A Container-based Trusted Multi-level Security Mechanism

Xiao-Yong LI[1], Chen JI[1,a] and Gang LIU[2]

[1]*School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China*
[2]*Information Technology Center, China Railway Corporation, Beijing, China*

Abstract. Multi-level security mechanism has been widely applied in the military, government, defense and other domains in which information is required to be divided by security-level. Through this type of security mechanism, users at different security levels are provided with information at corresponding security levels. Traditional multi-level security mechanism which depends on the safety of operating system finally proved to be not practical. We propose a container-based trusted multi-level security mechanism in this paper to improve the applicability of the multi-level mechanism. It guarantees multi-level security of the system through a set of multi-level security policy rules and trusted techniques. The technical feasibility and application scenarios are also discussed. The ease of realization, strong practical significance and low cost of our method will largely expand the application of multi-level security mechanism in real life.

1 Introduction

Multi-level security (MLS) refers to the ability to process information with different security levels in a computer system. It was originally used to support the security and confidentiality of military systems and databases. In a multi-level security system, the subject can't read the object of security level is higher than it and can't write the object of security level is lower than it. The implementation of multi-level security system can effectively guarantee the confidentiality of the application software and data, restrict illegal attacker, and prevent attackers from arbitrarily destructing system information. Moreover, it can also be easy to configure and manage the system policy.

In the past research, there are a lot of achievements about MLS have been made. Many research institutions and scholars try to support multiple security through a platform based on secure operating system, such as Adept-50, safety Xenix, SE-Linux, etc. However, the current development statuses of these systems prove that it was not successful in practical application. In order to resolve these problems, this paper proposes a trusted multi-level security container mechanism. This mechanism is based on lightweight virtualization technology and reference the idea of multiple independent levels of security and safety (MILS)[1, 2]. In section 3, we also prove the safety of it.

The innovations of this paper are: 1) This mechanism can be used directly in most commercial systems, and improve the problem of poor usability of traditional MLS mechanism; 2) the implementation of this method is not only simple but also diverse. It can meet different MLS requirements, and has a strong practical significance; 3) the safety of the trusted multi-level security container mechanism is proved in this paper, and the credibility of this method is improved.

[a] Corresponding author: 14120352@bjtu.edu.cn

In Section 2 we introduce the related research, including the latest research of MLS, lightweight virtualization technology and trusted system. According to the actual requirements of security applications, we propose the definition of multi-level security based on the lightweight virtualized environment in Section 3, and then propose a set of multi-level security policy rules. We also prove that these rules meet the definition of multi-level security, and analyses technical feasibility of the mechanism. We present several application scenarios of this mechanism in Section 4, and conclude this paper in Section 5.

2 Related works

2.1 Container and lightweight virtualization technology

Lightweight virtualization is relative to the heavyweight virtualization terms. Heavyweight virtualization usually refers to the KVM, VMware, etc. which based on the virtualization of hardware. Lightweight virtualization based on container technology refers to the operating system-level virtualization technology, such as Docker [3], Linux Containers (LXC) and so on.

Recently, most of cloud service providers provide a complete operating environment serves to running applications by using traditional virtualization technology, and the entire operating system running on a virtual hardware platform. But this approach greatly consumes and wastes systems resources, brings greater cost, and has higher demands on the server, so more and more enterprises are choosing to use lightweight virtualization technology. Lightweight virtual machines (called "container") can load applications directly on the host platform, use the same kernel with the host, and lead to a small performance loss. Lightweight virtualization technology uses lightweight isolation. In order to share resources between the host and containers, it also provides a shared mechanism.

In summary, for one host, using lightweight virtualization will virtualize more virtual machines than using traditional virtualization. It greatly improve the resource utilization and system overhead, and is a more economical way for enterprise and cloud providers.

2.2 Multi-level security

According to the importance and sensitivity of the information, the information will be divided into different security level. By dividing security level, multi-level security policies can ensure that the information can only be used by the people whose security level is higher than or equal to it. There are a lot of well-known multi-level security models have been proposed, such as BLP [4], Biba [5], clark-Wilson [6] and so on. In 2005, Jim Alves-Foss et al. propose the concept of multiple independent levels of security and safety (MILS). MILS is originally based on the Rushby's idea of isolation [7, 8], and is a high-security system. It can build a set of credible security solution by using several management units which can be independent analysis.

In addition to these classic multi-level security model, the development of cloud computing and virtualization technology provide more favorable conditions for a multiple single-level (MSL) system's implementation. MSL [9] use isolated computing or virtual machine for different security level data to separate these data. The purpose is to implement the multi-level security mechanism without special modification of operating systems or applications. References [10] propose an action-based multi-level access control (AMAC) model and gives the formal description of the policy. This model considers many information such as characteristics of structured document, changes of document's security level, changes of the user's security level, system status information and so on. It implement multi-level security access to structured documents in a cloud computing environment, and prove the security of multi-level access control policies of AMAC model. References [11] proposed a multi-level security mechanism based on virtualization technology. It avoids cross-contamination issues and hidden channels of different security levels through strong security isolation mechanism,

and reduce difficulty and costs of multi-level security's implementation by its seamless support for existing commercial operating systems and applications.

2.3 Trusted system

Trusted system can guarantee credibility from two parts: trusted source and trusted running. First, the source of the system determines the requirements and methods to ensure credibility of the system. For a third-party system, the credibility of the system can be achieved through process transparency [12] of system behavior specification's production and management, and the user can also refer to other users' reviews to evaluate the credibility of the system [13]; for the system developed by users themselves, possible errors and vulnerabilities in the system design process should be reduced or eliminated as much as possible. For example, formal description [14] can be used to regulate the production process and code size control [15] can be used to reduce the possible risks.

In addition to the problem of trusted sources, the trusted running is also important. We can use trusted computing platform (TCP) technology to ensure the trusted running [16], and ensure the system running as expected by a trusted root and trusted delivery mechanisms.

3 Container-based trusted multi-level security mechanism

3.1 Confidentiality Requirements of multi-level security

Multi-level security confidentiality policies divided the information into different security levels based on the importance and sensitivity of the information. The information can only flow from low security-level to high security-level. That is, a subject with high security level can read an object with low security level, a low security level subject can write a high security level object, and a subject can read or write an object if there security levels is equal. By this way, it can ensures that the information can only be one-way flow from low to high, and avoids the disclosure of information from high to low.

Depending on the above, we propose formal definition of the Multi-level security confidentiality requirements. This paper use S denotes the set of all domains of the safety system, use A denotes the set of entire address space of the security system, and use L denotes a set of security tags which have a partial order relation$\preccurlyeq$. $l_1 \preccurlyeq l_2$represents $l_2$has a higher security level than l_1. Function level: S → Ldenotes the security level of domain S. Function alter: S → A denotes the address space which is writable by domain S. Function observe: S → A denotes the address space which is readable by domain S.

Definition 1 For $\forall u,v \in S$ and $level(u) \preccurlyeq level(v)$, we say that the security policy meets the multi-level security confidentiality requirements if

$$alter(u) \cap observe(v) \neq \emptyset \tag{1}$$

$$observe(u) \cap alter(v) = \emptyset \tag{2}$$

For two domains with different security levels, if the intersection of the writable address space of the low security-level domain and the readable address space of high security-level domain is not empty, the information of the low security-level domain can flow to the high security-level domain. If the intersection of the readable address space of the low security-level domain and the writable address space of high security-level domain is empty, the low security-level domain can't read the information which is written by a high security-level domain. All above meet the basic idea of multi-level security confidentiality requirements.

There are differences between definition 1 and traditional models of multi-level security. Definition 1 supports the idea that low security-level can preserve their privacy. As shown in Figure 1, M and N are two arbitrary domain and N has higher security-level than M. $M = \{M_c, M_p\}$ is a division, in which N can read area M_c and can't read M_p. It means that M_p is M's private area. In this case, M

can not only report information to N through M_c, but also protect its privacy by M_p, which meets the practical safety requirements of many systems. If $M_c = M$, N can read all the information in M.

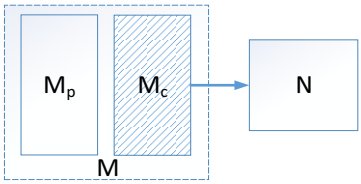

Figure 1. Schematic diagram of multilevel security policy.

3.2 Container-based trusted multi-level security mechanism

This section proposes a trusted multi-level security container mechanism. The purpose of the mechanism is to support that information flow between different security level containers meets the confidentiality requirements of definition 1.

1) Division of address space

In the lightweight virtualization environment, the system is divided into two domains: a) container domain: It is the private portion of each container, and a container domain usually corresponds to a container; b) host machine domain: The other parts of the system except for container domains. In this paper, different domains can access different address spaces according to their permissions.

According to the division of the domain, The system address space is divided into two categories: a) container domain address space (C_i): It is assigned to different container domains; b) host domain address space: including all address space of the system except for container domain address space, which is assigned to host domain. Because of different container domains has different permissions for the part of host domain address space, the host domain address space are further divided: a) host-sharing area (SA), each container can read the contents of this area, such as images, related information of basic software and so on; b) host-container area (CA_i): The content of this area is only related to the specific container, only corresponding container domain can read or write the corresponding host-container area, each container corresponds to a host container domain; c) host-private area (PA): No containers can read or write this area. It can only be accessed by the host. The four parts is a division of the system address space, the address space in each part is not overlapping, as shown in Figure 2.

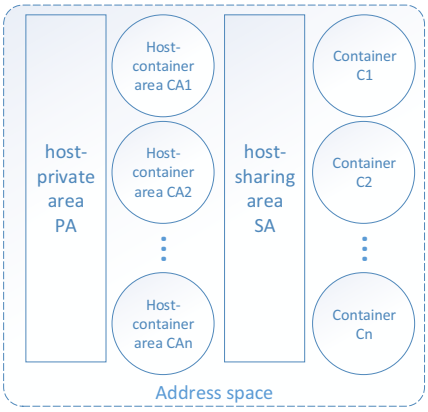

Figure 2. Address space partition in lightweight virtual environment

2) Security policy and proof

Based on the division of the address space in Figure 2, the container environment is defined as follows:

In this paper, we use $\mathrm{Container}(S, A)$ denotes the container environment $S = \{HD, CD_1, CD_2, \ldots, CD_n\}$. Among them, HD denotes the host machine domain, $CD_i (1 \le i \le n)$ denotes the container domain i and CD represents the set of all container domains. $A = \{SA, \mathrm{PA}, CA_1, CA_2, \ldots, CA_n, C_1, C_2, \ldots, C_n\}$, where SA denotes the set of address space of host-sharing area, PA denotes the set of address space of the host-private area, $CA_i (1 \le i \le n)$ denoted the address space of host-container area which can be accessed by the corresponding container domain i, and $C_i (1 \le i \le n)$ denotes the set of container domain address space which is corresponding to container domain i.

According to definition and, the following rules about trusted multi-level security policy are proposed:

Rule 1:

$$\forall i,,1 \le i \le n, C_i \cup CA_i \cup SA \subset observe(CD_i) \tag{3}$$

Rule 2:

$$\forall i,,1 \le i \le n, alter(CD_i) = C_i \cup CA_i \tag{4}$$

Rule 3:

$$\forall i,j,1 \le i \le n, 1 \le j \le n, i \ne j,\ level(CD_i) \preccurlyeq level(CD_j)$$
$$\Rightarrow C_i \subset observe(CD_j) \wedge C_i \not\subset alter(CD_j) \tag{5}$$

Rule 4:

$$\forall i,j,1 \le i \le n, 1 \le j \le n, i \ne j,\ level(CD_i) \preccurlyeq level(CD_j)$$
$$\Rightarrow C_j \not\subset observe(CD_i) \wedge C_j \not\subset alter(CD_i) \tag{6}$$

Rule 5:

$$\forall i,j,1 \le i \le n, 1 \le j \le n, i \ne j,\ CA_i \not\subset observe(CD_j) \wedge CA_i \not\subset alter(CD_j) \tag{7}$$

Rule 1 means that the readable address spaces of a container domain include the host-container area, host-sharing area, and the container domain address space which is corresponding with it. Rule 2 means that the writable address spaces of a container domain include corresponding container domain address space and corresponding host-container area. Here container domain CD_i can be understood as the domain M in Figure 1, and C_i, CA_i can be understood as M_c, M_p. Rule 3 means that if the security level of two container domains is different, the container domain with high security level can only read the container domain address space of the container domain with low security level, and can't write it. Rule 4 means that if the security level of two container domains is different, the container domain with low security level can neither read nor write the container domain address space of the container domain with high security level. Rule 5 means that for an arbitrary host container area, in addition to corresponding container domains, the other container domains cannot read and write the address spaces of these areas. The following is a proof of the security.

Proof:

According to the definition 1, for $\forall u, v \in S$ and $level(u) \preccurlyeq level(v)$, we say that the security policy meets the multi-level security confidentiality requirements if

$$alter(u) \cap observe(v) \ne \emptyset \tag{8}$$

$$observe(u) \cap alter(v) = \emptyset \tag{9}$$

So, for the trusted multi-level secure container policy, it is needed to prove that:

For $\forall i,j, 1 \le i \le n, 1 \le j \le n, i \ne j$ when $level(CD_i) \leqslant level(CD_j)$

$$alter(CD_i) \cap observe(CD_j) \ne \emptyset \tag{10}$$

$$observe(CD_i) \cap alter(CD_j) = \emptyset \tag{11}$$

1). According to the rule 2,

$$alter(CD_i) = C_i \cup CA_i \tag{12}$$

$$\because \forall i,j, 1 \le i \le n, 1 \le j \le n, i \ne j,$$

$$level(CD_i) \leqslant level(CD_j) \tag{13}$$

According to the rule 3,

$$C_i \subset observe(CD_j) \tag{14}$$

$$\therefore C_i \subset alter(CD_i) \wedge C_i \subset observe(CD_j) \tag{15}$$

$$\therefore alter(CD_i) \cap observe(CD_j) \ne \emptyset \tag{16}$$

$\therefore$（10）is proved

2). According to the rule 2,

$$alter(CD_j) = C_j \cup CA_j \tag{17}$$

$$\because \forall i,j, 1 \le i \le n, 1 \le j \le n, i \ne j,$$

$$level(CD_i) \leqslant level(CD_j) \tag{18}$$

According to the rule 4,

$$C_j \not\subset observe(CD_i) \tag{19}$$

According to the rule 5,

$$CA_j \not\subset observe(CD_i) \tag{20}$$

$$\therefore observe(CD_i) \cap alter(CD_j) = \emptyset \tag{21}$$

$\therefore$（11）is proved

In conclusion, the trusted multi-level security container policy proposed meets the multi-level security requirements.

In addition, we can also ensure the credibility of the method from two aspects: trusted source and trusted running. On the one hand, we can ensure the credibility by using trusted computing platform technology when the system is running. Trusted computing platform technology use trusted platform module (TPM) as the core, and ensure the trusted running through the trust transfer mechanism, identity authentication, safe storage mechanism and so on. On the other hand, because this method is developed and designed independently, we can use the idea of reference monitor to complete the

design and development, and as far as possible to reduce the size of key code in the process of implementation.

3.3 Feasibility analysis

According to the discussion in the section 3 and the structural characteristics of this method, we can use the union file system[17] (UnionFS) technology to implement the mechanism. UnionFS is a layered, lightweight and high performance file system, which can merge multiple file systems together and allowing the coexistence of read-only and read-write directories. By using the copy-on-write function, it allows virtual modification of the read-only file system, and the real modification will be saved to the read-write file system. The principle is shown in Figure 3.

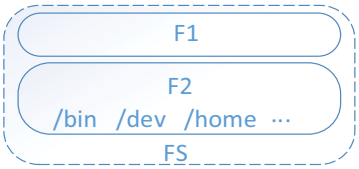

Figure 3. Schematic diagram of UnionFS

F1 and F2 formed a union file system FS through the method 'mount', where F1 is considered as a read-write layer and F2 is considered as a read-only layer. FS can only read the contents of F2. When FS wants to modify the file in F2, F1 will creates a file corresponding to the file in F2 and FS is actually modify the F1.

The trusted multi-level secure container mechanism proposed in section 3 can be implemented by using UnionFS technology. As shown in Figure 1, we can considered the file system Mc as the read-only layer of the UnionFS, and file system N itself as the read-write layer of the UnionFS. In this way, all changes of Mc can reflected in the file system N, and M can't know any changes in N. All above meet the requirements of the multilevel security model.

For example, Docker is a typical container-based lightweight virtualization system, the AUFS (Another UnionFS) it uses is a union file system. In addition to AUFS, the union file system types that Docker supports also include btrfs, vfs and DeviceMapper. Therefore, in most of the lightweight virtualization systems, such as docker, we can use the characteristics of the union file system to implement the trusted multi-level security container mechanism directly.

4 Scenarios

Trusted multi-level security container mechanism combines the traditional multi-level security policy and lightweight virtualization environment, and using the formal method proves its security, which has a strong application significance. In the reference[18], Rushby is divided multi-level security policy into two types: transitive model and intransitive model. The two kinds of models can be implemented by this mechanism. The following are explanation of implementation of these two models.

1) Transitive model

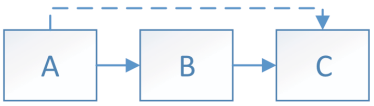

Figure 4. Transitive model

As shown in Figure 4, for three different domains A, B and C, the security level relationship of A，B，C is A<B<C. A ⇝ B denotes the information can flow from domain B to domain A. Then, the model can be referred to as A ⇝ B, B ⇝ C, A ⇝ C. The file A can be considered as the read-only layer of domain B, and the file system is considered as the read-only layer of domain C. According to the characteristics of the union file system, domain B can know the changes in domain A, domain C can know the changes in domain A and domain B. Conversely, the lower level domain will not aware of the changes in high level domain.

This implementation way has more practical significance than the traditional BLP model. In practical application, the lower security level user can report information to the higher security level user, but it doesn't mean that he can write something on the contents of the higher security level user. For the organization with security requirements, the high security level user can know the low security-level user's modify of files, and low security level user can not know the high security level user's file system. In addition, high security level user can modify files of low security level user, and this modification will not affect the original file system of low security level user.

2) Intransitive model

Figure 5. Intransitive model (1)

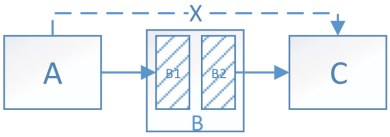

Figure 6. Intransitive model (2)

As shown in Figure 5, for three different domains A, B and C, the non-transitive model can be referred to as A ⇝ B, B ⇝ C, A ↛ C. According to characteristics of non-transitive model, B divided into two disjoint domains: B1 and B2. A is considered as the read-only layer of B1, and B2 is considered as the read-only layer of C, as shown in Figure 6. B acts as a middleman or arbiter to judge the information in B1. If the information meet the requirements of C to read, B will copy the information of B1 to B2, then C can read these information. If not, nothing to do. In this way, the information can flow from A to B, from B to C, but can't directly flow from C to A. This model is widely used in practical applications. B can be considered as a confidential department of a unit or organization. It can examine and verify the information of user A, and judge whether user C can read the information. If the requirement is met, the information from A will be written in the area which is readable for user C. This method can avoid the information leakage caused by the direct communication between user A and C.

Conclusion

Trusted multi-level security container mechanism is aimed at the problem of data protection in container-based lightweight virtualization environment, overcome the limitations of the current mainstream MLS in application. At the same time, the trusted multi-level security container mechanism adopt formal description and other trusted computing technologies to ensure the

credibility of this method. In addition, the trusted multi-level security container mechanism fully take into account the implementability and flexibility in the practical application, and has a strong guiding significance in design and implementation of the relevant multi-level security container system.

References

1. Alves-Foss J, Oman P W, Taylor C, et al. The MILS architecture for high-assurance embedded systems.[J]. International Journal of Embedded Systems, 2006, 2(3/4):239-247.
2. W. Scott Harrison, Nadine Hanebutte, Paul W. Oman, Jim Alves-Foss. The MILS Architecture for a Secure Global information Grid [J].The Journal of Defense Software Engineering, 2005(10):20-24.
3. Anderson C. Docker[J]. IEEE Software, 2015, 32(3).
4. Bell D E, Padula L J L. Secure Computer System: Unified Exposition and Multics Interpretation[J]. Secure Computer System Unified Exposition & Multics Interpretation, 1976:161-161.
5. Biba K J. Integrity Considerations for Secure Computer Systems[J]. Electronic Systems Div Air Force Hanscom Afb, 1977.
6. Clark D D, Wilson D R. A Comparison of Commercial and Military Computer Security Policies[C]// IEEE Symposium on Security & Privacy. IEEE, 1987:184-184.
7. Rushby J M. Design and verification of secure systems[M]// ACM SIGOPS Operating Systems Review. 2010:12-21.
8. Rushby J M. Proof of separability A verification technique for a class of security kernels[M]// International Symposium on Programming. 1970:352-367.
9. Wikipedia editor. MSL. Available from: https://en.wikipedia.org/wiki/Multiple_single-level.
10. Xiong Jinbo, et al. Action-Based Multilevel Access Control for Structured Document [J]. Journal of Computer Research and Development, 2013, 50(7):1399-1408.
11. Cheng Jing, Shi Yong. Study on Multilevel Security Mechanism based on Virtualization[J]. Communications Technology, 2013(1):35-39.
12. Chen Qingzhang. Characteristics of transparent environment[J]. Application Research of Computers, 1990(5):18-19.
13. Jurca R, Faltings B. Eliciting Truthful Feedback for Binary Reputation Mechanisms[C]// Ieee/wic/acm International Conference on Web Intelligence. IEEE Computer Society, 2004:214-220.
14. Kelly J C, Lead T, Kemp K, et al. formal methods specification and verification guidebook for software and computer systems volume i: planning and technology insertion[J]. 1998.
15. Anderson J P, Anderson J P. Computer Security Technology Planning Study. Volume 2[J]. Computer Security Technology Planning Study, 1972.
16. Group T. TCG specification architecture overview. TCG Specification Revision. 2007;1:1-24.
17. Quigley D, Sipek J, Wright C P, et al. Unionfs: User and Community-Oriented Development of a Unification File System[J]. Proceedings of the Linux Symposium, 2006:349--362.
18. Rushby J. Noninterference, Transitivity, and Channel-Control Security Policies[J]. 1992.