Study of Container-Based Virtualisation and Threats in Fog Computing

Poornima Mahadevappa and Raja Kumar Murugesan(✉)

Taylor's University, Subang Jaya, Malaysia
poornimamahadevappa@sd.taylors.edu.my,
rajakumar.murugesan@taylors.edu.my

Abstract. Fog computing has provided a virtualised platform by extending cloud services closer to IoT devices. This virtualised platform includes virtual servers, storage, or data centres by helping to minimise the offloading delay in execution and data access. This feature makes fog computing best suitable for applications that require low latency and real-time interaction. The role of virtualisation has predominantly contributed to achieving the best results for service providers and satisfy the user's experience. In fog computing, container virtualisation aids for the effective containment of resource usage. Containers here in fog data centres reduce energy consumptions, memory, CPU, and cost, thereby being efficient for resource-limited fog computing applications. However, in the applications, containers can directly access and communicate with the host kernels, and this gives an easy gateway for the attacker to invade containers. If we do not address this, then the attacker can attack the host application, control data on the containers, or access the docker engine. This paper studies a brief understanding of container-based virtualisation in fog computing and container-based attacks and discusses possible solutions. Besides, this study guides on how we can implement a secure container-based fog application with a balance in security, performance, and user experience.

Keywords: Fog computing · Container · Virtualisation · Security

1 Introduction

Fog computing is a distributed computing paradigm that has moved computation and storage closer to the end of IoT devices to reduce the network overload and compute the data collected at a faster rate. They provide an extension of cloud computing services through the virtualised platform called the fog layer [1]. The virtualisation platform allows multiple virtual machines to co-exist on a physical hardware source by providing virtual servers, storage, or data centres. The virtual machines are at the closer proximity of IoT devices with all its user's data to minimise the offloading delay in execution and data access [2]. Therefore, fog computing is suitable for applications that require low latency and real-time interactions [3]. The global fog computing market can grow up to USD 768 million by 2025, as fog computing is complementing the cloud by extending

M. Anbar et al. (Eds.): ACeS 2020, CCIS 1347, pp. 535–549, 2021.
https://doi.org/10.1007/978-981-33-6835-4_36

its features closer to the source of end-users [4]. In real-time, many applications have gained benefits for the business market needs to aggregate business users, designing new business scenarios, and create value. The best real-time example would be Smart Fog Hub Service (SFHS) deployed in Cagliari Elmas airport, Italy, or Smart Boat application [5]. In these applications, data collected from the edge sensors then passed to edge fog devices, which has computing power, storage capability to process data and run management functions. It also include cloud to handle scalable computing for massive data processing and gateways to connect edge fog and cloud [6].

In fog computing, Virtual Machines (VM) are responsible for providing computing capacity, i.e., processing and storage for the applications. They play a vital role in providing the decision-making process to ensure the best results for users and service providers. VM are the critical components of virtualisation, and many fog-based applications use container-based virtualisation for effective accountability and containment of resource usage [7]. The containers provide an efficient task scheduling without any significant overheads in fog data centres. Deploying containers in fog data centres reduce energy consumptions, memory, CPU, and cost. Docker is the most popular container framework deployed nowadays due to its many advantages like scalability, portability, density, and rapid delivery [8]. However, in the applications, containers can directly access and communicate with the host kernels and thereby allowing the attackers to access the kernel directly. During this process, the attackers can attack host applications, control data on the containers, or access the docker engine. Attackers establish this through any common network attacks like Malware, DoS, Spoofing, Malicious code infection, crypto-jacking, or container-specific attacks through creating an infected image, handling privileges through kernels [9]. Therefore, the main problem on a containerised fog-based application can be fog servers being vulnerable to the risk of surveillance of user's data by attackers, spying communication channels, and hacking storage authentication credibility. Therefore, this paper contributes towards understanding fog-based container virtualisation, analysing the security threats on containers, and finally, propose some possible solutions to address these threats on containers to secure data.

The rest of this paper is organised as follows: Sects. 2: gives a brief study of virtualisation and container-based virtualisation in fog computing. Section 3: Analyses the threats and attacks on data containers. Section 4 discusses relevant work, while Sect. 5 discusses some proposed solutions in Sect. 5 followed by the conclusion.

2 Virtualisation in Fog Based Application

Virtualisation is a process of creating a virtual version of computer hardware, software, network resources, or a server. Virtual Machines (VM) are the key components of virtualisation, and hypervisor is the software that generates and runs VM. The hypervisor operates one or more VM on the host machine [10]. Fog computing uses server virtualisation and network virtualization: Server virtualisation plays a vital role in cloud and fog computing by encapsulating the hardware server from the software server [11]. They include OS, applications, and storage. While in network virtualisation, the users are linked directly to a company network or resources with no physical link, called Virtual Private Network (VPN). VPN can allow the users to link to a network and access the resources from any internet linked network [12].

In hypervisor-based virtualisation, the VM takes high time for OS (Operating System) booting increasing the performance overhead for the application level resource scheduling. The hypervisors have no idea about the applications running on VM and block numerous optimizations like caching and perfecting. This lack of information was a significant drawback for application-specific fog-based use cases. Container-based virtualisation emerged to address the issue of hypervisors. Now it is a growing technology in the cloud, fog, and edge computing. The main advantage of container virtualisation is lightweight, scalability, and operational flexibility. Container virtualisation shares the kernel with multiple containers while the hypervisor does not maintain an OS instance, as shown in Fig. 1, so the disk images are smaller than VM. As container virtualisation is at the OS level, so they are more efficient than the hypervisor [13].

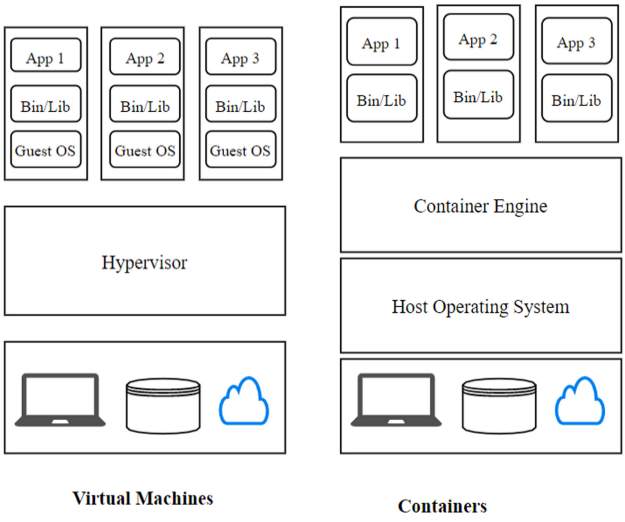

Fig. 1. Comparison of virtual machines and containers

Containers are an abstraction of package code and dependencies at the application layer. They provide resource isolation, resource control, and predictable behaviour for the containerised application. LXC and Docker are the most commonly used containers, while docker is a microservice-based container framework that builds, package, and run the application inside the containers[1]. Applications are in the form of images with all the libraries and executable files. These images are in a public repository, and they can be pulled on the local compute fog nodes. Using a base image such as ubuntu, centos, busy box, or fedora, we can create containers on the fog nodes. The second and third layers on the containers are image 1 and image 2, read-only images. The images above these are thin writable layers often called container layer; all the changes made like writing, deleting, or modifying the existing files are on the writable layer. These layers are as shown in Fig. 2. [14]. The fog nodes hosted using these containers on a fog layer include application data, necessary metadata, information about fog node state, and policies for

[1] https://docs.docker.com/.

security and configurations [15]. The fog nodes are a resource-rich server, routers, or an access point with compute network and compute resource capabilities. Containers run appropriate microservices on the fog nodes.

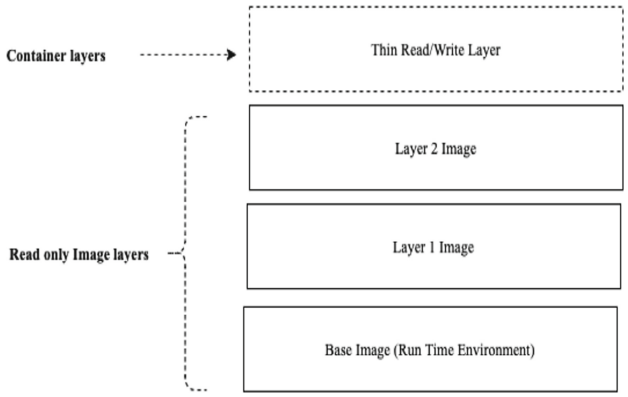

Fig. 2. Container image layer

2.1 Advantages of Container-Based Virtualisation

The adoption of lightweight container virtualisation in fog computing has provided high manageable and interoperable ways to create fog nodes with limited resources availability. This section presents how fog middleware has benefited through this configuration:

Container-Based Macro Operations: Deploying macro operation is general-purpose fog nodes that have a standard gateway configuration. The possible macro operations include inputs, outputs, storage, networking, and computations, as shown in Fig. 3 and are called fog node skeleton. Every fog node specifies the list of macro services and the skeleton to achieve the final desired behaviour. Every time new functionality is released, we can upload a new container version suitable for long term container management. As each container is independent and isolated, it is easy to update or upload based on the requirement. This property allows to dynamically create fog nodes for a particular application based on the macro operations. For example, in compute-intensive applications, we can employ more compute-powerful nodes, rather than storage-intensive applications where we need to enhance database operations [16].

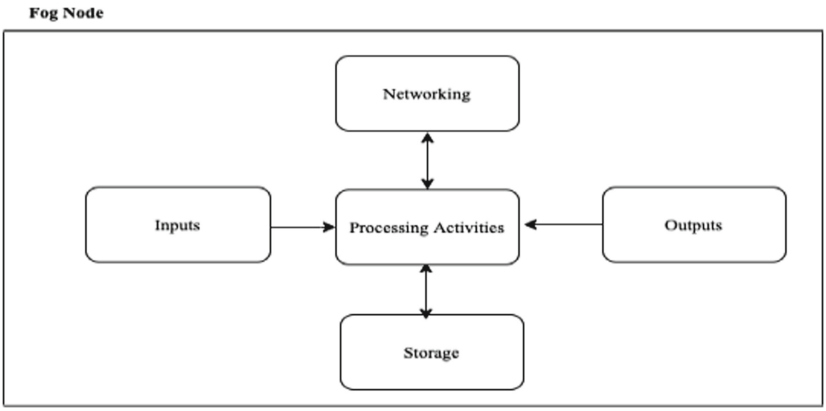

Fig. 3. Fog node skeleton

1. **Resource provisioning**: Container virtualisation enables resource provisioning in fog data centers. The memory footprint of the containers is smaller than the hypervisor; multiple containers can be deployed easily on the host machines. As the containers use a base image, restarting the container upon migration requires building a new writable container layer, as shown in Fig. 1 rather than restarting the OS. Apart from the list of available fog nodes with their capacity is listed by the discovery server, this would be more significant for reducing latency due to resource provisioning [7].
2. **Load balancing:** Load balancing in containers helps to maintain traffic in the containerised applications. When we deploy containers across clusters on fog nodes or fog servers, the load balancer will allow accessing multiple containers through the same host [17]. Kubernetes or Docker Swarm can be used to run on every fog node and load balance the request across the containers.
3. **Develop once, deploy everywhere flexibility:** Container-based services do not depend on the given platform, language, or specific application; instead, it gives code portability. This feature develops microservices operation and communicates using the Application Programming Interface (API). Furthermore, this can assemble into an extensive application while updating and maintaining applications individually. Therefore it is easy to add, improve, or roll back functionality without damaging the application or delaying development [18]**.**

3 Security Threats

Containers reduce the computational delay from development to deployment state with the capabilities like namespace, cgroup, and union file system. These capabilities provide secure isolation between the containers and a significant amount of network, especially in docker containers. However, they are vulnerable to the risk of hacking data on fog servers due to lack of decryption credentials or data communication channel can have a threat of sniffing [19]. Therefore, this section studies and analyses the common security threats on containers.

- **Container Escape Attack** – Container share kernels with the host through namespace and cgroups. When running some kernel specific code inside the containers, the container may crash or escalate the user's privilege to root privileges and get complete control of the host. This type of attack is called an escape attack. There can be two types of escape attack one through switching namespace and the other through modifying shared memory. The code below shows the escape attack through the task struct. The nsproxy namespace contains information about the process. The task struct gets the address of the parent process and initialises the PID to the real parent process's task struct of the host. Next, initialise the fs address of the task_struct and mount the writable container layer, thereby converting the host namespace by "switch task namespace" [20].

```
struct task_struct{
...
 pid_t pid;
 struct task_struct __rcu *real_parent;
 struct fs_struct *fs;
 /* namespaces */
 struct nsproxy *nsproxy;
...
}
```

Dirty COW, "Copy-On-Write," is the typical escape attack on the container; using this, an attacker can exploit write to the files as the root user.

- **Dangling Volume** – Active containers store data in a writable container layer; when we delete a container, this layer gets deleted. However, using Bind mount and volumes, data can persist even after deleting the containers. Bind mount works by mounting a file or directory from the host to the containers. Volumes, on the other hand, can mount data on several containers simultaneously. Moreover, deleting a container will not erase the mounted volume [21]. When an application is running on multiple containers and containers are not removed using the *docker prune* command. The attacker can access the data on any containers through kernel communication.
- **Shell shock and backdooring images** – Container images are deployed using a base image from a public repository. Docker base images used from docker hub[2] repositories verify the users by creating an account in the repositories with a signed manifest. In such a case, an attacker can have a signed manifest and create an infected image [22]. The infected image is backdooring images, and unknown developers can use them. Similarly, Shellshock is a type of privilege escalation through malicious code injection to the environment variables. This injection can lead to elevated privileges in the system [23].

[2] https://hub.docker.com/.

Apart from the above security threats, there are common security threats like Denial of Service (DoS), spoofing, Media Access Control (MAC) flooding, or port flooding. These threats can affect containers by using a large number of resources, compromise the container through man-in-the-middle, or degrade the performance [24]. For latency-sensitive and resource-limited fog-based applications, these threats can cause severe drawbacks by exploiting data on the containers, conflict on shared resources on fog nodes.

4 Relevant Work

4.1 Container-Based Systems in Fog Computing

Virtualisation has brought a promising swift and efficient deployment in computing infrastructure. In fog computing, virtualisation has gained more advantage, and many works show how we can achieve flexibility and efficiency through virtualisation. Container-based virtualisation gives the advantage of deploying the application anywhere. Some of the container-based systems and architectures are discussed in this section and are tabulated in Table 1. The multi-cloud multi fog architecture by Luo *et al.* is based on container based virtualisation. This architecture includes two service models, a temporary service model and a long term model based on containers resource units. They improve the service efficiency by handling real-time request and response services by temporary service model; while long term service handles subscribing and publishing service. They contributed significantly toward improving resource utilisation of fog nodes and reduces service dealy [25]. In [26], automatic data stream processing to an application is achieved through container-based software architecture in the fog computing network. Here fog node controllers and applications are deployed as a docker container, and they can be dynamically scaled up and down to allocate resources to the fog nodes. This architecture provides a full advantage for intra fog node and inter fog node interactions. In inter fog node interaction set of data is fed from the data providers to the application and data consumers provides the real-time results through data analytics. And in intra fog node interaction filtering of data received from data providers by discarding the irrelevant data and selection of appropriate data for the computation from the received data. Similarly, the fogflow framework for smart city application uses docker containers to allocate data processing tasks to worker fog nodes through topology master. The topology master knows the computational abilities of each nodes and establishes data flow across all the tasks and generate data streams [27].

To handle these container-based applications, Puliafito *et al.* has developed MobfogSim simulator to manage the services that are implemented as containers. MobfogSim is the only simulator available to simulate data streams in the containers for fog computing. It evaluates container migration techniques in fog computing [28]. All this work shows that containers have been appropriate solutions in fog computing. The key aspect that we can achieve using container-based virtualisation is the elasticity and multi-tenants. The number of resources like CPU or memory required by fog nodes during processing may scale up or down, so using containers, we can allocate them dynamically. Likewise, when multiple programs are running on the same host, they can

reduce the performance of the application. Still, by using an isolated docker container, we can accomplish multi-tenant without interfering with each other.

Table 1. Related work on container-based virtualisation in fog computing

Ref	Technique	Methods	Evaluation
[26]	Autonomic for data stream processing using container-based support	Intra fog scenario – Docker containers share the limited physical resources to stream data among fog nodes and controllers Inter fog scenario - dockers freeze and restore application on the fog nodes	They provide great flexibility to deploy fog nodes in geographically distributed location and establish interaction among them with efficient service latency
[25]	Multi-cloud multi fog architecture using container-based virtualisation	Two service models based on containers are used: Temporary service model and long-term service model Scheduling algorithm based on energy balancing to prolong the life of WSN	This work shows that containers are better than VM as they reduce resource utilisation and service delay The scheduling algorithm balances the energy consumption of terminal devices without increasing the data transmission time
[28]	Simulator to handle mobile container migration	Simulate speed of movement, the network connection between source and destination fog nodes and data volume to transmit	Service migration solutions to support the mobility of fog nodes
[27]	FogFlow frame for smart city platform using the container for task deployment	Topology master nodes assign data processing task to the worker nodes based on their computational abilities	Provide elastic IoT services quickly on cloud and edge services

4.2 Security in Containers Based Virtualisation in Fog Computing

The container technology is not fully matured due to inadequate security protocols which lead to the risk of security threats to the container-based system. Some of the work towards security issues are addressed in Table 2 with their proposed possible solutions. As shown in Table 2, many attacks are handled with respect the docker containers due to its flexibility towards deploying anywhere without any modifications. The attacks addressed can be categorized as insider or outsider attacks, kernel attacks and network attacks. Most of the attacks here are due to kernel vulnerabilities as the container provide OS-level virtualisation and the applications running on the OS can have direct access to the kernel. Another major attack is due to image vulnerabilities as the images are

used from the public repositories; there are chances of tampering with images through insecure channels [29]. The proposed solutions from these works are providing OS-level virtualisation through cgroups, IPC isolation, file isolation and so on [24, 30]. But these features can be exploited through network attacks like disable cgroup activation, deny the service to docker socket or mounting sensitive container directories. In the implementation of AppArmor, Unikernels is a security enhancement model that can be used to establish security in the containers. And lastly, establishing container security through orchestrators like Kubernetes or Docker swarm may be a better security options but security issue in the orchestrator must be investigated separately.

Table 2. Relevant work in container security issues

Ref	Description	Attacks considered	Solution
[24]	Reviewed security concerns and solutions for four container use cases	Insider attacks, inter container attack, host, and semis host attacks	A software solution like namespace, CGroup and seccomp Hardware solution like trusted platform modules
[22]	Review of security vulnerabilities in docker containers	Insecure configurations, Linux kernel attacks, inside image attacks, attacks during image decompression and storage process	Alternative to dockers - Unikernels
[31]	Attacks on multi-container applications and internal database server	Identifies how an attacker can exploit typical application system and network through containers	NA
[32]	Addresses insider and outsider attacks in docker containers with mitigating strategy	Kernel exploit, DoS, container breakouts, poisoned images, compromised secrets, man in the middle, ARP spoofing	Access control policy, secure deployment guidelines, network namespace, logging or auditing, SELinux/AppArmor, daemon privilege and security audit
[29]	Vulnerabilities that affects the usage of docker	Insecure local configuration, weak local access control and vulnerabilities in the image distribution process	Establish better isolation and remove host dependencies through orchestration
[30]	The security issue in container-based virtualisation techniques	DoS and privilege escalation attack	Process isolation, filesystem isolation, device isolation, IPC isolation, and network isolation

All the above studies imply that container-based applications are gaining popularity and evolve predominantly in the next future. There are many numbers of studies based on docker containers as they are offering a lightweight and efficient way to deploy the packages to all the applications. Many works have addressed potential security issues and given mitigation strategies to address these issues. The precautionary measures can offer a more secure and reliable container platform for future application development. Thereby we can ensure the more significant security in Docker containers.

5 Proposed Solution

Using Unikernels – Unikernels are relatively a new way to create quickly and deploy the virtual machines without much functional and operational overhead. Compare to the containers and virtual machine in Fig. 1 unikernels are more compressed by leaving out the most unnecessary parts. Unikernels have replaced the bulky virtual machines to the hypervisors as shown in the Fig. 4. They are lightweight with the size of few MB (MegaBytes), compact, single address space, memory safe and single-purpose appliance. When unikernels deployed to the cloud platform, they are compiled into a standalone kernel and sealed against modification [33]. Goethals *et al.* has compared docker versus unikernels for execution time and memory footprints. The results show that unikernel in java and python performed 16% better and 38% better in Go than the containers. At the same time, python unikernels reach 50% efficiency. Another remarkable result is device drivers in the REST API are less complex in Unikernels. In case of memory utilisation, Unikernels have an extra overhead of a kernel in each application, but they are quite less than a large OS running on a few containers [34].

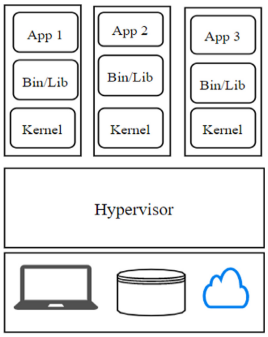

Fig. 4. Unikernels

Authorising and Monitoring Containers – Containers are prone to more kernel vulnerabilities. Although docker has its design and strategies for security, adopting them to fog computing, need some external security issues expanded to the containers. Malicious users can reach the kernel and crash the host, so including periodic authorising and monitoring of containers will be a good practice to track the path of all the host kernels.

This process can ensure that the containers are protected and monitored during runtime [35]. Currently, there are much monitoring and visualising tools that are used to analyse the stored data in IoT and cloud computing. These tools gather the data in InfluxDB or Prometheus with the programming tools with the Grafana monitoring system at the remote-control centres [36]. These tools can be used in fog computing to analyse data stored in computational fog nodes and identify if any events trigger the containers. The conceptual data flow in this tool is as shown in the Fig. 5. It includes the multiple docker containers that are running on the application; the data on these containers are saved on Prometheus local storage. These data are visualised in Grafana tool based on the metrics like CPU usage, network bandwidth, latency and so on. If there is any change in the metrics, then an alert message can be sent through email or slack.

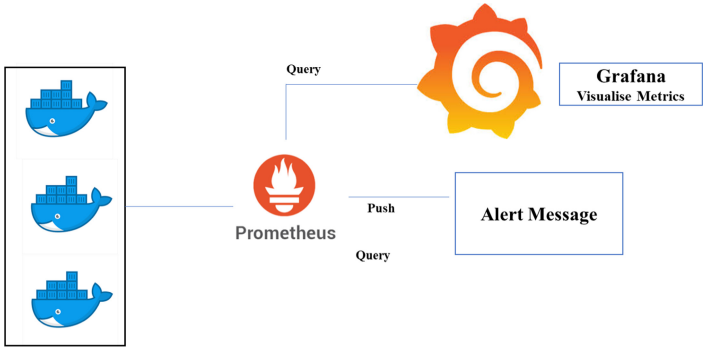

Fig. 5. Data monitoring in containers using Grafana

AppArmor and Seccomp – AppArmor[3] and Seccomp (Secure Computing with filters)[4] is a LINUX security system that proactively protects the OS and applications from the attackers. AppArmor secure the third-party applications from internal and external threats. They provide access control attributes to the program through enforcement and identify if there are any violation attempts through two profile modes called enforcement mode and violation mode. Seccomp allows secure one-way transition in a secure state through system calls like sigreturn(), read() and write() to already-open file descriptors. These system calls filters the incoming calls and reduces the risk of exposing the kernel space to the applications [33].

6 Discussion

Fog computing is gaining popularity as it reduces a significant amount of data sent to the cloud. This is achieved through the computation of the data in fog nodes through virtualized infrastructure. Containers are efficient, lightweight and stateless instances to create virtualisation using the physical or virtual topology in fog computing. The

[3] https://gitlab.com/apparmor/apparmor/-/wikis/home.
[4] https://www.kernel.org/doc/html/v4.16/userspace-api/seccomp_filter.html.

persistence storage for the data can be addressed through containers efficiently through scaling up and down the containers based on the requirements. Along with this, it also provides a good approach to maintain and support mobility to the data in geographically distributed fog nodes. However, when there is mobility to data, the security credentials need to ensure that the data is accessible by the appropriate applications and not exploited by the unknown users. Many works show that containers in fog computing layer can be prone to many security threats and the need for it to be addressed immediately. Our work proposes some of the solutions that are tabulated in Table 3.

Table 3. Proposed solutions

Solutions	Description	Advantages
Unikernels	The executable image executed on the hypervisor. The image includes application code and OS functions	Efficient execution time in Java, Python, Go Incur memory overhead of a kernel
Authorising and monitoring containers	Visualisation and analytic software's to query, visualise and alter	Users can query, visualise, and receive alter message when certain events trigger the containers
AppArmor,	Linux security system	Create two modes: Enforcement mode to define policies to the applications and Complain mode to report any violations
Seccomp	Linux security system	Secure one-way transitions through secure system calls

7 Conclusion

Container-based virtualisation provides better performance and a high virtual environment for fog-based applications. Containers provide dedicated storage and communicate resources at the user's proximity. They have many advantages by deploying macro operation and improving the efficiency of the applications. However, the container is reasonably secured and requires high security when running non- privileged processes. They are prone to many kernel leakage vulnerabilities. By addressing these security issues, we can implement a secure container-based fog application with a balance in security issues, performance, and user experience.

References

1. Bellavista, P., Berrocal, J., Corradi, A., Das, S.K., Foschini, L., Zanni, A.: A survey on fog computing for the Internet of Things. Pervasive Mob. Comput. **52**, 71–99 (2019). https://doi.org/10.1016/j.pmcj.2018.12.007
2. Bittencourt, L.F., Lopes, M.M., Petri, I., Rana, O.F.: Towards virtual machine migration in fog computing. In: Proceedings - 2015 10th International Conference on P2P, Parallel, Grid, Cloud Internet Computing 3PGCIC 2015, pp. 1–8 (2015). https://doi.org/10.1109/3PGCIC.2015.85
3. Bonomi, F., Milito, R., Natarajan, P., Zhu, J.: Fog computing: a platform for Internet of Things and analytics. In: Bessis, N., Dobre, C. (eds.) Big Data and Internet of Things: A Roadmap for Smart Environments. SCI, vol. 546, pp. 169–186. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05029-4_7
4. Global Fog Computing Market Size & Trends 2018: By Component Type, Applications, Industry Share, Segments, Analysis and Forecast 2018–2025
5. H2020: mF2C: Towards an Open, Secure, Decentralized and Coordinated Fog-to-Cloud Management Ecosystem 2017, p. 17 (2017)
6. Rivera, F.F., Pena, T.F., Cabaleiro, J.C. (eds.): Euro-Par 2017. LNCS, vol. 10417. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64203-1
7. Saurez, E., Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B.: Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In: Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems - DEBS 2016, pp. 258–269 (2016). https://doi.org/10.1145/2933267.2933317
8. He, S., Cheng, B., Wang, H., Xiao, X., Cao, Y., Chen, J.: Data security storage model for fog computing in large-scale IoT application. In: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), April 2018, pp. 39–44 (2018). https://doi.org/10.1109/INFCOMW.2018.8406927
9. Tomar, A., Jeena, D., Mishra, P., Bisht, R.: Docker security: a threat model, attack taxonomy and real-time attack scenario of DoS. In: Proceedings of the Confluence 2020 - 10th International Conference on Cloud Computing, Data Science and Engineering, pp. 150–155 (2020). https://doi.org/10.1109/Confluence47617.2020.9058115
10. Li, J., Jin, J., Yuan, D., Zhang, H.: Virtual fog: a virtualization enabled fog computing framework for Internet of Things. IEEE Internet Things J. **5**(1), 121–131 (2018). https://doi.org/10.1109/JIOT.2017.2774286
11. Jaber, A.N., Zolkipli, M.F., Shakir, H.A., Jassim, M.R.: Host based intrusion detection and prevention model against DDoS attack in cloud computing. In: Xhafa, F., Caballé, S., Barolli, L. (eds.) 3PGCIC 2017. LNDECT, vol. 13, pp. 241–252. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-69835-9_23
12. Sharma, A., Kumar, R., Mansotra, V.: An effective review on fog computing using virtualization. Int. J. Innov. Res. Comput. Commun. Eng. (An ISO Certif. Organ.) **3297**(6), 11449–11455 (2016). https://doi.org/10.15680/IJIRCCE.2016.0404207
13. Siddiqui, T., Siddiqui, S.A., Khan, N.A.: Comprehensive analysis of container technology. In: 2019 4th International Conference on Information Systems and Computer Networks, ISCON 2019, pp. 218–223 (2019). https://doi.org/10.1109/ISCON47742.2019.9036238
14. Sri Raghavendra, M., Chawla, P.: A review on container-based lightweight virtualization for fog computing. In: 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) ICRITO 2018, pp. 378–384 (2018). https://doi.org/10.1109/ICRITO.2018.8748346
15. Syed, M.H., Fernandez, E.B., Ilyas, M.: A pattern for fog computing. ACM International Conference Proceeding Series, p. 10 (2016). https://doi.org/10.1145/3022636.3022649

16. Bellavista, P., Zanni, A.: Feasibility of fog computing deployment based on docker containerization over RaspberryPi. ACM International Conference Proceeding Series (2017). https://doi.org/10.1145/3007748.3007777
17. Takahashi, K., Aida, K., Tanjo, T., Sun, J.: A portable load balancer for Kubernetes cluster. ACM International Conference Proceeding Series, pp. 222–231 (2018). https://doi.org/10.1145/3149457.3149473
18. Pérez de Prado, R., García-Galán, S., Muñoz-Expósito, J.E., Marchewka, A., Ruiz-Reyes, N.: Smart containers schedulers for microservices provision in cloud-fog-IoT networks. Challenges and opportunities. Sensors **20**(6), 1714 (2020). https://doi.org/10.3390/s20061714
19. De Lucia, M.J.: A Survey on Security Isolation of Virtualization , Containers , and Unikernels. US Army Research Laboratory, vol. 8029 (2017). https://www.dtic.mil/docs/citations/AD1035194
20. Jian, Z., Chen, L.: A defense method against docker escape attack. ACM International Conference Proceeding Series, pp. 142–146 (2017). https://doi.org/10.1145/3058060.3058085
21. Pahl, C., Lee, B.: Containers and clusters for edge cloud architectures-a technology review. In: Proceedings - 2015 International Conference on Future Internet Things and Cloud, FiCloud 2015, 2015 International Conference on Open Big Data, OBD 2015, pp. 379–386 (2015). https://doi.org/10.1109/FiCloud.2015.35
22. Martin, A., Raponi, S., Combe, T., Di Pietro, R.: Docker ecosystem – vulnerability analysis. Comput. Commun. **122**, 30–43 (2018). https://doi.org/10.1016/j.comcom.2018.03.011
23. Kabbe, J.-A.: Security analysis of Docker containers in a production environment, p. 91, June 2017
24. Sultan, S., Ahmad, I., Dimitriou, T.: Container security: issues, challenges, and the road ahead. IEEE Access **7**, 52976–52996 (2019). https://doi.org/10.1109/ACCESS.2019.2911732
25. Luo, J., et al.: Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT. Futur. Gener. Comput. Syst. **97**, 50–60 (2019). https://doi.org/10.1016/j.future.2018.12.063
26. Brogi, A., Mencagli, G., Neri, D., Soldani, J., Torquati, M.: Container-based support for autonomic data stream processing through the fog. In: Heras, D.B., Bougé, L. (eds.) Euro-Par 2017. LNCS, vol. 10659, pp. 17–28. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75178-8_2
27. Cheng, B., Solmaz, G., Cirillo, F., Kovacs, E., Terasawa, K., Kitazawa, A.: FogFlow: easy programming of IoT services over cloud and edges for smart cities. IEEE Internet Things J. **5**(2), 696–707 (2018). https://doi.org/10.1109/JIOT.2017.2747214
28. Puliafito, C., et al.: MobFogSim: simulation of mobility and migration for fog computing. Simul. Model. Pract. Theory **101**, 102062 (2020). https://doi.org/10.1016/j.simpat.2019.102062
29. Combe, T., Martin, A., Di Pietro, R.: To docker or not to docker: a security perspective. IEEE Cloud Comput. **3**(5), 54–62 (2016). https://doi.org/10.1109/MCC.2016.100
30. Bui, T.: Analysis of Docker Security, January 2015. https://arxiv.org/abs/1501.02967
31. Mcgrew, W.: An Attacker Looks at Docker: Approaching Multi-Container Applications. Blackhat18 (2018). https://i.blackhat.com/us-18/Thu-August-9/us-18-McGrew-An-Attacker-Looks-At-Docker-Approaching-Multi-Container-Applications-wp.pdf
32. Yasrab, R.: Mitigating Docker Security Issues, April 2018. https://arxiv.org/abs/1804.05039
33. Watada, J., Roy, A., Kadikar, R., Pham, H., Xu, B.: Emerging trends, techniques and open issues of containerization: a review. IEEE Access **7**, 152443–152472 (2019). https://doi.org/10.1109/ACCESS.2019.2945930
34. Goethals, T., Sebrechts, M., Atrey, A., Volckaert, B., De Turck, F.: Unikernels vs containers: an in-depth benchmarking study in the context of microservice applications. In: 2018 IEEE 8th

International Symposium on Cloud and Service Computing (SC2), November 2018, pp. 1–8 (2018). https://doi.org/10.1109/SC2.2018.00008
35. Yu, D., Jin, Y., Zhang, Y., Zheng, X.: A survey on security issues in services communication of microservices-enabled fog applications. Concurr. Comput. Pract. Exp. **31**(22), e4436 (Nov. 2019). https://doi.org/10.1002/cpe.4436
36. Cicioglu, M., Calhan, A.: Internet of Things based firefighters for disaster case management. IEEE Sens. J. 1 (2020). https://doi.org/10.1109/JSEN.2020.3013333