second, Microservices architecture has a characteristic of fine granularity so that a large software system based on Microservices usually contains a great number of services. These services may be developed by different technical teams using different technologies, and their data would also be stored in different databases, which introduces more attacked surfaces in Microservices architecture. In addition, when Microservices is deployed in the cloud environment, it would also introduce data security issues that the private information of cloud users may be affected or misused. While the cloud offers some advantages, until some of the risks become better understood, quite a few users would be tempted to hold back. According to a recent IDCI survey, 74% of IT executives and CIO's cited security as the top challenge preventing their adoption of the cloud services model.[13] Not only is there a risk of information being intercepted, but competitors are also likely to infer business operations from the message flow. Overall, since Microservices is also closely related to the cloud environment, it is necessary to look into the tamper-proof issues of data sources, the authenticity detection, and the protection of the transmitted data in the Microservices architecture.

Third, trust is an important dimension of safety so that we shall consider the permission issues in Microservices. On the one hand, Microservices architecture should verify the authenticity of every service. When a single service is controlled by an attacker, the service may maliciously influence other services. For instance, a malicious service would take up most of the resources to decrease the performance of other services, or interfere with the network, which would make a serious disturbance to the software system using Microservices architecture. On the other hand, when a service receives a message, it needs to find out whether the message is spurious and whether the source service has valid authority. In addition, a Microservice may require authorization to access users' resources and exchange data with third-party services. Then, we need a method to secure the important privacy information, such as a user password. OAuth[14] is a standard that does not require user's account information (such as user password) for the authorization of a third-party application, but its suitability for Microservices needs to be analyzed thoroughly. In general, we shall consider the authentication and authorization issues when services communicate with each other in Microservices architecture.

Finally, the network issues should be considered because only with the secure network, secure communication among the Microservices can be guaranteed. Software Defined Network (SDN)[15] is commonly used with Microservices, which could separate network control from the underlying physical network to effectively get rid of the limitation on the network hardware imposed by the equipment manufacturer. In this way, enterprises can modify the network architecture in a similar fashion as software installation and upgrade to cater for the enterprise's adjustment, expansion, or upgrading of the entire software architecture. Due to the flexibility of SDN, SDN is often used to monitor network flow inside Microservices architecture. However, inevitably, Microservices faces some potential safety hazard of SDN. Especially when the network becomes more complex

and communication is more frequent because of the large quantity Microservices, we need a thorough security analysis of the network issues in Microservices architecture.

This paper will describe the various security issues that may occur during the services communication of Microservices. The remainder of this paper is organized as follows. Section 2 has a brief outline of Microservices architecture and its usage. Section 3 describes the research methodology we adopted for the survey. Section 4 describes the security issues of containers. Section 5 describes the security issues of data in Microservices. Section 6 describes the security issues of Authorization and Authentication in Microservices. Section 7 describes the security issues of network in Microservices. Section 8 presents an ideal solution. Section 9 provides conclusions and future works derived out of this survey.

2 | MICROSERVICES ARCHITECTURE

Microservices is a relatively new software architecture that its research is still limited in both industry and academia domain. In this section, we introduce the definition and the usage of Microservices architecture.

2.1 | Definition of Microservices

The Microservices architecture is an approach to develop a single application as a suite of small services, each running in its own process and communicating with each other by lightweight mechanisms (eg, REST API or SOAP protocol[16]). These services are built around business capabilities and independently deployed by fully automated deployment tools (eg, Jenkins[17]).

Microservices extends the notion of Service Oriented Architecture (SOA).[18] In other words, the difference between Microservices and SOA is that Microservices is more decentralized and distributed. Microservices distributes all the logic (routing, message parsing) into smart endpoints and adopts lightweight API gateway for managing services instead of using heavier and more sophisticated Enterprise Service Bus.[19]

Fine granularity and loose coupling are two significant features in Microservices. Fine granularity means that there is a bare minimum of centralized management of these services, which may be developed by different program languages and with minimum development resources.[20,21] However, not all services in a Microservices architecture are necessarily micro. Microservices will become as big as it needs to be to provide coherent, efficient, and reliable functions. Loose coupling means that each of Microservices components has less dependencies on other separate components, which makes the deployment and development of Microservices more independent.

There are many studies describing the advantages of Microservices architecture over traditional monolithic architecture.[22] For example, Villamizar et al presented a cost comparison that an enterprise application was developed and deployed in the cloud using a monolithic approach and a Microservices architecture, respectively.[23,24] It validates that the provider may reduce infrastructure costs by 17% using Microservices architecture. Figure 2 shows both the monolithic system architecture and Microservices system architecture.

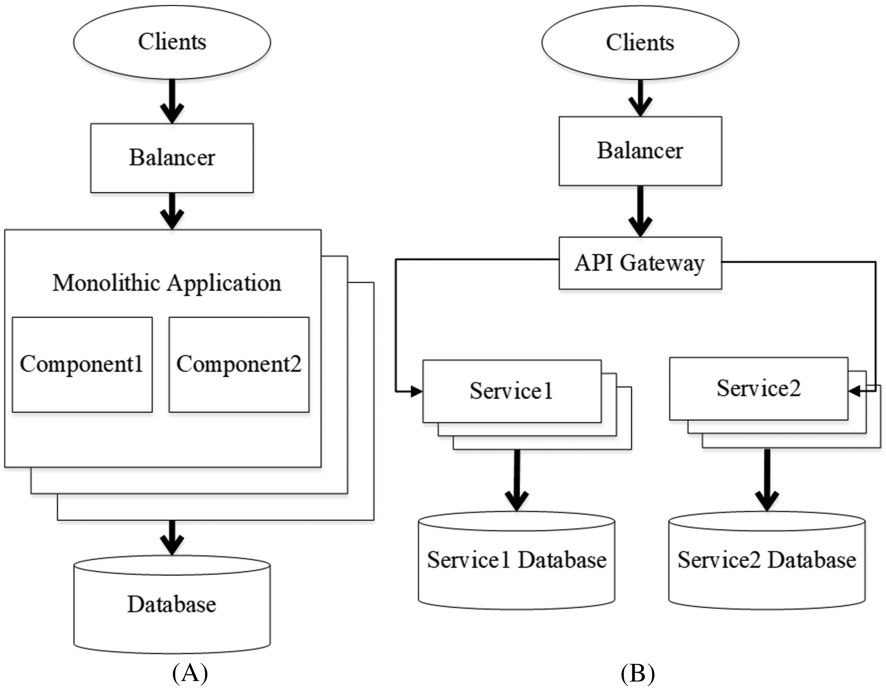

FIGURE 2 The monolithic system architecture and Microservices system architecture

Furthermore, the maintenance and test of software system could be quite convenient because of the fine granularity and loose coupling in Microservices, which shorten software product delivery cycles. Companies have recognized that there is a strong commercial advantage to provide new features to customers ahead of their competitors.[18] These features make Microservices more adopted and popular in the industry.

2.2 | Usage of Microservices

Microservices architecture has been widely used in many industrial practices. For example, the architecture is gaining popularity in the field of Internet of Things (IoT).[25] With the continuous development and evolvement of IoT, traditional monolithic architecture applications become much larger in scale and even more complex in structure. In 2015, Krylovskiy et al applied the Microservices architectural to build a smart city IoT platform,[26] which is used for a variety of applications involving different stakeholders to increase the energy efficiency of a city at the district level. In the next year, Bao et al also proposed a decentralized Microservices architecture based on a message-oriented middleware for IoT in the domain of smart building.[27] Since fog computing paradigm enables a wide range of benefits, including but not limited to reduced latency and increased bandwidth, the implementation of fog paradigm using Microservices architecture is a suitable solution for many IoT services.[28]

In the other domains, Microservices is usually used to improve system flexibility and deal with complex requirements. For instance, Zimmermann et al[29] proposed an architecture based on Microservices to extend enterprise architecture to cater for flexible and adaptive digitization of new products and services. In another case, Bak et al[30] presented three novel Microservices, namely, contextual triggering, visualization, and anomaly detection and root cause analysis, which can substantially accelerate the development and evolvement of location and context-based applications. Moreover, Gadea et al introduced an architecture implemented by a collaborative rich-text editor that makes use of Microservices to enable and enhance its scalable co-editing functionality.[31] The architecture also shows how Microservices makes it possible for multiple users to co-edit a document where images containing faces are added and recognized as part of the document content. In addition, Malavalli and Sathappan[16] proposed a Microservices architecture to implement Distributed Management Task Force (DMTF) Profiles in the middleware layer of a management console.

3 | RESEARCH METHODOLOGY

In this section, we study the existing research efforts that contribute to the research topics on Microservices security. We conducted a structured literature review following the guidelines outlined in other works.[32,33,34] As a result, our literature review process consists of three main phases: literature search and collection, literature selection, and literature analysis and evaluation. To put effort into limited resources, instead of conducting a truly exhaustive literature review across various domains, we aim to focus our research work closely related on the topic of Microservices security in the process of service communication. The research questions and the motivations are outlined in Table 1.

3.1 | Literature search and collection

Before conducting a structured literature search, it is necessary to define a research scope and provide the conceptualization of the topic. Hence, the topic of concern is about Microservices. The terms microservice, microservices, micro service, micro services, micro-service, and micro-services were searched in articles published in journals, conferences, books, and workshops. The research was restricted to articles published between 2010 and 2017, as there was no consensus on the term Microservices architecture in the field prior to that date. We chose two popular scholarly databases, Scopus and Web of Science, for the literature search, since both databases are well-known sources for citations.

Next, the search results in the two databases were manually combined by removing the duplicates between the search results. At the end, we obtained a total of 262 studies about Microservices from our literature search.

TABLE 1 Research questions

Research Question	Motivation
What are the main practical motivations behind using Microservices?	The aim is to get insight into what are the main reasons for organizations architect to adopt a Microservices style.
What are the security challenges that Microservices based fog applications face?	The aim is to explore all the published studies that were relevant to Microservices security issues.
What are the existing methods and techniques to enable secure development and operation in Microservices architecture?	The aim is to identify and compare existing methods and techniques that guarantee Microservices architecture security.
What should be the future research agenda?	The aim is to identify future directions and look for future solution foundations.

3.2 | Literature selection

In this phase, we performed a careful literature selection among the overall 262 studies retrieved from the previous phase. Our literature selection was carried out in three sequential steps.

First, we removed papers that are obviously irrelevant, as informed by their titles. We were cautious about excluding a paper in this step. A paper would not be removed unless the title indicates that it is well beyond the context of the topic related to Microservices security.

Next, we conducted selection based on the topic of each paper. In this step, we searched for papers containing the relevant keywords, such as containers, security, privacy, network, authentication, and authorization. Then, the abstract of a paper was studied and sometimes the content was also reviewed to gain a better understanding about the topic of the paper.

Finally, we performed a paper quality screening. If a paper does not have a clear contribution or suffers from an unclear solution to the research questions raised in the paper, we do not consider it qualified for review and evaluation in the next phase. A number of papers were found, which suffer from quality issues, and were removed.

After applying the exclusion and inclusion criteria, a total of 66 articles were collected in this study.

3.3 | Literature analysis and evaluation

In this phase, we aim to roughly classify the selected papers based on the steps during the communication among Microservices. Once papers were selected, a qualitative assessment was conducted to create an outline model for the whole research work. This helps to abstract various possible dimensions for characterization and categorization. As a result, we classify the papers into four categories: containers security issues, data security issues, permission security issues, and network security issues. These four aspects are all involved in the process of service communication in Microservices architecture.

4 | SECURITY ISSUES OF CONTAINERS

The usage of containers in the cloud makes Microservices development and deployment more agile. However, using the same kernel as the host of different containers could make it possible for attackers to gain unauthorized access to a container when users are not aware. Except kernel exploit, there are several other potential issues of containers: Denial of Service attacks, Escapes from a containers, Poisoned images, and Secret compromise.[35] Thus, securing a containers is necessary to prevent security attacks on Microservices.

In this section, we first analyze and evaluate the effect of containers isolation first, which provides fundamental security protection. Then, we list containers vulnerability and analyze current solutions from the selected papers. Finally, we compare containers with VMs in terms of security performance.

4.1 | Resource isolation

The isolation of the containers is the most critical point for containers security.[36] There are three important aspects to assess the containers security: the built-in native defense of the kernel OS, its prop up features plus namespace, and cgroup.[37-39] Namespace and cgroup are two Linux features that containers adopts to safely create virtual environment. Containers can be helpful for Microservices security by applying the techniques of namespace to isolate resources, such as users, processes, networks, and devices. Namespace creates barriers among containers at various levels and provides the first level of security through containers isolation, which prevents containers from connecting to another. Currently, containers use six namespaces to provide each container with a private view of the underlying host system: PID, Mount, UTS (UNIX Timesharing System), IPC (Inter Process Communication), Network, and User. Each of them works on specific types of system resources, such as process isolation, filesystem isolation, device isolation, IPC isolation, and network isolation.

For process isolation, compromised containers would only be allowed limited access to process management interfaces to impact other containers. This could be realized by applying the hierarchical features of PID namespace,[40] where processes can only observe those in its own namespace or its children namespaces. In addition, the child namespace cannot see anything in the parent PID namespace. The processes in the different containers belong to different namespaces, which do not interfere with each other because the PID namespace hides all processes that are running on a system except those that are running in the current containers. If attackers cannot see the other processes, it would be very difficult to initiate the attack process as the attackers cannot easily strike or trace them. In addition, tasks in different namespaces can have the same IDs. That is the reason that the PID namespace is also the major factor for the migration of containers among hosts. A task is not required to change its PID while migrating from a containers in one host to another host.

The filesystem isolation is also managed in the similar manner where Mount namespace is used. Containers use the Mount namespace, also called the filesystem namespace, to isolate the filesystem hierarchy associated with different containers. The Mount namespace provides the processes of

each containers a different view of the filesystem tree and restricts all the mount events occurring inside the containers.[41] Specifically, the threats from compromised containers to the host OS through filesystem can be removed by revoking the write permissions from the containers or not allowing any process of a containers to remount that filesystem in other containers.

Network namespace will isolate network-related system resources so that every network namespace has its own IP addresses, IP routing tables, network devices, and ports. Network isolation is necessary in the containers. For example, with the absence of network isolation, if two different containers all want to run the same web application, and they all need to use port 80 from the same host, then there will be a conflict. Network namespace can also be used to defend network-based attacks in the containers, such as ARP spoofing. Of course, the existing protection is not enough to cover more complicated patterns of network attacks, and we discuss it in the section as follows.

The IPC namespace provides isolation for various inter-process communication (IPC) mechanisms, namely, semaphore, message queues, and shared memory segments. The processes running in containers must be restricted so that they can communicate only via a certain set of IPC resources. Docker assigns an IPC namespace to each containers, which prevents processes to interfere with others in different containers because processes in an IPC namespace cannot read or write the IPC resources in other IPC namespace.

The UTS namespace ensures that different containers can view and change hostnames specifically assigned to them. UTS namespace provides the isolation of the host name and domain name, so that each containers can have independent host name and domain name. In other words, any containers can be regarded as an independent node in the network, rather than a process on host,[38] which prevents a necessary level of attack isolation.

The second type of containers resource isolation is cgroups, which is part of kernel subsystem and provides a fine-grained control over sharing of resources like CPU, memory, and group of processes. In multi-tenant systems, Denial of Service is the most common. Cgroups can be used to mitigate it by controlling the amount of resources that any Docker containers can use. Therefore, a Distributed Denial of Service attack[42] cannot deplete other containers resources.

For device isolations, it is essential to protect device drivers from those malicious containers. The solution provided in cgroups is Device Whitelist Controller, which could be used to control the devices used by containers. By default, containers do not give any device privileges to its containers. Therefore, attackers cannot access any device. Furthermore, containers takes advantage of the special cgroups that allows users to specify which device nodes can be used within the containers. It blocks the processes from creating and using device nodes that could be used to attack the host.

4.2 | Compare containers with VMs

Although containers can be flexible as they do not include guest OS compared with VMs, it is not straightforward to determine which one is better in terms of security performance.[43] The differences in the architecture design bring some benefits to containers-based virtualization over hypervisor-based virtualization (VMs). First, containers-based virtualization can provide higher density of deployable contents in the virtual environments. Since a containers does not include an entire OS, the size and the required resources to run an application in a containers are less than that in a VM running the same application. As a result, containers has more deployable capacity than traditional virtual machines. However, VM supported by hypervisor-based virtualization techniques claims to be more secure than containers as they add an extra layer of isolation among applications in the host. An application running inside a VM is only able to communicate with the VM kernel, not the host kernel. Consequently, in order for an attack on an application to escalate out of a VM, it must bypass the VM kernel and the hypervisor before it can attack the host kernel. On the other hand, containers can directly communicate with the host kernel, which allows an attacker to save a great amount of effort when breaking into the host system.[41] Containers are also featured with process isolation, network isolation, filesystem isolation, device isolation, and limitation of resources[38] to avoid such attacks from the malicious containers. However, Gupta[44] provided a brief overview about comparisons between security of containers and VMs and concludes that Docker isolates many aspects of the underlying host, but the separation is not as strong as that of VM.

As aforementioned, containers have a bigger attack surface since containers can directly communicate with the host kernel, which can be overcome by placing containers inside virtual machines. The ideal situation would be to have few VMs installed on physical machine and then have many instances of containers running on these VMs, so public cloud providers can offer containers that start inside virtual machines to protect their data center assets against external attacks.[45] However, Bacis et al[35] found out that installing Docker directly inside a virtual machine is not the right solution as it only creates a more complicated system, which will add more attack surface.

4.3 | Containers vulnerability

There are two main categories of adversaries in containers: direct and indirect.[46] Direct adversaries target the core services inside containers, and they can destroy or modify the network and system files. Locally or remotely, direct adversaries could attack several system components. For example, attackers can gain root privileges in a related containers from an Internet-facing containers service. Then, from the attacked containers, they can make an attack to other containers running on the same host operating system and gain access to critical host operating system files.

Indirect adversaries have the same capabilities as direct ones, but they target the containers ecosystem, such as the code and images repositories, to reach the software environment. The containers attack surface contains the whole deployment tool chain. The deployment tool chain includes image conception, image distribution process, automated builds, image signature, host configuration, and third-party components. In addition, the source of vulnerabilities in image distribution is the containers hub (eg, Docker Hub) and other registries in the containers ecosystem. Moreover, the setup of the containers images distribution adds several external steps, which increase the global attack surface.

There are some effective namespace-based protection mechanisms. Jian and Chen[47] proposed a method of inspecting containers namespace status to defend the exploitation of kernel vulnerability by malicious users. For instance, once the hacked program in the containers launches an effective escape attack and then it can gain root privilege of the host, which will affect the reliability of other containers or the entire system. The escaped process will spawn a command shell to obtain full control. The detection program proposed from the work of the aforementioned author[47] runs on user layer of the host system, mainly to detect escape behaviors by comparing the namespace status in each containers. In another case, Gao et al[48] presented a power-based namespace method for detecting information leakage on a multi-tenancy containers cloud service. Gao et al[48] first found out that the information leakage is mainly due to the incomplete implementation of system resource isolation mechanisms in the Linux kernel, and then it proposes the power-based namespace approach, which is able to record and monitor the power usage per containers. With power usage statistics for each containers, the proposed method can dynamically restrict the computing power of containers that have exceeded their predefined power thresholds, which achieve effective resource isolation.

SELinux (Security-Enhanced Linux)[36] is a widely used security module for Linux and its support for security policy modules has already proved to be a significant enhancement in the industry. Bacis et al[36] proposed an extension to the format of a containers file (ie, Dockerfile) to bind a specific SELinux policy module with the containers images, which enhances the security of containers. Bacis et al[36] also found out that the major threat of containers is mainly from compromised or malicious programs attacking other containers that are running on the same host. The adaptation of SELinux policy modules into protecting containers security allows the specification of SELinux domains for different containers' images, leading to an increase of security in containers.

There are also several security issues in the containers network. Bui[41] presented that the problem with containers was related to its default networking model. The virtual Ethernet bridge, which containers use as its default networking model, is vulnerable to ARP spoofing and MAC flooding attacks since it does not provide any filter on the network traffic passing through the bridge. However, this problem can be solved if the administrator manually adds filtering, to the bridge, or changes the networking connectivity to a more secure one, such as virtual network. It is also worth highlighting that, if the operator runs a containers as "privileged", container is able to grant full access permissions of the host operating system, which is nearly the same as that of processes running natively on the host system. Therefore, it is more secure to operate containers as "non-privileged."

Some researchers propose the idea of balanced security in containers. Manu et al[49] attempted to provide unified privacy and multilateral security architecture for cloud services stack. In their proposed prototype, they define various configurations, which can be used to achieve layered multilateral balanced security in containers implementation and balance the security concerns from vendors as well as from consumers. In order to provide security to containers running in data center, both customers and vendors need to keep continuous vigilance with multilateral monitoring and access. The security responsibility is shared with cloud stakeholders involving all the clustered cloud service customers and vendors. Certain major security implementation can be undertaken jointly by both stake holders. Continuous monitoring and verification are used to prevent MAC spoofing and IP spoofing on the containers.

Security issues also arise in multi-user environments. Since multiple users have the permission to deploy containers, and thereby are able to run potentially malicious program that is hidden inside a publicly available containers' image.[45] The multi-tenancy model has at least created two new security issues. First, shared resources (eg, hard disk, data, and VM) on the same physical machine are exposed not only to normal resource requests but also to malicious attacks. Second, the issue of multi-tenancy sharing will severely damage the reputation of genuine tenants who share the computing resources with malicious tenants. Since all the tenants may share the same network address, any malicious conduct will be attributed to all the users belonging to the same network address thus impacting the resource access for legitimate users.[50]

4.4 | Pros and cons of solutions

A few existing solutions focus on the vulnerability issue inside containers' kernel. The work of Jian and Chen[47] is mainly to detect escape attack behaviors by comparing the namespaces status in each containers. Escape attack is exploiting a kernel vulnerability to penetrate through the host and thus attack other containers. The proposed method in the work of the aforementioned author[47] is simple and effective for escape attack. However, it is not applicable for other potential threats such as DoS attack. Gao et al[48] proposes to introduce more security features, such as implementing more namespaces and control groups. However, some system resources are still difficult to be partitioned. The proposed version of containers implementation is no different from a virtual machine, and thus loses all the containers' advantages. Bacis et al[36] presents a method that adds a SELinux policy module for Docker to enhance security because SELinux is a sound security subsystem of Linux, but it can be only applied to essential kernel vulnerability, such as information leakage. Manu et al[49] presents a notion of balanced security in containers that the security responsibility is shared among cloud stakeholders involving all the clustered cloud service users and vendors, though there are no

TABLE 2 Pros and Cons of containers solutions

	Pros	Cons
Jian and Chen[47]	Detect escape attack behaviors by comparing the namespaces status in each containers.	It is not applicable for other potential threats.
Gao et al[48]	Implement more namespaces and control groups to isolate resource.	It is no different from the solutions based on virtual machine and loses all the containers' advantages.
Bacis et al[36]	Add a SELinux policy module to enhance security.	It is only applied to essential kernel vulnerability.
Manu et al[49]	The security responsibility is shared among cloud stakeholders.	Sharing security responsibility is difficult to balance.

actual targeted solutions and sharing security responsibility is difficult to balance. We list the pros and cons of existing containers solutions in Table 2.

5 | SECURITY ISSUES OF DATA

Microservices needs more complex communication because of its fined-granularity. Therefore, there is not only a risk that message data could be intercepted but also the threat that competitors might be able to infer business operations from message data.[51] Since Microservices is often deployed into cloud environments, Microservices also suffers from privacy issues in addition to message transfer and cloud consumers also have the concern that their stored information could be compromised or used inappropriately. Microservices is deployed in many distributed containers, so the customers could be more suspicious to their private information.[52] How to prevent transmitted messages from leaking remains a serious challenge.

More likely, data security is to detect security vulnerability inside the services themselves and detect malicious operations. To ensure data confidentiality, integrity, and availability, the Microservices provider must offer minimum data security capabilities including an encryption schema to protect all data in the shared storage environment, stringent access controls to prevent unauthorized access to the data, and safe storage for scheduled backup data.

In this section, first, we introduce some methods for data encryption. Next, we walk through several other data protection methods.

5.1 | Data encryption

With the increasing popularity of Microservices-based fog applications deployed to cloud environment, privacy, and confidentiality are gaining more focus and priority. Tenants accessing Microservices-based fog applications (eg, an individual, a business, a government agency, or any other entities) want to make sure their private data are secured.

Typically, cryptographic methods can be categorized to be symmetric or asymmetric.[53] The symmetric cryptography is simple in that a symmetric key is used to encrypt data, which can be decrypted in the same way. In comparison, in asymmetric cryptography, data is encrypted with a given public key and can only be decrypted with a corresponding private key. Generally, public keys are distributed to all parties interested in the data communication, while private keys are held securely by the message senders. To protect data in Microservices-based applications, a common practice is to use an efficient symmetric key to encrypt the tenant's critical data. To add another layer of security, a public or private key pair can be used to encrypt and decrypt the symmetric key. With this approach, three keys are created for each tenant as part of the provisioning process: a symmetric key and an asymmetric key pair consisting of a public key and a private key.[54,55] Using encryption to protect data is especially important in situations involving mission critical data or with high level concerns for the data privacy, or when multiple tenants share the same set of database tables (as in Software as a Service Cloud Model).

As a general rule, the longer the encryption key is, the more secure the encrypted data will be. As a result, the process of encryption and decryption requires more performance overhead. To protect data using encryption, we need to consider the usage scenario and sensitivity of various data in Microservices data models when making decisions which encryption algorithms to be used to keep a good balance between overhead performance and required data provenance protection. For sensitive information, data security might be the most important so that the most secure encryption algorithm shall be adopted, which inevitably would compromise certain system performance. For frequently transmitted and common data, which can have some security compromise, more efficient encryption methods can be used to ensure Microservices performance.

There are also a few works proposed to use data auditing protocols to complement data encryptions. Shah et al[55] argued that third-party auditing is important because it allows customers to evaluate risks and increase the efficiency of insurance-based risk mitigations for data privacy.

Approaches for supporting both internal and external auditing of online storage services are proposed as well. The purpose of the audit is to monitor the server and the network, record the usage of the system, and discover violation of the security policies. The network auditing system records the user's behavior in detail, and some systems can immediately report to the administrators about suspicious events. Wang et al[56] proposed that data encryption is only complementary to the privacy-preserving public auditing scheme. Without properly designed auditing protocols, data cannot be prevented from being leaked to external parties using encryptions alone.

Some argue that, for enterprise-level cloud services, encryption systems should offer additional high performance, full delegation, and scalability along with fine-grained access control. Unauthorized data leakage still remains a problem due to the potential exposure of encryption keys. To address this problem, one alternative is to combine the homomorphic authentication with random masking such that third-party authenticator no longer has all the necessary information to derive the owner's data content.[57]

5.2 | Data provenance protection

In data security, the protection of data source is also very important. Data source protection is a defense against malicious operators tampering the primary data. However, services inside the Microservices-based fog applications may originate from different vendors across multiple organizations, which raises a potential threat in the protection of data source and increases more attacked surface.

The data source is required to be determined if it is verifiable or trustworthy and how to protect against tampering attempts. Therefore, each data source should have a reputation degree associated with itself. This would give data consumers a way to determine the trustworthiness degree of the data prior to using it in their computations.[58]

Callegati[59] presented four aspects to protect the data source. First, proposing a data source management system with verifiability, accountability and reproducibility. Verifiability refers that the management system shall be able to verify the actors (or services) involved with the operations on the data source. Accountability refers that the management system shall be able to hold an actor (or service) accountable for its actions in the data source. Reproducibility means that the management system shall be able to reproduce a process on the data source. Second, creating a private and public key system for data stream certification. Third, using a provenance verification approach based on cryptography to ensure data properties and integrity for data source hosts. Fourth, combining data metadata propagation with key distribution propagation management to guarantee credibility of the data source management system.

Subashini and Kavitha[60] proposed that Vendor must adopt additional security checks to ensure data source security and prevent security vulnerabilities in the application. All sensitive enterprise data should be regularly backed up in a safe place to facilitate quick recovery in case of disasters. Strong encryption schemes are also required to protect the backup data.

5.3 | Pros and cons of data security approaches

Encryption is a widely used method to ensure security and privacy of the data in Microservices communication. However, it has some weakness. For instance, Somani et al[61] has tried to evaluate cloud storage methodology and data security by the implementation of RSA algorithm. RSA[61] is currently one of the most widely used public key cryptographic algorithms, which can resist many password attacks known so far, and has been recommended by International Standard Organization as the public key data encryption standard. RSA encryption arithmetic is of high strength and asymmetric encryption. Although the length of RSA key is very long and is very difficult to break, decryption could potentially take a very long time, which would reduce system performance and efficiency.

There are many other existing encryption techniques widely used to protect data in the cloud environment.[62-71] Agarwal et al[63] proposes a data encryption service in a cloud computing environment and provides a centralized framework for effectively managing the data encryption requirements of various applications. Cheung et al[64] proposes a mechanism for managing credentials on an electronic device and providing encryption and decryption services for the electronic device comprising a mobile communication device, which is configured with a data encryption service application and an associated secure data repository. The data encryption service application is configured to encrypt or decrypt data and optionally digitally sign the encrypted file. And the encrypted file is contained in a sandbox environment associated with the data encryption service application. We list the pros and cons of existing data solutions in Table 3.

6 | SECURITY ISSUES OF PERMISSION

Microservices is generally deployed in distributed computing environment, which can cause some security issues mainly due to the use of ineffective access control mechanisms.[72] In this section, we talk about the security issues of permission in Microservices such as authentication and authorization.

TABLE 3 Pros and cons of data solutions

	Pros	Cons
Somani et al[61]	Adopt RSA algorithm that makes encrypted data very difficult to break.	It will take very long time to encrypt and decrypt data.
Agarwal et al[63]	Propose a data encryption service for effectively managing the data encryption requirements.	The communication between services will be more complex and the encryption service requires non-trivial hardware devices.
Cheung et al[64]	Propose a mechanism for managing credentials on an electronic device.	There should be measures to guarantee the security of electronic device first.

6.1 | Authentication and Authorization

Microservices architecture should verify the authenticity of every service because if a single service is controlled by an attacker, the service may maliciously influence other services. Furthermore, when a service receives a message, it needs to find out whether the message is spurious and whether the source service has valid authority. Gegick and Barnum[73] proposed that only the minimum necessary rights should be assigned to a subject that requests access to a resource and should be in effect for the shortest duration necessary. Granting permissions to a user beyond the scope of the necessary rights can allow that user to obtain or change information in improper ways. Therefore, careful delegation of access rights can limit attackers from damaging a system.

Lightweight Directory Access Protocol (LDAP)[74] is quite a popular authorization protocol used in SaaS applications. The use of LDAP, a service that stores relatively static authorization information, is suitable for *Once Records*, *Multiple Reads* scenarios. In terms of data structure, LDAP is a tree structure that can effectively and clearly describe an organization's structural information. LDAP simplifies the DAP (Directory Access Protocol), in order to provide a lightweight one based on TCP/IP network and reduce the management maintenance costs. LDAP applications can easily add, modify, query, and delete user directory information.

There are some methods to ensure the security through permission. Targeting at shared resources inside organizations, it is argued that authorization policies should be built in a hierarchy way to be scalable, flexible, and expressible. To address the issues of access control, Grid Security Infrastructure (GSI)[75] is widely used. GSI includes private key protection and communication encryption. Compared with traditional authorization systems, Community Authorization Service (CAS) model[76] can provide mechanisms for distributed administration that are critical for solving the issues of scalability and flexibility. The CAS model allows resource providers to delegate some of the authority for maintaining fine-grained access control policies to communities, while still maintaining ultimate control over their resources. Patanjali et al[14] described a novel modular and Microservices-based design architecture for developing a dynamic rating, charging, and billing for cloud service providers with emphasis to the validation of the architecture. Security for each Microservices is guaranteed through the integration of OAuth. OAuth[77] is also a popular authorization protocol and a common way to protect Representational State Transfer (REST) APIs from unauthorized access. With OAuth, an authorization server issues access tokens to trusted client applications, which can then access the API on behalf of the end user. Additionally, the extension of OpenID Connect is used. OpenID Connect is an authentication layer on top of OAuth that also allows services to read basic user information.

Thanh[78] introduced an approach allowing cloud application developers and service providers to consider permission security requirements across the application lifecycle. The approach is to use a Microservices-based DevOps[79] framework that utilizes a few emerging technologies including Network Functions Virtualization. A remote patient monitoring scenario is implemented in which vital health parameters (eg, heart rate) of patients are securely collected, stored on the cloud, and accessed at real time through a set of APIs. The proposed framework has two significant security benefits. One is virtualized security functions/services that the framework enables users to select and integrate multi-vendor security solutions to secure the applications. To secure access at the network layer, the OpenStack Firewall as a Service (FWaaS) extension is used. The FWaaS enables users to apply different access control policies on network traffic entering and leaving the application network. To provide token-based authorization for API accesses, two security standards are included: OAuth version 2 and Attribute-based Access Control (ABAC). The primary goal of ABAC as an access control model is to fulfill the requirements of highly heterogeneous environments such as multi-cloud environment. Another benefit is centralized security management and orchestration that the framework allows its application to be configured and protected effectively according to consistent policies as well as to support a set of automated mechanisms.

6.2 | Pros and cons of solutions

Li et al[77] presents a Microservices architecture using OAuth to guarantee the security of permission. The goal of OAuth is to authorize third parties to access user resources in a controlled manner. In OAuth, the website actually gets the access to user account and confirms user identity, which introduces safety issue because the website gets access to user's unique identity at the same time accessing a provisional key of user's account. By default, the authorization method of OAuth can only read user's public information (not including E-mail addresses). If the website needs to get more advanced permission, it needs to make a clear statement and requires the user's consent. But some websites use OAuth as a default authorization

tool, which will by default to grant OAuthor more advanced authorizations that would leak out users' information while users are entirely negligence of this information breach. Cheung et al[64] presents a Microservices-based framework that adopts OAuth version 2 that fixes some security flaws of OAuth. OAuth version 2 adopts HTTPS protocol and uses simpler the authorization verification process, but the issues discussed above remain.

7 | SECURITY ISSUES OF NETWORK

With the development of the Internet, there are many illegal visits, malicious attacks and other network issues that have affected the interests of enterprises and individuals.[80] Thus, network security gradually becomes an important issue that we cannot ignore. To improve the network security, various solutions have been introduced.[81-85] In addition, network security is critical in Microservices because there is more frequent communication among services. Once a network attack has caused a delay or outage of a node, the entire Microservices framework would be paralyzed. Only secure networks can ensure secure services communication. In this section, we talk about network threats and corresponding effective solutions.

7.1 | Traditional network threats

In the domain of network security, traditional typical threats mainly include Address Resolution Protocol (ARP) spoofing,[86] Man in the Middle (MITM),[87] and Denial of Service (DoS).[88] ARP is a TCP/IP protocol that gets physical addresses from IP addresses. When a host sends a message to a targeted host, an ARP request containing the target IP address is broadcasted to other hosts in the network. Then, the host will receive the return message containing the physical address of the target host. When the return message is received, the target IP address and physical address are stored in the native ARP cache and retained a certain amount of time, which will save resources in the next request. ARP spoofing involves constructing forged target physical address of ARP replies. By sending forged ARP replies, the host could not communicate with the target host correctly. Some solutions have been proposed to handle the ARP spoofing. For instance, Abad and Bonilla[89] have outlined some requirements of an ideal ARP solution. First, the ideal solution should not require changes to be made to every host on the network, as this increases the administrative costs. Second, the performance of ARP request/replies should not be slowed down significantly. Therefore, the use of cryptographic techniques should be minimized or avoided. Third, the ideal solution should be widely available and easy to implement. Forth, costly hardware requirements should be minimized as much as possible. Fifth, the ideal solution should be backward compatible with ARP. Sixth, all types of ARP attacks should be blocked.

In brief, the MITM attack is to intercept normal network communication data and manipulate it when the two sides of the communication are unaware of it.[90] Haataja and Toivanen[87] proposed two new MITM attacks on Bluetooth Secure Simple Pairing (SSP). The attacks are based on the forged information during the input/output process and aim at the weakest spot. There are many solutions directed to MITM attack. Meyer and Wetzel[91] presented a solution for MITM attack on the Universal Mobile Telecommunication Standard (UMTS).[92] The UMTS defines the network authentication that depends on both the validity of the authentication token and the integrity protection of the security mode command. Nam et al[93] proposed an enhanced version of ARP to prevent ARP-based MITM attacks. The proposed mechanism is based on the following concept. When a node knows the correct MAC address of a given IP address and it retains the mapping while the machine is alive, then MITM attack is impossible for that IP address. In order to prevent MITM attacks of a new IP address, a voting-based resolution mechanism is proposed. The proposed scheme is backward compatible with existing ARP. Alicherry and Keromytis[94] developed a scalable solution named *Double Check* as SSH and Firefox extensions to prevent MITM attacks. The solution is achieved by retrieving the certificate from a remote host using multiple alternate paths. The scheme does not require any new infrastructure. Hence, its solution is easy to deploy in practice and does not introduce any privacy concerns. Joshi et al[95] proposed and implemented a novel approach to solve MITM over Secure Sockets Layer (SSL),[96] which hashes the user information with the public key of the server's digital certificate. This approach could prevent the MITM attack because once MITM attack tampers the user information, the hash would be changed, thus easily detected.

DoS attack is a network attack method that is commonly used to disable servers or networks. To be specific, DoS attack intentionally exploits the flaw of network protocol or depletes the object resources with the aim to render the target servers or network stop responding to users or simply collapse. DoS itself has been well studied in the past decades including but not limited to the works of Boraten and Kodi[97] and Khan et al.[98] Chang[99] first described various Distributed Denial of Service (DDoS) attack methods and presented a systematic review and evaluation of the existing defense mechanisms. A longer-term solution named Internet-firewall approach is discussed, which attempts to intercept attack packets in the Internet before reaching the victims. Besides, Hussain et al[100] introduced a framework for classifying DoS attacks based on header content, transient ramp-up behavior, and spectral analysis. In addition to helping understand attack patterns, classification mechanisms are important for the development of realistic models to detect and classify DoS attacks. The classification work can be packaged inside an automated tool to help generating rapid response to DoS attacks, and the tool can also be used to estimate the level of DoS severity.

There are also some promising solutions proposed. Gerlach and Lebert[101] discusses a simple and effective method using operationally-based thresholds to prohibit DoS attacks. anderson et al[102] proposed an approach to prevent and constrain DoS attacks that senders must first obtain permission from the destination receiver. A receiver provides tokens or capabilities to those senders to grant communication permission. The senders

then include these tokens in traffic packets. This enables verification points distributed around the network to check traffic that has been certified as legitimate in both endpoints and the path and to clearly discard unauthorized traffic packages.

7.2 | Other network issues

In general, securing monolithic services are relatively easier than securing Microservices.[103] Monolithic services have a clear boundary and the communication is internal and thus encapsulated. Whereas in a Microservices-based fog system, any function completion may require the communication among multiple Microservices over the deployed fog network. This will expose more data and information or endpoints of the system thus expanding the attack surface. The secure services communication will not achieve without the network protection.[104-109]

Underlying infrastructure also plays a vital role to impact the security of Microservices-based fog architecture. Newly emerging networking paradigms such as Software Defined Network (SDN) delivers advantageous features such as flexibility and efficiency to cloud management. However, SDN also introduces new threats that did not exist before and were harder to exploit in traditional networks, which potentially makes network penetration easier. One of the main threats to SDN is the authentication and authorization of network applications. Controller modules must undergo series of tests and verification to make sure that they are reliable and suitable to use in a production environment. However, it is difficult to guarantee the reliability and trustworthiness of a third-party application. The controller modules are centralized and have full information of the network under control. So if a malicious application takes over the controller, the result can lead to different types of attacks. Then, the consequences are severe and have impact on the entire Microservices architecture. Therefore, trust violation is a main threat and resides between the controller and the applications. SDN networks are designed with policies that explicitly allow network applications to apply changes in the network, but no formal verification technique or semantics to evaluate the trustworthiness of these applications. Malicious applications can abuse these privileges and harm network operations.

In view of the security issues of SDN aforementioned, Aliyu et al[15] proposed a framework that helps the control layer in SDN to authenticate network applications and set authorization permissions to restrict manipulation of network resources. The proposed framework has five modules. First, when an application initiates a request to implement network changes via the control plane, the Authentication module checks the authenticity of the application by consulting the data store in a Trust Database. Second, the Authorization module queries the Trust database where permissions are stored to assign privileges to applications. Third, the Trust Database module stores authentication and authorization information. Fourth, the Policy Database module defines the implementation of the global network policy. Fifth, the Monitoring and Evaluation module review and evaluate periodically the relationship that exists between controller and the network applications. The five modules are proved to be effective to improve the security and reliability of the control layer in SDN.

Except SDN security issues, Sun et al[110] considered two other network security issues. First, the network complexity introduced by the large number of Microservices greatly increases the difficulty in monitoring the security of the entire application. Second, compromise of a single Microservice may bring down the entire application. The authors propose a design named *Security-as-a-Service* for Microservices-based cloud applications. By adding a new API *FlowTap*[110] for the network hypervisor, the authors build a flexible monitoring and policy enforcement infrastructure for network traffic to secure cloud applications. *FlowTap* is a contract established between Microservices-based application and cloud infrastructure regarding how to deliver network events. The *FlowTap* is designed to be an API that can be invoked by cloud applications. To support monitoring policies at a higher level of abstraction, the authors provide a *FlowTap* compiler, *FTC*. This compiler translates a given set of high level security policies usually provided by the application to a sequence of *FlowTap* API calls necessary to implement the security policies. The compiler is designed to generate *FlowTrap* calls according to the network monitoring strategy chosen by the administrator, which can maximize the efficiency of the system while preserving the specified security levels.

7.3 | Pros and Cons of solutions

Although there is still no perfect solution to defend all the Microservices related network issues, some approaches could effectively mitigate them. Aliyu et al[15] proposes a trust framework that is designed to address a vulnerability in the SDN architecture that exists between the controller and the network applications. The SDN paradigm supports ;party application deployment, so it is affected by the problem of trust within Microservices-based fog applications. Breach of trust can result in different types of attacks and serious consequences that affect the entire Microservices-based fog system operation. The framework introduces modules to verify the authenticity of network applications and to assign privileges. However, the proposed framework needs to query permission information from the Trust database twice when authentication or authorization is requested. Due to the frequent communication among Microservices, the overhead of required permission query will increase the complexity of the system. And if the corresponding database table has a large amount of records data, this permission query will cause delay for the system. In addition, Sun et al[110] presents *FlowTap*, an API for the cloud infrastructure that enables it to support fine-grained virtual network monitoring. *FlowTap* establishes monitoring relationships between Microservices and security monitors and allows security monitors to execute policies on the network traffic. *FlowTap* is flexible enough to support various monitoring scenarios and strategies with minimal overhead. By using *FlowTap*,

TABLE 4 Pros and cons of network solutions

	Pros	Cons
Aliyu et al[15]	It's easy to verify the authenticity of third-party application in the Trust database.	The overhead of required permission query will increase the complexity of the system.
Sun et al[110]	Propose an API to support fine grain virtual network monitoring.	It also needs other defensive measures to deal with network attacks.

cloud vendors can provide security as a service for cloud applications based on microservice architecture. Although the proposed solution is helpful for network monitoring, companies also have to take other defensive measures to deal with network attacks when there is discovering some malicious attackers. We list the pros and cons of existing networks solutions in Table 4.

8 | AN IDEAL SOLUTION

Considering the security complexity introduced by Microservice-based application, a more thorough and integrated security mechanism is required. The ideal solution could integrate some security solutions into multiple layer system[111] that can prevent a single point of failure to compromise the security of the entire Microservices-based fog application. Other than the integrated security solution, Gegick and Barnum[52] presented the notion of *Reluctance to Trust* and the notion of *Never Assuming That Your Secrets Are Safe*.[112] The authors proposed that developers should assume that the environment where their system resides is insecure and shall minimize the trust dependency on other applications, which can potentially increase the security of their applications. In addition, attackers are more likely to attack the weakest spot in a software system than to penetrate a well-secured component.[113] For example, some cryptographic algorithms can take more procedures to break, so attackers would not likely attack encrypted communication data. Instead, the endpoints of communication may be much easier to attack. So for the ideal solution, weak links in the system should be mitigated as much as possible. In addition, since Microservices-based fog application heavily relies on HTTP request, secure message delivery process should also be enforced (eg, HTTPS).[114]

In this section, we are going to present the ideal solution to deal with the service communication of Microservices from four aspects. The whole ideal solution of service communication in Microservices architecture is shown in Figure 3. Containers adopt SELinux in the VMs to ensure the containers security. Mixed encryption algorithms for data and monitoring systems are used to ensure the data security at runtime. Adopting *Spring Cloud Security* framework[115] ensures the security of authentication and authorization. For network protection, we adopt SDN-based security monitors to deal with network attacks. We will describe each aspect in detail in the following sections.

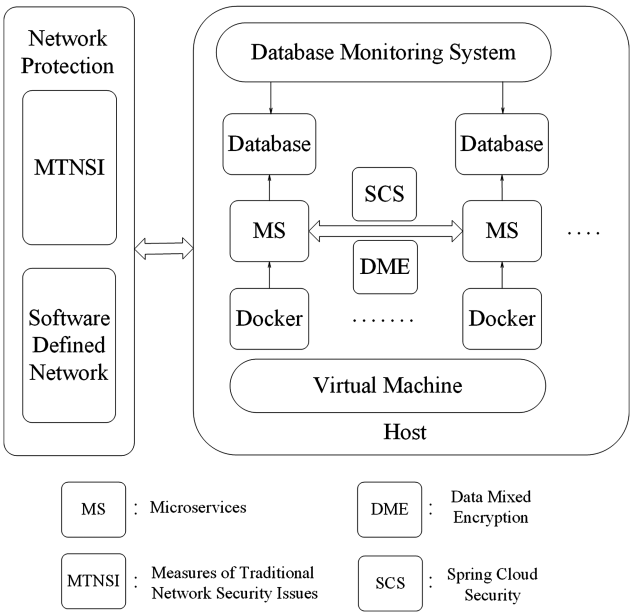

FIGURE 3 An ideal solution of service delivery in Microservices architecture

8.1 | Container security

As shown in Figure 3, installing VMs on physical machine and running many instances of containers on VMs could be a reasonable solution. Although the solution may increase some complexity of the whole software architecture, it could overcome the containers vulnerability that containers can directly communicate with the host kernel. In this case, even if there is a malicious containers trying to attack the host system, it must bypass both the VM kernel and the hypervisor, which provides a first line of defense. Moreover, the mechanism of namespaces and cgroups in containers also provides effective isolation for almost all resource, such as process, filesystem, device, and network, which provides the second line of defense. But the isolation capability of containers is not enough. Therefore, users need to take more additional security measures. For example, users should assess the security levels per application scenario and implement monitoring, fault tolerance, and other necessary mechanisms to guarantee the security and stability of the kernel system. In terms of mandatory access control, *SELinux* is the ideal security subsystem for Linux. In its access control system, the process can only access the files that are required for its tasks. *SELinux* defines the access permissions for each user in the system, process, application, and file. It also uses a security policy to control the interaction among these entities. The security policy specifies how tightly or loosely the interaction can be. The *SELinux* module can be integrated into containers, which is the last line defense for containers. So with these three layered defenses, the containers security issues of Microservices-based fog applications could be largely mitigated.

8.2 | Data security

As shown in Figure 3, the ideal solution for the data protection could adopt the mixed encryption mechanism that utilizes shared key and public key encryption. To be specific, we advise using Advanced Encryption Standard (AES)[116] and RSA encryption algorithm[61] as the shared key encryption and public key encryption scheme, respectively. AES encryption algorithm is the current international standard of shared key encryption. AES encryption algorithm has the advantages of short key establishment time, good sensitivity and low memory requirements. RSA encryption algorithm is one of widely-used public key encryption algorithms, which is very hard for cryptanalysis. Although the public key encryption has higher security assurance than the shared key encryption, its performance is worse. Therefore, we suggest to combine the advantages of two encryption methods together to protect the communication data security. As shown in Figure 4, first, we utilize the public key encryption method to exchange the shared keys, which will be used by shared key encryption later. Because the method of public key encryption is safer, we add an additional layer of protection to the shared key, so that the shared key could not be intercepted by the attackers. We also use a certificate authentication mechanism for public keys to ensure the correctness of the shared key. Then, we use the private key to decrypt the encrypted shared key and use

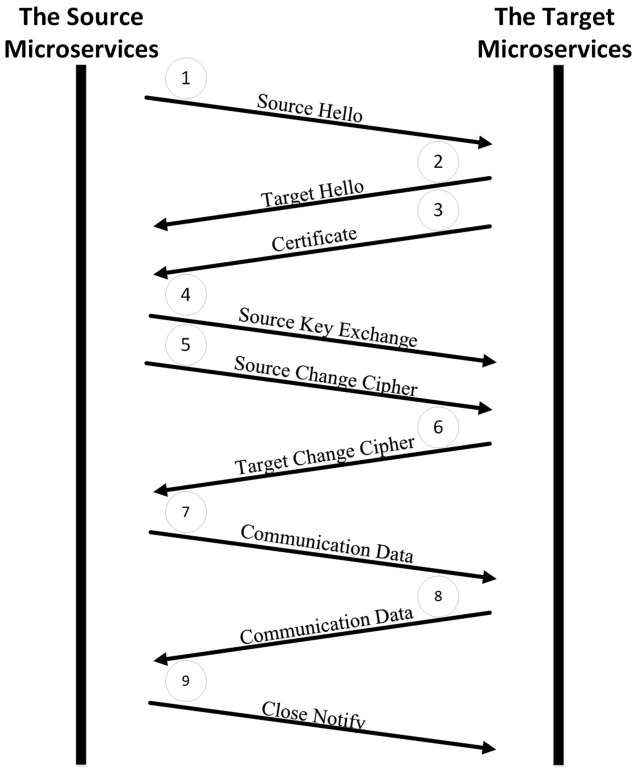

FIGURE 4 The ideal secure data communication process

shared key encryption method in the latter communication, which will ensure higher communication efficiency. So in this way, there is a double layer protection of data and it has less impact on communication efficiency. Once the data is intercepted during the communication, it is almost impossible to decipher the original communication data without obtaining a shared key. To protect the heterogeneous data source security in the Microservices architecture, we propose a supervisory system to monitor whether the data source is attacked or destroyed. In order to prevent data corruption or loss, we also need to regularly backup important data securely. The detailed steps of our proposed security communication are listed as follows.

Step 1: The source Microservices sends the *Source Hello* message to start communication.
Step 2: The target Microservices responds the *Target Hello* message to the source Microservices when receiving the *Source Hello* message.
Step 3: The target Microservices sends the *Certificate* message, which contains public key encryption certificate and the public key. The source Microservices can use the certificate to verify the correctness of the target Microservices public key.
Step 4: The source Microservices responds the *Source Key Exchange* message that contains the encryption key for the shared key, which is encrypted by the public key of target Microservices.
Step 5: The source Microservices sends the *Source Change Cipher* message. The message would notify the target Microservices that it will use the shared key encryption in the latter communication.
Step 6: The target Microservices uses the private key corresponding to the previous public key to decrypt the encrypted shared key and send the *Target Change Cipher* message to agree to use the shared key.
Step 7: The source Microservices starts secure communication as they send application data.
Step 8: The target Microservices responds with the application data.
Step 9: In the end, the source Microservices sends the *Close Notify* message to close the connection.

8.3 | Permission security

The small scale application can use the HTTP digest authentication mechanism that supports the insertion of user defined encryption algorithm, which can take further improvement for the security of APIs. The aforementioned method that utilizes the public key encryption to exchange information is also a digital authentication, which can verify whether the other side has the correct private key.

To secure permission, we can utilize HTTP digest authentication or the previously proposed public and private combined key encryption to protect digital authentication and authorization for the Microservices-based fog applications. Another option is to integrate OAuth 2 with *Spring Cloud Security* framework.[115] As shown in Figure 3, we think *Spring Cloud Security* is an ideal solution for permission security. *Spring Cloud Security* framework provides secure access control capabilities for the application. The framework provides several filters that can intercept requests and pass those requests to authentication and access control manager to guarantee security. It encapsulates some complex security configurations and supports a set of distributed system development framework for fundamental deployment and maintenance. It can be further improved to support Microservices-based fog application. OAuth 2 is a popular authentication mechanism, which fixes the security flaws of OAuth 1. In a nutshell, when we need to log on to an application, we can use a third-party application account to gain authorization through OAuth 2. OAuth 2 protects the account security of third-party application so that the crucial information could not be captured by other applications using the OAuthor user account. With OAuthor 2 integrated, *Spring Cloud Security* could protect the sensitive information of users to prevent user password from leaking. The authorization mechanism after authentication usually depends on the user role with the corresponding permission. *Spring Cloud Security* achieves a fine-grained authorization in that every authorized user has its corresponding permission scope.

8.4 | Network security

As shown in Figure 3, we adopt some measures against traditional network security issues and SDN technology to protect network security. For traditional network attacks (eg, DOS attack and MITM attack), we summarize corresponding defensive measures against traditional network security issues that can be solved by signature-based, automata-based, or simulation-based methods,[105,106,108,109,117-121] which we will skip details for brevity. However, there is still no perfect solution to address all the network security issues, some approaches are still needed to effectively mitigate them. For example, to mitigate the influence of DoS attack, adding validation or filtering out some network IP segments when visits reach a specific threshold may be a valid solution. The ARP spoofing is mainly caused by the forged MAC address in the local area network (LAN). Therefore, we can manually add IP/MAC address to the static ARP list to defend it. Even if there is a forged MAC address in the LAN, the computer will not be attacked by the ARP spoofing. We can limit the network segments and ports of the firewall in the LAN to filter and defend most network attack. In addition, although SDN could provide monitoring of network flow for Microservices, centralized control and openness of SDN will result in potential trust security risks.

9 | CONCLUSION

Although there are many advantages by using Microservices architecture, there are yet many practical and potential security problems, which have to be solved. Microservices is a relatively new software architecture so that its research is still limited in both industry and academic domain. The security of Microservices architecture still remains the primary concern. We have described the security issues and current solutions of Microservices for the service communication in four aspects: containers, data, permission, and network. Containers have vulnerability in the kernel leakage. User and enterprise private data could be intercepted during transmission. It is necessary to verify whether the service is running normally and whether it is compromised. Moreover, network attack and SDN security vulnerabilities also should be resolved. An ideal solution has been proposed to bridge the existing security gaps. In the future, we will consider to implement an integrated solution to secure Microservices-based fog applications with a balance among security level, application performance, and user experience.

ORCID

Dongjin Yu http://orcid.org/0000-0001-8919-1613
Xi Zheng http://orcid.org/0000-0002-2572-2355

REFERENCES

1. Hao Z, Novak E, Yi S, Li Q. Challenges and software architecture for fog computing. *IEEE Internet Comput*. 2017;21(2):44-53.
2. More P. Review of implementing fog computing. *Int J Res Eng Technol*. 2015.
3. Tao M, Ota K, Dong M. Foud: integrating fog and cloud for 5G-enabled V2G networks. *IEEE Netw*. 2017;31(2):8-13.
4. Sill A. The design and architecture of microservices. *IEEE Cloud Comput*. 2016;3(5):76-80.
5. Dong M, Ota K, Liu A. Preserving source-location privacy through redundant fog loop for wireless sensor networks. Paper presented at: 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM); 2015; Liverpool, UK.
6. boot2dockerio. Boot2Docker by Boot2docker. 2017. http://boot2docker.io/
7. Rancher. RancherOS. 2017. http://rancher.com/rancher-os/
8. Shi W, Dustdar S. The promise of edge computing. *Computer*. 2016;49(5):78-81.
9. Bonomi F, Milito R, Zhu J, Addepalli S. Fog computing and its role in the internet of things. Paper presented at: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, ACM; 2012; Helsinki, Finland.
10. Zheng X, Pan L, Yilmaz E. Security analysis of modern mission critical android mobile applications. Paper presented at: Proceedings of the Australasian Computer Science Week Multiconference; 2017; Geelong, Australia.
11. Lea G. Microservices security: all the questions you should be asking. 2015. http://www.grahamlea.com/2015/07/microservices-security-questions/
12. Linthicum DS. Practical use of microservices in moving workloads to the cloud. *IEEE Cloud Comput*. 2016;3(5):6-9.
13. Anthes G. Security in the cloud. *Commun ACM*. 2010;53(11):16-18.
14. Patanjali S, Truninger B, Harsh P, Bohnert TM. Cyclops: a micro service based approach for dynamic rating, charging and billing for cloud. Paper presented at: 2015 13th International Conference on Telecommunications (ConTEL); 2015; Graz, Austria.
15. Aliyu AL, Bull P, Abdallah A. A trust management framework for network applications within an SDN environment. Paper presented at: International Conference on Advanced Information Networking and Applications Workshops; 2017; Cracow, Poland.
16. Malavalli D, Sathappan S. Scalable microservice based architecture for enabling DMTF profiles. Paper presented at: International Conference on Network and Service Management; 2015; Barcelona, Spain.
17. Pathania N. Setting up Jenkins on Docker and Cloud. *Pro Continuous Delivery*. Berkeley, CA: Apress; 2017:115-143.
18. Wilde N, Gonen B, El-Sheikh E, Zimmermann A. Approaches to the evolution of SOA systems. In: Wilde N, Gonen B, El-Sheikh E, Zimmermann A, eds. *Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures*. Cham, Switzerland: Springer International Publishing; 2016;11:5-21.
19. Chappell D. Enterprise Service Bus. Paper presented at: 2007 Free and Open Source Software Conference; 2007; Skopje, Macedonia.
20. Singleton A. The Economics of microservices. *IEEE Cloud Comput*. 2016;3(5):16-20.
21. Zhu L, Bass L, Champlin-Scharff G. DevOps and its practices. *IEEE Softw*. 2016;33(3):32-34.
22. Savchenko DI, Radchenko GI, Taipale O. Microservices validation: Mjolnirr platform case study. Paper presented at: 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO); 2015; Opatija, Croatia.
23. Villamizar M, Garcés O, Castro H, et al. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. Paper presented at: 10th Computing Colombian Conference (10CCC); 2015; Bogota, Colombia.
24. Villamizar M, Zambrano A, Lang M, et al. Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. Paper presented at: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid); 2016; Cartagena, Colombia.
25. Sun L, Li Y, Memon RA. An open IoT framework based on microservices architecture. *China Commun*. 2017;14(2):154-162.
26. Krylovskiy A, Jahn M, Patti E. Designing a smart city internet of things platform with microservice architecture. Paper presented at: 2015 3rd IEEE International Conference on Future Internet of Things and Cloud (FiCloud); 2015; Rome, Italy.

27. Bao K, Mauser I, Kochanneck S, Xu H, Schmeck H. A microservice architecture for the intranet of things and energy in smart buildings. Paper presented at: Proceedings of the 1st International Workshop on Mashups of Things and APIs; 2016; Trento, Italy.
28. Alrawais A, Alhothaily A, Hu C, Cheng X. Fog computing for the internet of things: security and privacy issues. *IEEE Internet Comput*. 2017;21(2): 34-42.
29. Zimmermann A, Bogner J, Schmidt R, Jugel D, Schweda C, Möhring M. Digital enterprise architecture with micro-granular systems and services. Paper presented at: BIR Workshops; 2016; Prague, Czech Republic.
30. Bak P, Melamed R, Moshkovich D, Nardi Y, Ship H, Yaeli A. Location and context-based microservices for mobile and internet of things workloads. Paper presented at: 2015 IEEE International Conference on Mobile Services; 2015; New York, NY.
31. Gadea C, Trifan M, Ionescu D, Cordea M, Ionescu B. A microservices architecture for collaborative document editing enhanced with face recognition. Paper presented at: 2016 IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI); 2016; Timisoara, Romania.
32. Peinl R, Holzschuher F, Pfitzer F. Docker cluster management for the cloud-survey results and own solution. *J Grid Comput*. 2016;14(2):265-282.
33. Pahl C, Jamshidi P. Microservices: a systematic mapping study. Paper presented at: Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER); 2016; Rome, Italy.
34. Kratzke N, Quint PC. Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study. *J Syst Softw*. 2017;126:1-16.
35. Vase T. Integrating Docker to a Continuous Delivery Pipeline: A Pragmatic Approach. [Master's thesis]. 2016.
36. Bacis E, Mutti S, Capelli S, Paraboschi S. DockerPolicyModules: mandatory access control for docker containers. Paper presented at: 2015 IEEE Conference on Communications and Network Security (CNS); 2015; Florence, Italy.
37. Checconi F, Cucinotta T, Faggioli D, Lipari G. Hierarchical multiprocessor CPU reservations for the Linux kernel. Paper presented at: 2009 Proceedings of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009); 2009; Dublin, Ireland.
38. Dua R, Raja AR, Kakadia D. Virtualization vs containerization to support PaaS. Paper presented at: 2014 IEEE International Conference on Cloud Engineering (I2CE); 2014; Boston, MA.
39. Manu AR, Patel JK, Akhtar S, Agrawal VK, Murthy KNBS. A study, analysis and deep dive on cloud PaaS security in terms of docker container security. Paper presented at: 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT); 2016; Kollam, India.
40. Biederman EW. Multiple instances of the global Linux namespaces. Paper presented at: Proceedings of the Linux Symposium; 2006; Ottawa, Canada.
41. Bui T. Analysis of docker security. arXiv preprint arXiv:1501.02967. 2015.
42. Chelladhurai J, Chelliah PR, Kumar SA. Securing docker containers from denial of service (DoS) attacks. Paper presented at: 2016 IEEE International Conference on Services Computing (SCC); 2016; San Francisco, CA.
43. Soltesz S, Pötzl S, Fiuczynski ME, Bavier A, Peterson L. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. Paper presented at: ACM SIGOPS Operating Systems Review; 2007; New York, NY.
44. Gupta U. Comparison between security majors in virtual machine and Linux containers. arXiv preprint arXiv:1507.07816. 2015.
45. Truyen E, Landuyt DV, Reniers V, Rafique A, Lagaisse B, Joosen W. Towards a container-based architecture for multi-tenant SaaS applications. Paper presented at: Proceedings of the 15th International Workshop on Adaptive and Reflective Middleware; 2016; Trento, Italy.
46. Combe T, Martin A, Di Pietro R. To docker or not to docker: a security perspective. *IEEE Cloud Computing*. 2016;3(5):54-62.
47. Jian Z, Chen L. A defense method against docker escape attack. Paper presented at: Proceedings of the 2017 International Conference on Cryptography, Security and Privacy; 2017; Wuhan, China.
48. Gao X, Gu Z, Kayaalp M, Pendarakis D, Wang H. ContainerLeaks: emerging security threats of information leakages in container clouds. Paper presented at: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE; 2016; Denver, CO.
49. Manu AR, Patel JK, Akhtar S, Agrawal VK, Murthy KNBS. Docker container security via heuristics-based multilateral security-conceptual and pragmatic study. Paper presented at: 2016 International Conference on Circuit Power and Computing Technologies (ICCPCT); 2016; Kollam, India.
50. Dillon T, Wu C, Chang E. Cloud computing: issues and challenges. Paper presented at: 2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA); 2010; Perth, Australia.
51. Wei Y, Blake MB. Service-oriented computing and cloud computing: challenges and opportunities. *IEEE Internet Comput*. 2011;14(6):72-75.
52. Gegick M, Barnum S. Reluctance to trust. 2005. https://www.us-cert.gov/bsi/articles/knowledge/principles/reluctance-to-trust
53. Chong F, Carraro G, Wolter R. Multi-tenant data architecture. 2006:14-30. https://msdn.microsoft.com/en-us/library/aa479086.aspx
54. Juels A, Kaliski BS. PORs: proofs of retrievability for large files. Paper presented at: 14th ACM Conference on Computer and Communications Security; 2007; Alexandria, VA.
55. Shah MA, Baker M, Mogul JC, Swaminathan R. Auditing to keep online storage services honest. Paper presented at: Workshop on Hot Topics in Operating Systems (HotOS'07); 2007; San Diego, CA.
56. Wang C, Ren K, Lou W, Li J. Toward publicly auditable secure cloud data storage services. *IEEE Netw*. 2010;24(4).
57. Wang C, Wang Q, Ren K, Lou W. Privacy-preserving public auditing for data storage security in cloud computing. Paper presented at: 2010 Proceedings of IEEE INFOCOM; 2010; San Diego, CA.
58. Esposito C, Castiglione A, Choo KKR. Challenges in delivering software in the cloud as microservices. *IEEE Cloud Comput*. 2016;3(5):10-14.
59. Callegati F, Giallorenzo S, Melis A, Prandini M. Data security issues in MaaS-enabling platforms. Paper presented at: 2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging A Better Tomorrow (RTSI); 2016; Bologna, Italy.
60. Subashini S, Kavitha V. A survey on security issues in service delivery models of cloud computing. *J Netw Comput Appl*. 2011;34(1):1-11.
61. Somani U, Lakhani K, Mundra M. Implementing digital signature with RSA encryption algorithm to enhance the data security of cloud in cloud computing. Paper presented at: 2010 1st International Conference on Parallel Distributed and Grid Computing (PDGC); 2016; Solan, India.

62. Wang G, Liu Q, Wu J. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. Paper presented at: Proceedings of the 17th ACM Conference on Computer and Communications Security; 2010; Chicago, IL.
63. Agarwal A, Tirumalai SK, Sriramadhesikan K. Data Encryption Service; 2017.
64. Cheung KC, Peel C, Happe S. Method and System for Credential Management and Data Encryption for iOS Based Devices; 2013.
65. Chen X, Li J, Ma J, Tang Q, Lou W. New Algorithms for Secure Outsourcing of Modular Exponentiations. *IEEE Trans Parallel Distrib Syst*. 2014;25(9):2386-2396.
66. Chen X, Zhang F, Susilo W, Tian H, Li J, Kim K. Identity-based chameleon hashing and signatures without key exposure. *Inform Sci*. 2014;265:198-210.
67. Chen X, Li J, Huang X, Ma J, Lou W. New publicly verifiable databases with efficient updates. *IEEE Trans Dependable Secure Comput*. 2015;12(5):546-556.
68. Chen X, Li J, Weng J, Ma J, Lou W. Verifiable computation over large database with incremental updates. *IEEE Trans Comput*. 2016;65(10):3184-3195.
69. Li J, Chen X, Li M, Li J, Lee PP, Lou W. Secure deduplication with efficient and reliable convergent key management. *IEEE Trans Parallel Distrib Syst*. 2014;25(6):1615-1625.
70. Li P, Li J, Huang Z, Gao CZ, Chen WB, Chen K. Privacy-preserving outsourced classification in cloud computing. *Clust Comput*. 2017:1-10.
71. Li J, Zhang Y, Chen X, Xiang Y. Secure attribute-based data sharing for resource-limited users in cloud computing. *Comput Secur*. 2018;72:1-12.
72. Bu K, Xiao B, Qian Y. High performance and security in cloud computing. *Concurr Comput Pract Exp*. 2017;29(19).
73. Gegick M, Barnum S. Least privilege. 2005. https://www.us-cert.gov/bsi/articles/knowledge/principles/least-privilege
74. Definition DAP. Lightweight Directory Access Protocol. RFC 2013; 7(89).
75. Murri R. Grid Security Infrastructure; 2005.
76. Pearlman L, Welch V, Foster I, Kesselman C, Tuecke S. A community authorization service for group collaboration. Paper presented at: 2002 Proceedings of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks; 2002; Monterey, CA.
77. Li W, Mitchell CJ. Analysing the security of Google's implementation of OpenID connect. In: Caballero J, Zurutuza U, Rodriguez R, eds. *Detection of Intrusions and Malware, and Vulnerability Assessment*. Berlin, Germany: Springer-Verlag; 2015.
78. Thanh TQ, Covaci S, Magedanz T, Gouvas P, Zafeiropoulos A. Embedding security and privacy into the development and operation of cloud applications and services. Paper presented at: 2016 17th Telecommunications Network Strategy and Planning Symposium; 2016; Montréal, Canada.
79. Ebert C, Gallardo G, Hernantes J, Serrano N. DevOps. *IEEE Softw*. 2016;33(3):94-100.
80. Pan L, Zheng X, Chen HX, Luan T, Bootwala H, Batten L. Cyber security attacks to modern vehicular systems. *J Inf Secur Appl*. 2017;36:90-100.
81. Liu Z, Dong M, Zhou H, Wang X, Ji Y, Tanaka Y. Device-to-device assisted video frame recovery for picocell edge users in heterogeneous networks. Paper presented at: 2016 IEEE International Conference on Communications (ICC); 2016; Kuala Lumpur, Malaysia.
82. Guo L, Dong M, Ota K, et al. A secure mechanism for big data collection in large scale internet of vehicles. *IEEE Internet Things J*. 2017;4(2):601-610.
83. Zhang L, Wei L, Huang D, Zhang K, Dong M, Ota K. MEDAPs: secure multientities delegated authentication protocols for mobile cloud computing. *Secur Commun Netw*. 2016;9(16):3777-3789.
84. Wu J, Ota K, Dong M, Li C. A hierarchical security framework for defending against sophisticated attacks on wireless sensor networks in smart cities. *IEEE Access*. 2016;4:416-424.
85. Qi H, Dong M. New advances in future network technologies. *Concurr Comput Pract Exp*. 2017;29(16).
86. Venkataramulu S, Rao DCVG. Various solutions for address resolution protocol spoofing attacks. *Int J Sci Res Pub*. 2013;3(7):1.
87. Haataja K, Toivanen P. Two practical man-in-the-middle attacks on bluetooth secure simple pairing and countermeasures. *IEEE Trans Wirel Commun*. 2010;9(1).
88. Tan Z, Jamdagni A, He X, Nanda P, Liu RP. A system for denial-of-service attack detection based on multivariate correlation analysis. *IEEE Trans Parallel Distrib Syst*. 2014;25(2):447-456.
89. Abad CL, Bonilla RI. An analysis on the schemes for detecting and preventing ARP cache poisoning attacks. Paper presented at: 2007 27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07); 2007; Washington, DC.
90. Hwang H, Jung G, Sohn K, Park S. A study on MITM (man in the middle) vulnerability in wireless network using 802.1 X and EAP. Paper presented at: 2008 International Conference on Information Science and Security (ICISS); 2008; Seoul, South Korea.
91. Meyer U, Wetzel S. A man-in-the-middle attack on UMTS. Paper presented at: Proceedings of the 3rd ACM Workshop on Wireless Security; 2004; Philadelphia, PA.
92. Stasiak M, Glabowski M, Wisniewski A, Zwierzykowski P. Universal mobile telecommunication system. *Modelling and Dimensioning of Mobile Wireless Networks: From GSM to LTE*; 2013:15-42.
93. Nam SY, Kim D, Kim J. Enhanced ARP: preventing ARP poisoning-based man-in-the-middle attacks. *IEEE Commun Lett*. 2010;14(2):187-189.
94. Alicherry M, Keromytis AD. Doublecheck: Multi-path verification against man-in-the-middle attacks. Paper presented at: 2009 IEEE Symposium on Computers and Communications, (ISCC); 2009; Sousse, Tunisia.
95. Joshi Y, Das D, Saha S. Mitigating man in the middle attack over secure sockets layer. Paper presented at: 2009 IEEE International Conference on Internet Multimedia Services Architecture and Applications (IMSAA); 2009; Bangalore, India.
96. Ducic D. Features of the Secure Sockets Layer Protocol. [Thesis]. 2016.
97. Boraten T, Kodi A. Mitigation of hardware trojan based denial-of-service attack for secure NoCs. *J Parallel Distrib Comput*. 2017;111:24-38.
98. Khan S, Gani A, Wahab AWA, Singh PK. Feature selection of denial-of-service attacks using entropy and granular computing. *Arabian J Sci Eng*. 2018;43:499-508.
99. Chang RKC. Defending against flooding-based distributed denial-of-service attacks: a tutorial. *IEEE Commun Mag*. 2002;40(10):42-51.

100. Hussain A, Heidemann J, Papadopoulos C. A framework for classifying denial of service attacks. Paper presented at: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications; 2003; Karlsruhe, Germany.
101. Gerlach S, Lebert D. Mitigating Denial of Service Attacks; 2017.
102. Anderson T, Roscoe T, Wetherall D. Preventing Internet denial-of-service with capabilities. *ACM SIGCOMM Comput Commun Rev*; 2004;34(1):39-44.
103. Ahmadvand M, Ibrahim A. Requirements reconciliation for scalable and secure microservice (de)composition. Paper presented at: 2016 24th IEEE International Requirements Engineering Conference Workshops; 2016; Beijing, China.
104. Radhappa H, Pan L, Zheng JX, Wen S. Practical overview of security issues in wireless sensor network applications. *Int J Comput Appl*. 2017:1-12.
105. Jiang J, Wen S, Yu S, Xiang Y, Zhou W. Identifying propagation sources in networks: state-of-the-art and comparative studies. *IEEE Commun Surv Tutorials*. 2017;19(1):465-481.
106. Wang Y, Wen S, Xiang Y, Zhou W. Modeling the propagation of worms in networks: a survey. *IEEE Commun Surv Tutorials*. 2014;16(2):942-960.
107. Stojmenovic I, Wen S, Huang X, Luan H. An overview of fog computing and its security issues. *Concurr Comput Pract Exp*. 2016;28(10):2991-3005.
108. Wen S, Haghighi MS, Chen C, Xiang Y, Zhou W, Jia W. Sword with two edges: propagation studies on both positive and negative information in online social networks. *IEEE Trans Comput*. 2015;64(3):640-653.
109. Wen S, Zhou W, Zhang J, Xiang Y, Zhou W, Jia W. Modeling propagation dynamics of social network worms. *IEEE Trans Parallel Distrib Syst*. 2013;24(8):1633-1643.
110. Sun Y, Nanda S, Jaeger T. Security-as-a-service for microservices-based cloud applications. Paper presented at: 2015 7th IEEE International Conference on Cloud Computing Technology and Science (CloudCom); 2015; Vancouver, Canada.
111. Barnum S, Gegick M, Michael C. Defense in depth. 2005. https://www.us-cert.gov/bsi/articles/knowledge/principles/defense-in-depth
112. Gegick M, Barnum S. Never assuming that your secrets are safe. 2005. https://www.us-cert.gov/bsi/articles/knowledge/principles/never-assuming-that-your-secrets-are-safe
113. Gegick M, Barnum S. Securing the weakest link. 2005. https://www.us-cert.gov/bsi/articles/knowledge/principles/securing-the-weakest-link
114. Callegati F, Cerroni W, Ramilli M. Man-in-the-middle attack to the HTTPS protocol. *IEEE Secur Priv*. 2009;7(1):78-81.
115. Spring. Spring Cloud Security. 2017. https://cloud.spring.io/spring-cloud-security/
116. Barik RC, Changder S, Sahu SS. Image texture-based new cryptography scheme using advanced encryption standard. In: Behera H, Mohapatra D, eds. *Computational Intelligence in Data Mining*. Singapore: Springer; 2017.
117. Zheng X, Julien C, Kim M, Khurshid S. Perceptions on the state of the art in verification and validation in cyber-physical systems. *IEEE Syst J*. 2015;11.
118. Zheng X, Julien C, Podorozhny R, Cassez F, Rakotoarivelo T. Efficient and scalable runtime monitoring for cyber-physical system. *IEEE Syst J*. 2016.
119. Zheng X, Julien C, Chen H, Podorozhny R, Cassez F. Real-Time Simulation Support for Runtime Verification of Cyber-Physical Systems. *ACM Trans Embed Comput Syst (TECS)*. 2017;16(4):106.
120. Zheng X, Julien C, Podorozhny R, Cassez F. Braceassertion: runtime verification of cyber-physical systems. Paper presented at: 2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), IEEE; 2015; Dallas, Texas.
121. Zheng X, Pan L, Chen H, Di Pietro R, Batten L. A testbed for security analysis of modern vehicle systems. Paper presented at: 2017 IEEE Trustcom/BigDataSE/ICESS, IEEE; 2017; Sydney, Australia.

How to cite this article: Yu D, Jin Y, Zhang Y, Zheng X. A survey on security issues in services communication of Microservices-enabled fog applications. *Concurrency Computat Pract Exper*. 2019;31:e4436. https://doi.org/10.1002/cpe.4436