Design and implementation of endogenous security container based on union file system

Fukang Xing1*, Zheng Zhang1, Bolin Ma1, and Bingzheng Li1

1 Computer technology, Information Engineering University, Zhengzhou, Henan, 450001, China
*Corresponding author’s e-mail: k36666@163.com

Abstract. In order to solve the increasing attacks on container file system and the IO errors of containers in big data processing scenarios in cloud computing environment, a scheme based on the idea of heterogeneous redundancy in endogenous security and transformation of container union file system was proposed to improve the security and fault tolerance of containers. Based on the above scheme, experiments are carried out on Docker, the most popular container technology, and OverlayFS, the most representative union file system. The experimental results show that this scheme can improve the security and fault tolerance of containers on the premise of ensuring availability, and realize the endogenous security of containers.

1. Introduction
Cloud computing is a network-based computing mode. The cloud computing system provides users with CPU, memory and other resources in the form of virtual machines through virtualization technology [1]. There are usually three types of cloud computing services, namely infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS) and software-as-a-service (SaaS) [2]. Cloud computing needs to be flexible, scalable and cost-effective [3], while container technology meets the demand for efficient resource scheduling in cloud computing [4,5]. With the extensive application of container technology, how to protect container security is regarded as the key of container use. Container technology provides separate rootfs for container processes through the mount Namespace technology, isolating file operations between containers. In order to avoid consuming a large amount of storage space when creating containers, mainstream container technologies such as Docker adopt union file systems such as OverlayFS and AUFS [6] to add a read/write layer on the basis of read-only images to record the modifications based on read-only images during the operation of containers, and reduce the use of storage by reusing read-only images. In the cloud computing environment, attackers often use file system-level methods such as uploading malicious files, writing malicious content, and reading sensitive file contents to attack. To prevent attacks, it is very important to protect container file systems. At the same time, due to the flexibility and extensibility of cloud computing, big data processing through cloud computing has become increasingly popular in recent years. In the process of data processing, errors often occur in the data system, resulting in incorrect IO results. In aviation, finance, medical and other fields, such errors will bring in calculable consequences [7]. Therefore, based on the idea of heterogeneous redundancy in the endogenous security theory [8], this paper improves the security and fault tolerance of containers at the file system level through multi-variant execution technology and transformation of container union files.
2. Related work

2.1. Multiple variant execution techniques
Multi-variant execution technology (MVX) was first proposed by Cox\cite{9}, which ensures the security of software operation according to the randomization of memory distribution of two processes with the same function and the monitor. On this basis, Salamat \cite{10} proposed the monitor run independently many variations of architecture, and put forward such as Koning MvArmor variant system architecture\cite{11}, the architecture based on system call for a vote, and adopting the architecture of the user can choice on the security and performance of two dimensions, in order to achieve the protection software security under the premise of guarantee its performance. Figure 1 is a typical multi-variant architecture diagram.

![Figure 1. Typical multi-variant architecture](image1)

2.2. Union file system
The union file system was originally used to realize that files saved in different locations are viewed together\cite{12}. The union file system presents its own virtual file system (VFS) as the kernel to the file systems stacked beneath it, and because the union file system provides a view of the file system to the kernel, it can be used by any application. Because it intercepts operations bound to low-level file systems, it can modify operations to present a unified logical view. Figure 2 shows the structure of the union file system.

![Figure 2. Structure of the union file system](image2)

3. Design
The overall design of this scheme can be divided into the construction of heterogeneous equivalent multi-variant container, IO operation interception proxy and IO operation comparison voting.
3.1. Heterogeneous equivalent multi-variant container construction
In order to produce a group of functionally equivalent but heterogeneous containers, a dual-mode redundant architecture is used to create containers, that is, two containers are generated from the same image each time. By operating the system to provide the address space layout randomization (ASLR), can realize the dual mode redundancy architecture to create two containers executes, its process in the stack segment and place on the base address of the Shared library inconsistency, at the same time because the ASLR function does not affect the normal process execution process, so the generated two containers on functional equivalence, but there is heterogeneity in the memory space layout. To facilitate the judgment of equivalent containers, the ID of equivalent containers needs to be recorded when creating dual-mode redundant containers.

3.2. IO operation interception proxy
Mainstream container file systems use union file systems, which, when mounted, act as a transparent middle layer between the VFS and the underlying file system. Compared with I/O interception at the VFS layer and the underlying file system, I/O interception in the union file system avoids the huge impact of VFS modification on the kernel and tedious modification of all the underlying file systems. In this scheme, the key IO functions of the union file system are intercepted by proxy, and the container ID, IO content and directory of each IO operation are recorded.

3.3. I/O operation comparison vote
From the perspective of security, the scheme in this paper adopts dual-mode redundancy architecture to generate equivalent heterogeneous containers. By enabling the ASLR mechanism provided by the operating system, the two containers have different memory space layouts. When attackers attack, memory information by means of memory information leakage, as the two container memory space heterogeneity exists, it is difficult to attacks on two containers at the same time, and if the attacker only successful attack one of the equivalent heterogeneous container, and seized the control flow of the vicious IO operations, will inevitably lead to be attacked containers and not be attacked produce inconsistent IO operations. From the perspective of fault tolerance, this paper adopts dual-mode redundancy architecture to generate equivalent containers based on the same image. When an error occurs in one equivalent container and IO operation is generated, inconsistent operation with the IO operation of another functional equivalent container will occur, so as to timely discover and correct the errors generated during program running. In this scheme, I/O queue mechanism is used to record I/O operations of two equivalent containers, and the consistency of I/O operations in the two queues is compared asynchronously by a voter. By comparing the directories and contents of I/O operations performed by the two containers, determine whether the I/O operations performed by the two containers are equivalent. If the I/O operations performed by the two containers are not equivalent, determine whether the I/O operations performed by the two containers are attack behaviors or errors occur, and terminate the program using the feedback mechanism. At the same time, asynchronous mechanism is adopted in the comparative voting to avoid the waiting operation required by container IO synchronization, which further reduces the cost of voting and improves the efficiency of voting.

4. implementation
The scheme described in this paper is implemented based on Docker container technology and its dependent union file system OverlayFS.

4.1. Heterogeneous equivalent multi-variant container construction
The user mode script encapsulates the command of docker client to create containers. When the user needs to create containers, on the basis of the same image, the docker run command is executed twice to create equivalent heterogeneous containers generated by dual-mode redundant architecture. To determine whether the I/O operation is from a group of heterogeneous equal-cost containers during container file I/O voting, record the PID of a group of heterogeneous equal-cost containers after they
are created. In this scheme, a new system call method is adopted to transfer a group of heterogeneous equal-cost container Pids from user state to kernel state, and an EQ_container_PID structure linked list is maintained in the kernel to store the PID information of heterogeneous equal-cost container.

```c
long sys_getcontainerpid(unsigned int container1_pid, unsigned int container2_pid)
{
    The equivalent heterogeneous container process PID is stored in eq_container_PID structure linked list eq_container_PID.
}
```

```c
struct eq_container_pid
{
    unsigned int pid1;
    unsigned int pid2;
    struct eq_container_pid *next ;
};
```

4.2. IO operation interception proxy

Linux system through the VFS file management, VFS as a virtual file system, capable of supporting a variety of underlying file system, VFS file object through the file structure of related operations, kept in the file structure refers to the function pointer file operations function, when the file object creation, VFS will function pointer point to correspond to the underlying file system file operations, the current mainstream OverlayFS container technology mainly used as the file system, OverlayFS skillfully use of the characteristics of the VFS, when mount OverlayFS, Function Pointers in VFS point to file-manipulation functions in OverlayFS, which in turn call file-manipulation functions in specific file systems in the lower layer of OverlayFS. OverlayFS can therefore be viewed as an intermediate layer between the VFS and the underlying file system. The scheme described in this paper modifies the read/write function of OverlayFS so that it can record the contents and directory of IO operations performed by the container, as well as the PID of the container process, so as to conduct comparative voting of IO operations in heterogeneous equivalent containers. OverlayFS IO operations in key function is the file. The c file ovl_read_iter and ovl_write_iter function into the participation kiocb type structure preserved the buffer to be read or written to the file from the file content, and to be read or written to the target file, this scheme adopts the new structure IO_info get kiocb stored in the content of the need to read or write, read or written to the target file and the current IO operation process of pid, It is then passed to the subsequent IO operation comparison vote function.

```c
struct IO_info
{
    bool IOflag;
    unsigned int pid;
    char* path;
    char* context;
};
```

4.3. I/O operation comparison vote

The scheme described in this paper improves security and fault tolerance by comparing different IO operations in equivalent dual-mode heterogeneous redundant architecture containers. In this scheme, IO_compare is established in kernel for heterogeneous redundant architecture containers to record and compare IO operations. According to the information in eq_container_PID structure linked list, IO_compare structure is created. In IO_compare, two queues are included to record IO operation structure IO_info of two equivalent containers, as shown in Figure 3. When both container operation normal, should produce consistent IO operations, the two in the queue IO_info shall be consistent, when the queue is not empty, the corresponding IO_info structure for both the queue for the
comparative judgment, when two queue corresponding IO_info record structure in IO types, read or written to the file path, read or written to the file contents are consistent, the determination of the container IO operation safety and correct, when the two queues corresponding IO_info record structure in IO types, read or written to the file path, read, or write the content of the files is not consistent. In this scenario, comparison voting function determines that the I/O operations of the container have security risks or are running incorrectly, and stop the container process to prevent unsafe or incorrect I/O operations.

```c
struct IO_compare
{
    struct IO_info_queue* queue1;
    struct IO_info_queue* queue2;
}
struct IO_info_queue
{
    unsigned int pid;
    struct Link* front;
    struct Link* rear;
    int size;
};
```

![Figure 3. Structure of the IO_info queue](image)

5. Experiment and result analysis

5.1. security
The security of this scheme is verified by simulating typical malicious file read and write behaviors that may exist in real scenarios. Docker was used as a container management tool in the experiment, and containers with dual-mode redundant architecture were generated through the same centos:7 image. As ASLR mechanism was enabled, container processes had heterogeneity in memory space. The experiment assumes that the attacker can only perform a successful malicious read/write operation on one of the equivalent containers. Three typical malicious read and write operations, namely uploading malicious files, writing malicious content and reading sensitive information, are experimented
5.1.1. UPloading Malicious Files

In order to simulate malicious file upload, docker-exec enters one of the containers, creates files and writes contents to simulate malicious file upload through echo instruction, and the other container does not perform this operation. After that, the two containers carry out unified file read and write operations. The experimental results are shown in Table 1.

| Container 1 operation | Container 2 operation | Container 1 state | Container 2 state |
|-----------------------|-----------------------|-------------------|-------------------|
| Step 1 Echo "hello" >> hello. PHP | null | The simulated malicious file hello.php was uploaded | The normal operation |
| Step 2 cat /root/a.txt | cat /root/a.txt | The container stops running because the I/O operations are different | The container stops running because the I/O operations are different |

After a malicious file upload to the container 1, two container for the next IO operations, at this point in the IO_compare structure, both container process IO operations queue is not empty, comparative vote function to proceed to a vote, the container 1 IO operations queue contains two IO operations, respectively, to hello. PHP file into content "hello" and read/root/a.txt file content, but the container 2 IO operations only for read/root/a.txt file content. Due to the first-in-first-out nature of the queue, the comparison voting function determines that the first IO operation of container 1 is different from the first IO operation of container 2, thus preventing further malicious operations and protecting container security by preventing the operation of both containers.

5.1.2. Malicious content writing

In order to simulate malicious content writing, docker-exec is used to enter the interior of one of the containers, and echo instruction is used to add a line to the interior of the file, and another container adds a different line to the interior of the corresponding file. The experimental results are shown in Table 2.

| Container 1 operation | Container 2 operation | Container 1 state | Container 2 state |
|-----------------------|-----------------------|-------------------|-------------------|
| Step 1 Echo "hello" >> /root/a.txt | Echo "bye" >> /root/a.txt | The container stops running because the I/O operations are different | The container stops running because the I/O operations are different |

Container 1 writes malicious content, while container 2 writes normal content. At this point, in IO_compare structure, the I/O operation queues of the two container processes are not empty, and the comparison voting function begins to vote. At this point, container 1 and container 2 write the same object file, but the contents are different. By comparison voting function, the two containers are judged to have different I/O operations. In this way, the two containers are prevented from running to prevent further malicious operations and protect container security.

5.1.3. Sensitive information reading

In order to simulate the reading of sensitive information, docker-exec is used to enter the inside of one container, and the contents of the file /root/a.txt are read by CAT instruction, and the contents of the file /root/b.txt are read by the other container. The experimental results are shown in Table 3.
Table 3. Results of sensitive information reading.

| Container 1 operation | Container 2 operation | Container 1 state | Container 2 state |
|-----------------------|-----------------------|-------------------|-------------------|
| Step 1 cat /root/a.txt | cat /root/b.txt | The container stops running because the I/O operations are different | The container stops running because the I/O operations are different |

Container 1 reads sensitive information files, while container 2 reads normal files. At this time, in IO_compare structure, the IO operation queues of the two container processes are not empty, and the comparison voting function begins to vote. At this time, the target files read by container 1 and container 2 are different. By comparison voting function, the two containers are judged to have different I/O operations. In this way, the two containers are prevented from running to prevent further malicious operations and protect container security.

5.2. Fault tolerance

The fault tolerance of the solution is verified by simulating I/O errors that may occur during container running in real scenarios, and the security of the solution is verified by simulating typical malicious file read/write behaviors that may occur in real scenarios. Docker was used as a container management tool in the experiment, and containers with dual-mode redundant architecture were generated through the same centos:7 image. Because it is generated by the same image, there is functional equivalence between the two containers. The experiment assumes that IO errors can occur during container process execution and only exist in one of the equivalent containers.

When the same binary program containing random function is run in two containers, the program will write the results of random function into the specified file. The running results of random function are different, so as to simulate the IO error in the container, resulting in inconsistent IO operations in the two containers. The experimental results are shown in Table 4.

Table 4. Results of Fault tolerance.

| Container 1 operation | Container 2 operation | Container 1 state | Container 2 state |
|-----------------------|-----------------------|-------------------|-------------------|
| Step 1 Write random function result 1035 to /root/a.txt | Write random function result 2787 to file /root/a.txt | The container stops running because the I/O operations are different | The container stops running because the I/O operations are different |

Container 1 and container 2 write different contents to the file. The comparison voting function determines that the I/O operations of the two containers are different. In this way, the operation of the two containers is prevented to prevent further operations and further errors, so as to improve the fault tolerance of container I/O operations.

5.3. Performance

The original OverlayFS and the modified OverlayFS in this solution are tested using the file system performance test tool FIO. The read and write operations per second (IOPS) test is performed for 100 times in random and sequential scenarios. Figure 4 shows the comparison of average IOPS between the two scenarios.
Figure 4. Performance comparison between OverlayFS and modified OverlayFS

As shown in the scheme of the modified OverlayFS with original OverlayFS in order to read situations 25.0% of the loss of performance, in order to write scenarios have a 18.1% performance losses, the random read scenario 28.3% of the performance loss, in the performance of the random write scenarios have a 21.0% loss, in the premise of guarantee of availability of container security and fault tolerance.

6. Conclusions
In order to solve the shortage of security and fault tolerance of containers, this paper proposes a scheme that can improve security and fault tolerance of containers and realize the endogenous security of containers by referring to the idea of heterogeneous honor and reforming the union file system of containers. Generated through ASLR technology function equivalence but there are differences in memory layout more containers, the union file system, so it can record the container of IO operations, finally by comparing the IO operation to realize the discovery of malicious file IO and error file IO, prevent these malicious and wrong IO in order to enhance the security of the container and fault tolerance. Experiments show that the proposed scheme can improve the security and fault tolerance of the container while only about 20% performance loss, so it has usability.

Acknowledgments
This work was funded by the National Natural Science Foundation of China (61521003) and the National Key Research and Development Project (Grant No.2018YF0804003, and Grant No.2017YFB0803204), and thanks to the National Digital Exchange System Engineering Technology Research Center for funding.

References
[1] Nguyen Q T, Tuong N H. Virtual Machine Allocation in Cloud Computing for Minimizing Total Execution Time on Each Machine [ C ] / / Proceedings of International Conference on Computing, Management and Telecommunications. Ho Chi Minh City, Vietnam:[ s. n. ], 2013:241-245
[2] WANG Xiong. The History and Advantages of Cloud Computing[J]. Computer & Network, 2019, 45(2): 44.
[3] LI Wenjun. Analysis of Computer Cloud Computing Andits Implementation Technology[J]. Dual Use Technologies & Products, 2018, 18(22): 57-58.
[4] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, An updated performance comparison of virtual machines and Linux containers, presented at IEEE Int. Symp. Performance Analysis of Systems and Software, Philadelphia, PA, USA, 2015.

[5] B. Xavier, T. Ferreto, and L. Jersak, Time provisioning evaluation of KVM, Docker and unikernels in a cloud platform, in Proc. 16th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing, Cartagena, Colombia, 2016.

[6] Yang N, Liu J. Optimized fault tolerance as services provisioning for cloud applications. Ruan Jian Xue Bao/ Journal of Software, 2019,30(4):1191-1202 (in Chinese). http://www.jos.org.cn/1000-9825/5372.htm

[7] J. Okajima, Aufs5-advanced multi layered unification filesystem version 5.x, http://aufs.sourceforge.net/, 2013.

[8] WU J X. Endogenous security of cyber space:mimic defense and generalized robust control (volume 1)[M]. Beijing: Science Press, 2020.

[9] B. Cox, D. Evans, A. Filipi, et al. N-Variant Systems: A Secretless Framework for Security through Diversity[C]. USENIX Security Symposium (SEC'06),2006: 105-120.

[10] Salamat B., Jackson T., Gal A., et al. Orchestra: Intrusion Detection Using Parallel Execution and Monitoring of Program Variants in User-space[C]. Proceedings of the fourth ACM european conference on Computer systems - EuroSys '09, 2009: 33-49.

[11] Koning K., Bos H., Giuffrida C.. Secure and Efficient Multi-Variant Execution Using Hardware-Assisted Process Virtualization[C]// Proc. of the Int'l Conf. on Dependable Systems and Networks. IEEE, 2016.

[12] Wright C P, Zadok E. Unionfs: Bringing Filesystems Together [ EB/ OL ]. ( 2004-11-28 ). http://www.linuxjournal. com/ article / 7714.