Computers, Materials & Continua
DOI:10.32604/cmc.2021.019432
Article

Tech Science Press

Container Introspection: Using External Management Containers to Monitor Containers in Cloud Computing

Dongyang Zhan[1,*], Kai Tan[1], Lin Ye[1,2], Haining Yu[1,3] and Hao Liu[4]

[1]School of Cyberspace Science, Harbin Institute of Technology, Harbin, 150001, China
[2]Temple University, Philadelphia, 19122, USA
[3]City University of Hong Kong, Kowloon Tong, 518057, Hong Kong
[4]Qianxin Technology Group Co., Ltd., Beijing, 100000, China
[*]Corresponding Author: Dongyang Zhan. Email: zhandy@hit.edu.cn
Received: 13 April 2021; Accepted: 06 June 2021

Abstract: Cloud computing plays an important role in today's Internet environment, which meets the requirements of scalability, security and reliability by using virtualization technologies. Container technology is one of the two mainstream virtualization solutions. Its lightweight, high deployment efficiency make container technology widely used in large-scale cloud computing. While container technology has created huge benefits for cloud service providers and tenants, it cannot meet the requirements of security monitoring and management from a tenant perspective. Currently, tenants can only run their security monitors in the target container, but it is not secure because the attacker is able to detect and compromise the security monitor. In this paper, a secure external monitoring approach is proposed to monitor target containers in another management container. The management container is transparent for target containers, but it can obtain the executing information of target containers, providing a secure monitoring environment. Security monitors running inside management containers are secure for the cloud host, since the management containers are not privileged. We implement the transparent external management containers by performing the one-way isolation of processes and files. For process one-way isolation, we leverage Linux namespace technology to let management container become the parent of target containers. By mounting the file system of target container to that of the management container, file system one-way isolation is achieved. Compared with the existing host-based monitoring approach, our approach is more secure and suitable in the cloud environment.

Keywords: Container introspection; management container; external approach; one-way isolation

1 Introduction

Cloud computing is one of the most important computing infrastructures, which is necessary for the development of the Internet of things and big data. There are many popular cloud service

providers (e.g., Amazon, Microsoft, Google) hosting the applications of lots of developers or companies. Cloud computing is based on virtualization technology, which can divide huge physical resources into many small virtual resources. There are two mainstream virtualization technologies, including virtual machine and container. Each virtual machine has its own operating system kernel, making it heavyweight and less efficient in deployment. Compared with virtual machines, containers are lightweight and more efficient in deployment, because they share the operating system kernel with the host. Therefore, it is more popular.

The widespread development of cloud computing and containers has also brought security risks. In a cloud computing environment with multi-tenant tenants, tenants need to monitor and manage their container instances. However, the current host-based or container-based monitoring methods cannot meet the requirements of security and flexibility in cloud computing. Firstly, container-based monitors running inside target containers can be detected or subverted by in-container attackers, since they are running in the same namespace. Although monitors can be executed in the host to hide themselves, it is not secure in the cloud since host applications are privileged for all containers. Secondly, redirecting execution information of target containers to security monitors of cloud tenants is not flexible, because it needs the close cooperation of the cloud host. It is not practicable for cloud service providers to corporate with each cloud tenant.

In this paper, an external container monitoring architecture is proposed to solve these problems. In this architecture, monitoring tools of cloud tenants or CSPs are deployed in external management containers, which can intercept the execution information of target containers but are transparent to them. The external container provides a secure and flexible monitoring environment for security tools. Security tools running inside the management containers are transparent and isolated to target containers. Since the management container is unprivileged, in-container security tools cannot affect host security. So, cloud tenants can run security tools as they want. After providing a one-way transparent monitoring environment, CSP needs not to closely cooperate with different security tools of cloud tenants. Therefore, our architecture is more secure and flexible compared with existing host-based or container-based security monitoring approaches.

To achieve one-way transparent management containers, process and file one-way isolation approaches are proposed. For process isolation, we leverage Linux namespace technology to make management container become the parent of target containers. Since parent namespace has full privilege over child namespace, security tools running in management container can intercept the execution information of target containers, and are transparent to them. We do not leverage the container-in-container solution to achieve one-way isolation, because it needs the parent container to be privileged. For file isolation, we mount the file system of the target container to that of the management container. Therefore, security tools can access files of target containers transparently. After the implementation, we test the effectiveness and performance of our prototype. The experimental results show that our system can make security tools be transparent to target containers with high performance.

In summary, the contributions of this paper are as follows.

- An external container monitoring approach is proposed to build management containers over target containers. Security tools running inside management containers are transparent to target containers, but can intercept the execution information of them.
- A namespace-based process isolation approach is proposed by making the namespace of the management container to be the parent of target containers and ensuring the access security of the management container.

- For file isolation, a file-system-based file introspection method is used for the management containers to access the files of target containers outside.

The rest of this paper is organized as follows. Section 2 gives the related work. The system design is described in Section 3. The design details about process introspection and file system introspection are given in Section 4 and Section 5 respectively. Section 6 evaluates the effectiveness and performance of the prototype. Section 7 concludes this paper and discusses the future work.

2 Related Work

At present, container introspection technology is still a new research topic, and this concept is related with cloud security, virtual machine introspection and container security technologies [1,2].

2.1 Virtual Machine Introspection

The current research direction closest to the container introspection technology is virtual machine introspection (VMI) [3], which monitors VMs from the hypervisor. The hypervisor has the highest privilege and can intercept the execution information of target VMS. Under the framework of VMI, the monitoring of the VMs generally includes three steps: execution information collection, semantic reconstruction, and behavior analysis.

There are two approaches to collect VM information, including static analysis and dynamic interception. CFMT [4] obtains the contents of the VM disk from the outside and saves the checksums of all the original file contents in the VM files. Then it compares the checksum of the existing contents with the original one during each poll to detect whether there the file has been tampered with. VMWatcher [5] maps the file systems of a VM to the privileged virtual machine (DOM0), and uses anti-virus software to perform security analysis on these files to protect the security of the VM.

Compared with static approaches, dynamic approaches can obtain the execution information in real time. Ether [6] can capture the behavior of virtual machine system calls in real time. The system modifies the content of a specific register (MSR register) of the target VM. When the VM executes a system call, it throws an exception, which can trigger an event in the hypervisor. At this time, Ether can obtain specific information about the system call, such as call number and parameters. Nitro [7] is similar to Ether, but Ether is designed for the Xen platform and Nitro works in KVM platform.

The execution information collected in the VMM layer is binary low-level information, but security analysis needs high-level information. After collection, it needs to be reconstructed into high-level semantic information. Filesafe [8] reconstructs file system from disk image based on the layout of Windows FAT32 file system and then maps files and disk blocks. vMon [9] reconstructs the map between file and disk block for Linux VM. Volatility is an open-source memory analysis tool, which can reconstruct kernel objects from binary memory snapshot based on the profiles of layouts of different operating systems. Reference [10] can identify the kernel version by analyzing the VM kernel automatically.

VMI-based security tools are usually applied for cloud security [11–13], such as Cloud-VMI [14], SECLOUD [15] and ESI-Cloud [16].

2.2 Container Security

Container security [17] is a hot topic, so there are many works to analyze and protect container security. Reference [18] compares virtual machines and containers. Compared to virtual

machine technology, containers are more light-weight and can reduce the resource consumption, because containers share the operating system of the host. But it also raises security risks due to the huge attack surface of the operating system. In addition, the ecosystem of the Docker containers also contains security challenges, this paper also focuses the security of it.

SCONE [19] leverages Intel SGX technology to protect the Docker containers from external malicious attacks and the untrusted cloud hosts. To defense against container escalation attacks, Reference [20] proposes an escape defense method by checking the status of Linux Namespace, which can detect abnormal processes and prevent users from malicious escaping behaviors. Reference [21] focuses on strengthening the Docker's access restriction, and hopes to extend the dockerfile format so that the Docker image maintainer can provide the SELinux security policies to enhance the security of the container. SELinux is widely used to enhance the security of certain service programs, the expansion and adjustment of dockerfile will allow SELinux security policies to be specified for different images, improving the security of Docker. Reference [22] exploits the Linux cgroups from containers and proves that cgroups technology is not enough to limit the resource access of containers in cloud computing.

2.3 Container Introspection Technology

To analyze the security of containers, Reference [23] uses the introspection tool Prometheus to capture the information including the Docker engine itself and the memory usage of the container and the host OS. This method analyzes the data difference between the Apache server running in the container during normal operation and when it is infected by malware, and concludes that introspection tools can be used as data collection and forensic analysis tools for the early warning system in the containerized system. Reference [24] proposes a malicious node identification method.

In summary, there has not been much research on the concept of container introspection technology. This paper is a preliminary exploration of this concept.

3 Design of Container Introspection

This section describes the design overview of our system. We first give the motivation and then describe the threat model and assumption. After that, two one-way isolation approaches for process and file system are proposed.

3.1 Motivation

With the development and wide application of container technology, more and more containers are deployed in cloud computing. Containers are facing increasing security risks since they are providing services on the Internet. Therefore, many cloud tenants and cloud service providers need to manage and monitor their containers. There are several security and flexibility requirements of container monitoring, which are as follows.

RQ1: Monitoring tools should be secure. Many containers execute processes with root privilege by default. It is not secure for monitors running inside them. Even though container processes are not privileged, they can also detect monitoring tools, which is not secure for monitoring tools.
RQ2: Security tools should be isolated from the host. Processes running inside containers can be accessed in the host, but it is not secure to run security tools of cloud tenants in the host in cloud computing.

RQ3: Cloud service providers should not closely cooperate with security tools. Another monitoring solution in the context of virtual machines is that the host intercepts execution information of the target VM and then sends it to the security tools running inside another secure VM. In the context of containers, it is possible for CSP to transmit the execution information of target containers to a secure container. However, this solution needs the close cooperation of CSPs. CSPs need to provide the customized monitoring APIs or required information for different monitors. It is very complicated for CSPs to perform API authorization and access control, since there are many containers of different cloud tenants running on the host.

To meet these requirements, this paper proposes an external approach for container introspection, which builds a one-way isolated management container over target containers. The management container is a secure and flexible environment for monitoring tools.

3.2 Threat Model & Assumptions

Before describing the design of our system, we first discuss the threat model and some assumptions.

We first assume that containers cannot escape to the cloud host. Container privilege escalation is a serious attack for cloud computing and security researchers are keeping fix vulnerabilities in the host kernel. But currently there no perfect host-based solution to defense container privilege escalation. Therefore, we do not consider container escape attack in this paper.

In addition, the CSPs are considered to be trusted, which is a common trust base in cloud computing. Most security monitors in cloud computing rely on the isolation provided by cloud hosts or hypervisors. The design of our system is also based on the security protection of cloud hosts.

Cloud tenants are responsible for the security of their security monitors. In this paper, we do not analyze the security of monitoring tools from cloud tenants and do not provide mechanisms for cloud tenants to intercept their security monitors.

3.3 System Overview

The core method of our system is to build a one-way isolation management container, which can intercept the execution information of target containers but is transparent to them. The architecture of our system is shown in Fig. 1.

The host provides OS virtualization service for containers, which has the highest privilege and isolates different containers. There are two modules in the system, building the one-way isolation environment for management containers. The PID NS management module makes the PID namespace of the management container be the parent of target containers by leveraging the host namespace mechanism. After that, the management container can intercept the process information of the target containers without the cooperation of the host. File system management module analyzes the structure of target containers and mounts them to the management container. Security tools of cloud tenants run in the management container, so that they can access the execution information of target containers. There can be many target containers managed by only one management container.

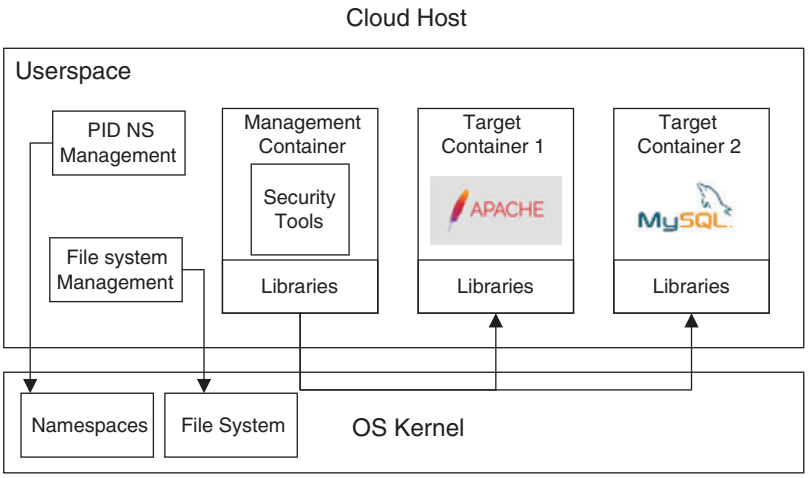

Figure 1: The system architecture of container introspection

4 Namespace-based Process Introspection

We leverage the Linux namespace mechanism to achieve one-way process isolation. The main steps include: 1) constructing a parent PID namespace of target containers; 2) letting the management container join the parent namespace dynamically; 3) hiding other processes of the parent PID namespace. At first, we make a brief introduction of Linux namespaces.

4.1 Linux PID Namespace

There are 6 different namespaces introduced by the Linux kernel after v2.6. These namespaces are used to isolate different kinds of resources for containers, including PID namespace, UTS namespace, IPC namespace, MNT namespace, NET namespace and USER namespace. Among them, the PID namespace is used for process domain isolation. The relationship between different PID namespaces is shown in Fig. 2. Only processes within the same PID namespace could see each other. The parent PID namespace has full privilege over child namespaces. Processes of child namespace are mapped into parent namespace. For instance, Process 1 of Namespace 1 is mapped as Process 5 in Namespace 0, so all of the processes in Namespace 0 are able to see it. For the Docker container, a new namespace is automatically created when the container is created. All processes running inside one container belong to the same PID namespace, so they can see each other. Since different containers have different PID namespaces, processes of different containers can see each other.

Inspired by the PID namespace, we found that if the PID namespace of the management container is that of the target container, the management container can visit the processes of the target container transparently.

4.2 Docker-in-docker Architecture

There are several docker-in-docker solutions (e.g., dind), which make the docker create a new docker inside it. A child namespace can also be the parent of other namespaces. Based on this mechanism, running a docker in another docker is possible. But most dockers cannot create child dockers, because there are several challenges First, the parent docker should be privileged. However, privileged dockers are not safe and not accepted in cloud computing. Second, the file system of Docker (AUFS) should only consist of normal file systems, which means docker cannot

run based on multiple AUFS file systems. These challenges are shortcomings of dind. So, dind should be created by Docker with 'privileged' flag, and the file system of child docker should be a volume of the parent docker.

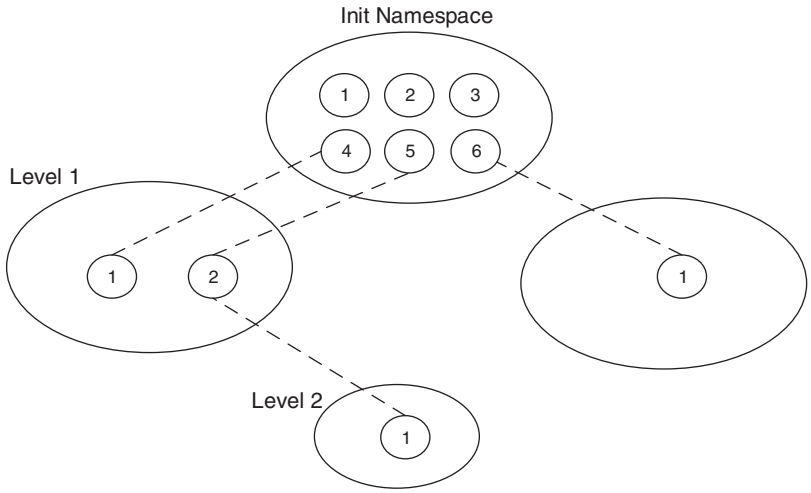

Figure 2: The relationship between different namespaces

Even though docker-in-docker solutions can build multiple levels of namespaces and make external monitoring possible, the security risks of privileged containers are unacceptable in cloud computing. Therefore, our approach leverages the architecture of the docker-in-docker solution and tries to overcome the problems of it.

4.3 Joining into Parent Namespace

To address the problems of the docker-in-docker solution, our system is based on the docker-in-docker solution but does not execute security monitors directly in the parent docker. The main approach is to let the target containers run inside a parent container, and then let management container join the PID namespace of the parent docker. The steps are as follows.

1) Creating a parent container. When a cloud tenant creates her first container, the host first creates a parent container. This container can create child containers, so it is created with the 'privileged' flag. But this step is transparent to the cloud tenants, and the tenants cannot operate the parent container.
2) Creating target containers. Target containers are created within the parent container, so the PID namespace of the parent container is the parent of those of target containers. Parent container has the client which can create child containers for cloud tenant. Only containers of the same cloud tenant can be created in one parent container.
3) Creating the management container. Management container is the execution environment of security monitors. To access the information of target containers, we let it join the PID namespace of the parent container, when it is created. Since this container only shares the same PID namespace of the parent container, it is not as privileged as the parent container. Therefore, it is more secure.

4.4 Process Hiding

Since the management container shares the same PID namespace of the parent container, the management container can obtain the process information of client tools of the parent container, making the parent container be visible. To overcome this problem, we leverage a kernel module to hide the client in the parent container.

The process hiding approach is inspired by kernel rootkits. Kernel rootkits are used to hide processes or files in operating systems. They usually hook kernel functions and inject malicious code in kernel system calls. Among them, adore-ng is a popular rootkit, which is used for process and file hiding. It injects malicious code into several system calls (e.g., gedents64, etc.). The injected code deletes the process that needed to be hidden from the result list.

Our process hiding module is also based on the idea of kernel rootkit, hooking some key kernel functions and injecting code into several system calls (e.g., getdents). The difference is the module only hides the process information of the client from the management container. Since the management container and parent container share the same PID namespace, we hide the client process in the parent container from other processes within this namespace. Therefore, the most important step is to identify the processes of different namespaces.

To address this challenge, we first explore the relationship between processes and namespaces. In Linux, all task structures are linked by a doubly linked list, which first task is labeled with the 'init_task' symbol. As shown in Fig. 3, every task structure has a nsproxy object to record its different namespaces. The pid_ns fields of different tasks in a same container point to only one PID namespace object, so we can identify all the processes of a container in the kernel by identifying the pid_ns pointer of its task structure.

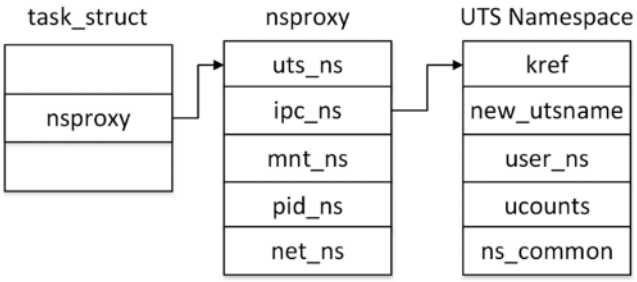

Figure 3: Relationship between process structure and namespace

When a parent container is created, the corresponding PID namespace is identified. Then, the pid of client in the parent container is obtained. After that, the namespace and pid are transferred to the kernel module via a customized system call. The kernel module reads the information and performs process hiding. If a process of parent container wants to obtain the process list by using ps command, the result of the system call is checked and the information of the client is cleared.

5 File System Introspection

The file system introspection is based on the AUFS file system, which is widely used by the Docker containers. So, we first make a brief introduction of AUFS, and then describe how we leverage AUFS to perform external file system introspection.

AUFS (short for advanced multi-layered unification file system) is an implementation of the Union File System, which can merge file directories stored in different locations and mount them to the same directory. As shown in Fig. 4, the file system is copy-on-write. There are several layers in the file system of a docker. The lower layers are image layers, which are read-only. Therefore, these layers can be used for many dockers at the same time. The upper layer is writable, which records the modifications to the underlying images. When files are added, deleted, and modified in the container, a runtime copy will be generated in the upper layer. All the layers are mounted to the same directory with a number, which is not the ID of the container. So, we need to find the correspondence between this number and the container ID.

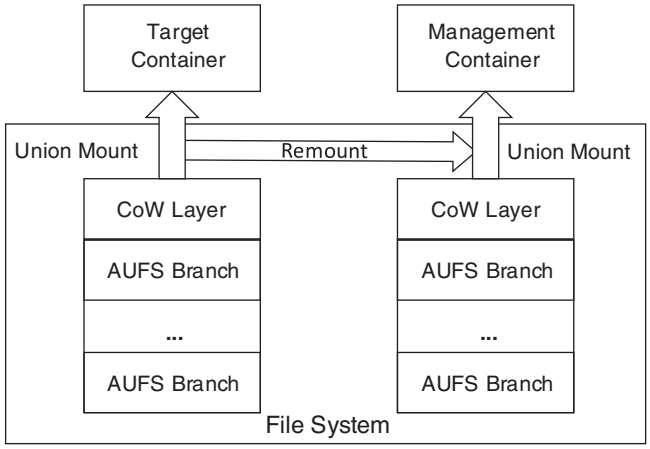

Figure 4: File system introspection

Based on the features of AUFS, we achieve the one-way file system isolation by mounting the mount point of the target container in the host to a subdirectory of the management container and analyzing the mapping between container ID and mount ID automatically, which is shown in Fig. 4. In addition, when the target containers are going to be removed, the file system of it should be unmounted from the management container first, to prevent the failure of container removal.

6 Evaluation

This section evaluates the effectiveness and performance of our system. The testbed is a PC equipped with 3.2 GHz Intel i5 CPU and 8 GB RAM. The host operating system is Ubuntu 16.04. Dind is selected as the image of the parent container. The image of the target container is Ubuntu 14.04.

6.1 Effectiveness

Our system can obtain the process information and files of the target container from the management container, so the first step is to deploy the monitoring tools in the management container after deploying the target container. We test the effectiveness of process introspection and file system introspection respectively.

To test the effectiveness of process introspection, we first introduce a workload in the target container and then run the 'ps' command in management to obtain the process list of the target

container. The target container is running the ps command, and the management container is able to get the corresponding process information. Then, we run the ping command in the management container, and run the ps command in the target container. The results show that the processes of the management container are transparent to the target container.

We obtain the file list of the target container in the management container to test the effectiveness of file introspection. As shown in Fig. 5, we first add a new file in the target container, then get the file list in the management container in Fig. 6. From the result, we can know that the management container is able to access the file system of target containers.

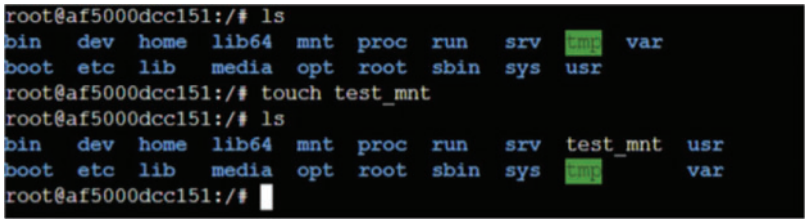

Figure 5: Adding a file in the target container

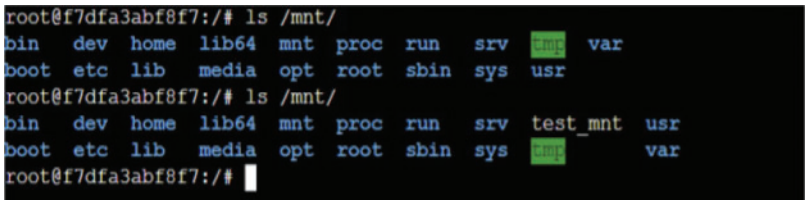

Figure 6: Accessing files from the parent container

6.2 Performance

Since our system introduces a parent container to user containers, the parent container will consume file space. Therefore, we measure the size of the dind image, which is 533 MB. According to the results, the space cost is acceptable for cloud computing.

There is a kernel module to hide the client processes in parent containers, which could introduce overhead to the 'ps' command. So, we test the execution time before and after the injection of the kernel module for 1000 times, and then compare the performance. The average execution time of 'ps' before module injection is about 23.05 ms, and it is 23.2 ms after injection. The experimental results show that the overhead is acceptable since the 'ps' operation is not time-sensitive.

7 Conclusion & Future Work

This paper proposes an external container introspection approach to monitor target containers from a management container, which is built by achieving a one-way transparent process and file isolation. Based on Linux namespaces, the namespace of the management container is the parent of target containers, so security tools running inside the management container can obtain the execution information of the target containers. For file system introspection, we analyze and mount the file system of target container to the management container. After the implementation,

we test the prototype. The experimental results show that our system is effective with acceptable overhead. In this paper, we do not analyze cross-host containers of one cloud tenant. Containers belonging to one cloud tenant may be deployed in different physical cloud hosts, but our system can only analyze containers in one host. To analyze cross-host containers of one cloud tenant, we need to correlate and analyze different cloud hosts. This work is left to future work.

Funding Statement: This paper is supported by National Natural Science Foundation of China (http://www.nsfc.gov.cn/) under Grant No. 61872111, and Sichuan Science and Technology Program (http://kjt.sc.gov.cn/) under Grant No. 2019YFSY0049 which are both received by L. Ye.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] L. Jiang and Z. Fu, "Privacy-preserving genetic algorithm outsourcing in cloud computing," *Journal of Cyber Security*, vol. 2, no. 1, pp. 49–61, 2020.
[2] W. M. Eid, S. Atawneh and M. Al-Akhras, "Framework for cybersecurity centers to mass scan networks," *Intelligent Automation & Soft Computing*, vol. 26, no. 6, pp. 1319–1334, 2020.
[3] T. Garfinkel and R. Mendel, "A virtual machine introspection based architecture for intrusion detection," in *Proc. NDSS*, San Diego, California, USA, pp. 191–206, 2003.
[4] S. Anjali, S. Gupta and K. Padam, "A light weight centralized file monitoring approach for securing files in cloud environment," in *Proc. ICITST*, London, UK, pp. 382–387, 2012.
[5] X. Jiang, X. Wang and D. Xu, "Stealthy malware detection through vmm-based" out-of-the-box" semantic view reconstruction," in *Proc. CCS*, Alexandria, Virginia, USA, pp. 128–138, 2007.
[6] A. Dinaburg, P. Royal, M. Sharif and W. Lee, "Ether: Malware analysis via hardware virtualization extensions," in *Proc. CCS*, Alexandria, Virginia, USA, pp. 51–62, 2008.
[7] J. Pfoh, C. Schneider and C. Eckert, "Nitro: Hardware-based system call tracing for virtual machines," in *Proc. IWSEC*, Tokyo, Japan, pp. 96–112, 2011.
[8] J. Wang, M. Yu, B. Li, Z. Qi and H. Guan, "Hypervisor-based protection of sensitive files in a compromised system," in *Proc. ACSAC*, Orlando, Florida, USA, pp. 1765–1770, 2012.
[9] N. Li, B. Li, J. Li, T. Wo and J. Huai, "vMON: An efficient out-of-VM process monitor for virtual machines," in *Proc. HPCC-EUC*, Zhangjiajie, Hunan, China, pp. 1366–1373, 2013.
[10] Y. Hebbal, S. Laniepce and M. J. Menaud, "K-binID: Kernel binary code identification for virtual machine introspection," in *Proc. DSC*, Taipei, Taiwan, pp. 107–114, 2017.
[11] C. Lv, J. Zhang, Z. Sun and G. Qian, "Information flow security models for cloud computing," *Computers, Materials & Continua*, vol. 65, no. 3, pp. 2687–2705, 2020.
[12] J. Pan, Y. Zhuang, X. Hu and W. Zhao, "Fine-grained binary analysis method for privacy leakage detection on the cloud platform," *Computers, Materials & Continua*, vol. 64, no. 1, pp. 607–622, 2020.
[13] J. Qin, Y. Cao, X. Xiang, Y. Tan, L. Xiang *et al.*, "An encrypted image retrieval method based on simhash in cloud computing," *Computers, Materials & Continua*, vol. 63, no. 1, pp. 389–399, 2020.
[14] H. wook Baek, S. Abhinav and J. Van der Merwe, "Cloudvmi: Virtual machine introspection as a cloud service," in *Proc. IC2E*, Boston, MA, USA, pp. 153–158, 2014.
[15] H. Zhou, H. Ba, Y. Wang, Z. Wang, J. Ma *et al.*, "Tenant-oriented monitoring for customized security services in the cloud," *Symmetry*, vol. 11, no. 2, pp. 252, 2019.
[16] J. Ren, L. Liu, D. Zhang, H. Zhou and Q. Zhang, "ESI-Cloud: extending virtual machine introspection for integrating multiple security services," in *Proc. SCC*, San Francisco, CA, USA, pp. 804–807, 2016.
[17] T. Bui, "Analysis of docker security," arXiv preprint, arXiv:1501.02967, 2015.
[18] T. Combe, A. Martin and R. Di Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 54–62, 2016.

[19] S. Arnautov, B. Trach, F. Gregor F, T. Knauth, A. Martin *et al.*, "SCONE: Secure linux containers with intel SGX," in *Porc. OSDI*, Savannah, GA, USA, pp. 689–703, 2016.

[20] Z. Jian and L. Chen, "A defense method against docker escape attack," in *Proc. ICCSP*, Wuhan, China, pp. 142–146, 2017.

[21] E. Bacis, S. Mutti, S. Capelli and S. Paraboschi, "Dockerpolicymodules: Mandatory access control for docker containers," in *Proc. CNS*, Florence, Italy, pp. 749–750, 2015.

[22] X. Gao, Z. Gu, Z. Li, H. Jamjoom and C. Wang, "Houdini's escape: Breaking the resource rein of linux control groups," in *Proc. CCS*, London, UK, pp. 1073–1086, 2019.

[23] T. Watts, R. Benton, W. Glisson and J. Shropshire, "Insight from a docker container introspection," in *Proc. HICSS*, Grand Wailea, Hawaii, USA, 2019.

[24] S. Su, Z. Tian, S. Liang, S. Li, S. Du *et al.*, "A reputation management scheme for efficient malicious vehicle identification over 5G networks," *IEEE Wireless Communications*, vol. 27, no. 3, pp. 46–52, 2020.