Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2023.026371

ARTICLE

PIMS: An Efficient Process Integrity Monitoring System Based on Blockchain and Trusted Computing in Cloud-Native Context

Miaomiao Yang[1,2], Guosheng Huang[1,2], Junwei Liu[3], Yanshuang Gui[1,2], Qixu Wang[1,2,*] and Xingshu Chen[1,2]

[1]Cyber Science Research Institute of Sichuan University, Chengdu, 610207, China

[2]Cyber Science and Engineering of Sichuan University, Chengdu, 610207, China

[3]China Mobile (Suzhou) Software Technology, China Mobile, Suzhou, 215163, China

*Corresponding Author: Qixu Wang. Email: qixuwang@scu.edu.cn

Received: 01 September 2022 Accepted: 25 October 2022

ABSTRACT

With the advantages of lightweight and high resource utilization, cloud-native technology with containers as the core is gradually becoming the mainstream technical architecture for information infrastructure. However, malware attacks such as Doki and Symbiote threaten the container runtime's security. Malware initiates various types of runtime anomalies based on process form (e.g., modifying the process of a container, and opening the external ports). Fortunately, dynamic monitoring mechanisms have proven to be a feasible solution for verifying the trusted state of containers at runtime. Nevertheless, the current routine dynamic monitoring mechanisms for baseline data protection are still based on strong security assumptions. As a result, the existing dynamic monitoring mechanism is still not practical enough. To ensure the trustworthiness of the baseline value data and, simultaneously, to achieve the integrity verification of the monitored process, we combine blockchain and trusted computing to propose a process integrity monitoring system named IPMS. Firstly, the hardware TPM 2.0 module is applied to construct a trusted security foundation for the integrity of the process code segment due to its tamper-proof feature. Then, design a new format for storing measurement logs, easily distinguishing files with the same name in different containers from log information. Meanwhile, the baseline value data is stored on the blockchain to avoid malicious damage. Finally, trusted computing technology is used to perform fine-grained integrity measurement and remote attestation of processes in a container, detect abnormal containers in time and control them. We have implemented a prototype system and performed extensive simulation experiments to test and analyze the functionality and performance of the PIMS. Experimental results show that PIMS can accurately and efficiently detect tampered processes with only 3.57% performance loss to the container.

KEYWORDS

Blockchain-based protection; dynamic monitoring; remote attestation; integrity verification

1 Introduction

Cloud-native is a software approach to building, deploying, and managing modern applications in a cloud computing environment. With the development of cloud-native technology with containers

at its core, cloud computing is undergoing profound changes from technical architecture to business models [1–4]. The container has the advantages of being lightweight and providing a consistent operating environment. Hence, container-based applications are being deployed on a large scale in cloud computing platforms. However, containers face many security threats due to the shared kernel and low resource isolation strength of container technology [5–8]. When the container is running, it is more vulnerable to malware attacks [9]. Because of the vulnerability of container technology itself, this allows attackers to take advantage of it. Attackers have exploited the vulnerability to achieve their attack purpose and gain super privileges, leading to a severe compromise of the security of container clusters [10]. Therefore, how to secure containers has become a concern for researchers.

As one of the mainstream technical architectures for cloud platform infrastructure, it faces various security threats, such as kernel sharing, weak isolation measures, and tampering with container images [11]. While the container is running, a malicious program inside the container can attack the services inside the container and break through the weaker isolation measures to directly attack the co-resident container on the host [12]. At the same time, the reliability of the security software on the container cloud platform cannot be assessed due to the lack of essential physical security support. Consequently, it is trapped in the state of "protecting one software by another" and loses the basis of trust [13]. In conclusion, protective measures with physical security foundation support are essential for the program's correct operation.

In order to provide a single trusted physical foundation, the Trusted Computing Group (TCG) proposed the Trusted Computing Technology and designed the Trusted Platform Module (TPM). TPM is a small tamper-proof hardware chip embedded in the motherboard. It provides trusted hardware roots and offers cryptographic services [14]. In the Infrastructure as a Service (IaaS) cloud computing platform, mainly for virtual machines, the trustworthiness evaluation scheme of the target system based on trusted computing technology has been widely studied and applied. However, the trustworthiness evaluation scheme for container clouds needs further research. Based on trusted computing technology, a virtual trusted root is configured for each container [9]. Although a certain degree of trustworthiness assessment can be achieved, container instances in a single server in a container cloud are typically in the tens [15]. Thus, a significant performance loss is caused. In addition, some scholars have also implemented an evaluation scheme based on the Integrity Measurement Architecture (IMA) supported by TCG [16]. Since IMA is already a kernel subsystem of Linux, it is possible to take full advantage of the kernel features of Linux to implement integrity measurement of files and programs. Unfortunately, the IMA measurement occurs at program load time and does not enable monitoring of processes. Therefore, it cannot realize the security protection of the running state of the container. Besides, the IMA measurement log cannot distinguish between files of the same name in different containers.

Remote attestation is one of the essential functions of trusted computing technology. It is the process by which a TPM proves its trustworthiness to external entities [17,18]. Compared with traditional authentication, remote attestation further extends the content of verification. So the verification parties can perform deeper and more detailed authentication. The baseline value library data is vital to the remote attestation process. However, it is usually stored in plaintext or downloaded to a local native-based security mechanism for protection. Despite the ease of access through this storage method, the integrity of the baseline value data relies on strong security assumptions [19].

Blockchain technology has the characteristics of decentralization, immutability, and traceability [20]. Thus, blockchain can solve the problem of data tampering and falsification of the baseline value database [21]. Also, because blockchain technology can record all transactions and audit trails, it

ensures the transparency of the remote attestation process and effectively addresses the problem of corrupted baseline value data [22].

To address the runtime integrity protection of applications within Docker in container cloud environments, and to resolve the reliance on strong security assumptions for baseline value data. From the perspective of integrity monitoring, we propose a dynamic monitoring method for the integrity of process code segments. Firstly, we studied the way memory is managed in Linux. According to the characteristics of Linux OS memory management and based on the process runtime information provided by the procfs file system, we performed a paged measurement of the code segments in the container. Secondly, we investigated how the existing IMA's trusted base is constructed and how the TPM module is invoked. Furthermore, the process's metric values are protected by utilizing the security features provided by TPM. Thirdly, we build a baseline value library of paged code segments using the ELF files corresponding to the processes to realize the recoverable traceability of the measurement results in the integrity verification stage. We also used the tamper-evident property of the blockchain to store the baseline value library data on the blockchain. Finally, to satisfy the need to distinguish programs with the same name from different containers in the integrity verification phase, we designed the system's storage measure log format by adding container image names as distinguishing labels.

In a nutshell, the main contributions of this paper are summarized as follows:

- We propose an expeditious integrity measurement method to enhance container runtime security. The method implements a dynamic measure of the process integrity, while ensuring the trustworthiness of the dynamic metric values with the security features of the hardware TPM.
- We propose a novel method for constructing and storing the baseline value library. The method establishes the baseline value library by paging measurement of ELF files and stores the baseline values on the blockchain, providing a convenient access mechanism to the baseline value data while ensuring integrity.
- We have implemented a prototype system, PIMS, that allows real-time monitoring of containers in a running state. The evaluation results show that the system can accurately and efficiently verify processes with compromised integrity, helping cloud platform managers promptly control anomalous containers.

The rest of the paper is organized as follows: Section 2 introduces related work. Section 3 presents the related knowledge. Section 4 overviews the PIMS design goals, architecture, and attack model. In Section 5, the detailed design of PIMS is described. Section 6 gives the integrity analysis of baseline value data, as well as the performance evaluation and discussion of PIMS. Finally, Section 7 concludes our work.

2 Related Work

This section reviews the related work in the direction of integrity measurement and remote attestation of applications inside containers.

2.1 Application Integrity Measurement in Container

Existing schemes for measuring the integrity of applications in the user layer or container can be broadly classified into two categories.

(1) The first type is the implementation of the Virtual Trusted Platform Module (vTPM) in a higher privilege level component or entity. The vTPM is then used to implement measurement operations or other functions.

Hosseinzadeh et al. [23] proposed two options by referring to the implementation of virtual trusted computing technology in virtual machines. (a) A vTPM is implemented in the kernel as a kernel module for each container; (b) vTPM is implemented for each container in the privileged container, and the normal container communicates with the simulated device through the front and back ends. Guo et al. [24] implemented the (a) scheme. Moreover, they generate a signature list for the executable files in the container based on the container image, and bind the list to the container image as a whitelist of runnable programs in the container to achieve a trusted start of the container. Since a single server in a container cloud typically runs dozens of container instances simultaneously [15]. Moreover, containers also need to be started and destroyed quickly. Therefore, equipping every container with vTPM can seriously affect the system's availability. In conclusion, the above schemes [23,24] are not practical enough in real container cloud scenarios.

Tian et al. [15] used Intel SGX technology to implement the software emulated TPM module named tpmsgx in Enclave, and the LXC container communicates with the emulated software through socket sockets. However, this scheme uses tpmsgx only as a library for providing secure cryptographic algorithms to applications locally or in containers during runtime, and is not designed for integrity measurement.

(2) Another type is to rely on the existing IMA architecture of Linux, and then design other auxiliary measures to achieve the integrity measurement of the application.

The integrity measurement architecture IMA proposed by Sailer et al. [16] is the classical application integrity measurement scheme. IMA is a mandatory access control solution based on Linux's Linux Security Module (LSM). It measures, stores, and evaluates the integrity of files before they are opened or programs are loaded into memory by monitoring system calls such as open, mmap, and exec. In addition, the IMA generates the corresponding measurement log information each time a measurement operation is performed. The measurement log records information such as the metric value and path of the measured file. Furthermore, it extends the generated integrity information to the Platform Configuration Register (PCR) of the TPM. Nevertheless, this traditional host integrity measurement scheme is overwhelming in the container cloud scenario. Because it can only implement measurement for file static data and cannot distinguish files of the same name in different containers in the measurement log, it is challenging to verify the integrity.

de Benedictis et al. [6] also implemented DIVE, a container integrity verification engine based on IMA. DIVE first measures the files in the container through IMA. Then, the device number tag of the container instance is generated based on the Devicemapper storage driver. Finally, this device number tag is applied to the IMA storage measurement log to distinguish between applications in different containers. However, Docker's current default storage driver is overlay2, and all Linux distributions support overlay2. In contrast, Devicemapper is only supported by default by RHEL and CentOS, and Devicemapper performance is poor [25]. In addition, overlay2 does not map the container's file system to a device, so the use of this scheme is limited in scope.

In all of the above literature, the measurement timing chosen for the integrity measurement scheme for in-container applications is before the program is loaded into memory, and the measurement object is static text data. While Liu et al. [26] pointed out that static text integrity measure cannot guarantee the integrity of the running state. Subsequently, for the integrity problem of program runtime,

Pan [27] proposed an integrity measurement method based on process address space code segments. This method provides a new idea for the integrity measurement of container processes. This method implements the hardware TPM function through the TPM Emulator software simulation. However, the metric values are stored in the virtual PCR module, which lacks hardware security foundation support, and the security of the emulator software itself cannot be guaranteed. Therefore, the method cannot effectively guarantee the credibility of the metric values.

2.2 The Remote Attestation

The remote attestation technique, first proposed by TCG, has become the primary method for assessing the trustworthy state of a target system. In most of the existing disclosed schemes [19,28], the baseline value data is stored in the cloud management center. Moreover, it is assumed that the cloud hypervisor is absolutely trustworthy. However, this assumption is unrealistic in practical applications where, for example, the behavior of cloud service provider staff is not controlled by the subscriber. Consequently, the security of the baseline value database data is clearly an issue that cannot be ignored.

The protection of data can be done in many ways [29–32]. Among them, the academic field widely recognizes blockchain technology for its distributed and multi-copy mechanism that brings the advantage of tamper-proof data on the chain [32–34]. Ritzdorf et al. [33] proposed an extension to the TLS protocol called TLS-N that provides non-repudiation proofs for each TLS session. The protocol uses a generator to generate signed evidence that can be verified by any third party via public key infrastructure. Pavithran et al. [34] proposed an IoT and blockchain architecture using mobile edge cloud computing. They use decentralization to achieve confidentiality, integrity, and availability of data. Yet, the solution requires various performance requirements, such as the edge devices' storage capacity and computing power.

In summary, to address the problem of malicious tampering during container runtime in cloud environments, we have achieved a fine-grained measurement of the integrity of process code segments in containers using trusted computing technology. At the same time, the trustworthiness of the baseline value library data is ensured with the help of blockchain technology. Finally, the integrity of the processes in the container is effectively guaranteed by remote attestation technology.

3 Background

This section briefly introduces trusted computing, integrity measurement architecture, and blockchain-related knowledge, respectively.

3.1 Trusted Computing Foundation

TCG defines trustworthiness as "an entity is trustworthy if it always behaves expectedly, toward the desired goal" [14]. To achieve the goal of trustworthiness, a mechanism for storing and verifying evidence information based on tamper-evident hardware is required. For this reason, TCG has designed TPM. TPM enhances the system's security protection capability through trusted hardware. At first, it ensures the security of components in the trust chain through the secure storage of physical trust roots and the trust chain establishment mechanism. At the same time, it ensures that the code integrity of critical components in the system is not damaged. Then, the confidentiality of user data is guaranteed through the data security storage mechanism supported by the physical trust module. Finally, the integrity of the components on the trust chain of the target platform is tested through the remote attestation mechanism, and the trustworthiness status of the target platform is evaluated. In

a nutshell, in terms of functional delineation, TPM has the functions of ensuring secure data storage, application integrity, and remote attestation of the platform [35].

TPM is a System on Chip (SOC) [36]. According to the design of the TCG, the TPM has a protected storage area dedicated to the Platform Configuration Register (PCR). Each TPM has at least 24 PCRs, numbered from 0 to 23, and each PCR is 20 bytes (e.g., SHA-1 digest size). To prevent PCR values from being tampered with by malicious code, the TPM hardware restricts the behavior of operations on the PCR. Indeed, only two operations are allowed to modify the PCR value: the Reset operation and the Extend operation. The Reset operation restores the PCR to its default value after the machine is powered off or restarted. The Extend operation is performed in the following ways:

$$New\ PCR_i = SHA - 1\ (Old\ PCR_i || new\ Value) \tag{1}$$

where the symbol || represents the connection and the letter i represents the index value of the PCR register.

For each component, the metric is used as a unique identifier and proof that integrity has not been compromised. Generally, software components are "measured" by the binary verification method, i.e., by calculating the summary value of the binary file. Integrity measures are used to assess the level of trust in the platform. Each component in the system startup process measures the component to be loaded before transferring control of the platform. At the same time, these results obtained by measurement are stored in the PCR. This measurement process is defined as trust chain transfer, which is the basis for remote attestation.

3.2 Integrity Measurement Architecture

Integrity Measurement Architecture (IMA) was first implemented by IBM research as a trusted computing-based measurement system [16]. Since version 2.6.30, it has been integrated into the Linux kernel. IMA maintains integrity measurement logs that conform to the TCG specification. It is also the first time that the concept of TCG trust measurement has been extended from BIOS to the application layer, which makes remote attestation techniques applicable in real-world scenarios [37]. IMA policies can constrain the types of measurable events, such as memory-mapped and executable files. When the IMA measurement function is activated, first, each time the operating system opens a file or executes a program, it performs a measurement operation on that file or program. Subsequently, the measurement results are extended into PCR 10 of the TPM module. Finally, the records are stored in the metric log imaSML. When IMA's evaluation function is activated, the measurement results are compared with the baseline values, and inconsistencies interrupt the opening or execute the operation. This way, it ensures that the operating system's files or programs loaded into memory are trusted by the administrator.

3.3 Blockchain

As the underlying technology of the Bitcoin cryptocurrency system [38,39], blockchain is actually a distributed digital ledger. Each node in the blockchain system maintains the same backup of the entire ledger, and in subsequent developments, some nodes have emerged to keep only partial ledger blockchains, such as Stefano's scheme [40]. Consistency of the digital ledgers on these nodes is guaranteed by consensus protocols [41,42] among the nodes. The implementation mechanism of the blockchain ensures that data cannot be changed once written to the blockchain.

The two most important parts of the blockchain are the block structure and the consensus algorithm. The block structure specifies how the data and records on the blockchain are organized

and stored, while the consensus algorithm specifies how data from different nodes are kept consistent. Generally speaking, a blockchain is a chain of blocks connected by a hashing algorithm. Each block contains a block header and a block body. The block header contains summary information such as the previous block's hash value, the Merkle tree root of the transaction record, and the timestamp. The block body contains all the transaction information, such as the subject and object of the transaction, input, output, etc., for a certain period.

4 Overview

In this section, we present the PIMS design architecture and design goals.

4.1 Design Goals

Design goals for a dynamic monitoring approach to process code segment integrity at container runtime:

(1) *Integrity*. The proposed approach should ensure tamper-proof of process measurement value and baseline value data.

(2) *Validity*. The proposed approach should ensure that the integrity of the process can be verified while the container is running.

(3) *Accuracy*. We have designed the system to ensure that files with the same name can be distinguished from different containers.

4.2 Assumptions

To focus more on the problem to be solved in this paper, we make the following assumptions:

(1) Cloud server with hardware TPM.

(2) Physical attack methods (e.g., hardware tampering) are outside the scope of the threats considered in this paper.

(3) The blockchain is a trusted component, and once enough nodes confirm a transaction, the data on the blockchain is trusted and tamper-proof.

4.3 Attacker Model

Adversary model: The attacker model in this paper is similar to the commonly used honest but curious model, where an attacker can compromise the integrity of containers in the system. All entities (i.e., users and service providers) are computationally bounded. However, the service provider can request additional resources from a powerful server or cloud computing. Therefore, this feature enables service providers to access unlimited resources compared to users. In addition, all entities are honest and perform their functions strictly as designed.

4.4 Architecture

In this paper, we have designed and implemented a process integrity monitoring method in container clouds. First, the monitoring system has given the ability to manipulate physical memory by loading virtual character devices in the Linux kernel. Thus, it provides process integrity measurement service for the cloud platform. Next, constructing a hardware-trusted foundation for process integrity relies on TPM 2.0. We stored measurement values in the TPM and protected the measurement results from tampering. In the end, integrity analysis is based on the remote attestation function of trusted

computing technology. The values recalculated from the measurement logs were compared with the baseline value library data stored on the blockchain to judge whether the integrity was broken.

Fig. 1 gives the general architecture of the Process Integrity Monitoring System (PIMS) for container runtime. It contains three main modules with the following functions:

(1) Dynamic measurement module (DMM). DMM is mainly composed of a process information collection submodule in user space and a measurement submodule in character device Dymeasure. The process information collection submodule is responsible for collecting the virtual address space mapping information of all processes in the monitored container. Then, via the Input/Output Control (IOCTL) system call, it transfers the information to the measurement submodule in the kernel space and initiates the measurement command. The measurement submodule is responsible for completing the translation from the process's virtual address to the physical address depending on the information sent down by the process information collection submodule. Subsequently, the actual physical address of each page of the process code segment is determined. Finally, the integrity measure of these code segment paging is performed by invoking the hashing algorithm provided by the kernel.

(2) Trusted foundation building module (TFBM). TFBM is responsible for extending the measurement results into the PCR of the TPM device based on obtaining the measurement values of the process code segment from the DMM. Meanwhile, the dynamic measurement trust chain of the monitoring system is constructed based on the TPM 2.0 hardware device. Finally, the corresponding measurement logs are generated for later integrity verification work.

(3) Integrity verification module (IVM). IVM has two functions. The first one is the collection function. When IVM receives a remote verification request, the agent deployed in the host is responsible for collecting the integrity information and sending it to the remote verifier. The second is the verification function, reproducing the dynamic measurement process through the collected integrity information. The recalculated hash value is first compared and analyzed with the baseline value stored on the blockchain, and then the integrity is judged to be broken based on the comparison result.

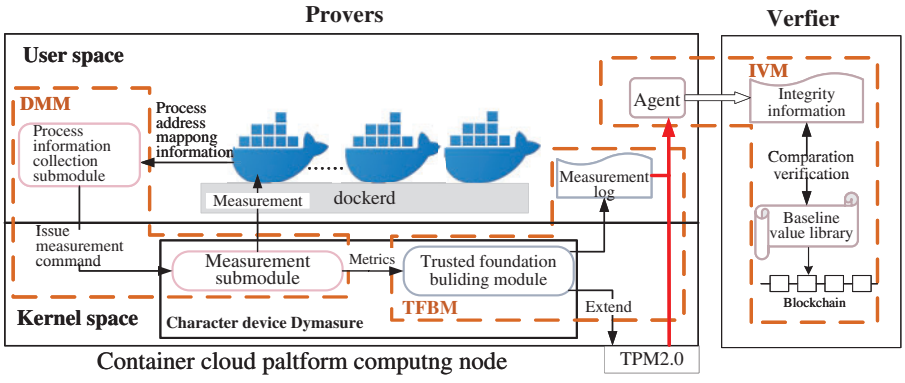

Figure 1: The architecture of PIMS

5 PIMS Design

In container cloud environments, applications of different users are deployed in containers. From the user's perspective, there is a need to ensure that the applications running in the containers are not maliciously corrupted. Therefore, we propose a dynamic monitoring method for the integrity of Docker's processes from the perspective of integrity monitoring, aiming at the integrity measurement of process code segments and the verification of the measurement results. The method consists of four processes: dynamic measurement at container runtime, construction of a trusted baseline value library, storage of measurement values and generation of measurement log, and integrity verification.

5.1 Dynamic Measurement for Container Runtime

The Dynamic Metrics Module (DMM) loads a virtual character device called Dymeasure in kernel space. The measurement submodule in Dymeasure implements a dynamic measurement for the integrity of process code segments in containers in user space. Indeed, the measurement submodule consists of two components: physical address resolution and pagination measurement, as shown in Fig. 2. The physical address resolution component accomplishes the resolution of the virtual memory address to the physical memory address mapping of the code segment; the pagination measurement component performs the measurement operation on individual pages of the code segment. The specific execution flow is as follows:

(1) The system administrator or cloud tenant adds the container IDs that need to be monitored to the.config configuration file via the Agent.

(2) Read the configuration file through the user space tool UserUtil, and check whether the container ID exists in the system. If not, return an error signal to the exception locator module; if the container ID is valid, jump to Step (3).

(3) UserUtil obtains the host process number PID of all processes in the container to be monitored (i.e., the process number identified by the container's host) and the image name of the container's image by interacting with the container daemon.

(4) UserUtil reads the/proc/[PID]/maps file based on the host pid of the process. Furthermore, the start virtual address vaddr_start and the end virtual address vaddr_end of the process code segment are resolved, as well as the path elf_path of the ELF file corresponding to the process.

(5) The process-related information obtained in the above steps is passed to Dymeasure via the IOCTL system call. Meanwhile, a measurement operation command is launched for each process in the container.

(6) The physical address resolution component determines whether the current process paging has been loaded into memory (in-ram) or swapped out (swapped) based on the process-related address information that has been obtained. If the paging is loaded into memory, translate the virtual memory address of the paging of the code segment using the mapping records in pagemap to get its physical memory addresses phyaddr_start and phyaddr_end, and jump to Step (7); otherwise, the current process paging is ignored. At the same time, a bitmap data structure, page_bitmap, is used to record the loading of process paging in physical memory. Among other things, a bitmap page_bitmap is used to facilitate subsequent verification of the integrity of the code segment. The bitmap records whether the current virtual memory page is loaded into memory, marking it as 1 if it is and 0 otherwise.

(7) The physical address resolution component passes the starting physical memory address phyaddr_start and ending physical memory address phyaddr_end of single paging of the code segment to the paging measurement component.

(8) The pagination measurement component invokes a set hash algorithm according to the starting physical address of the process pagination. When completing the integrity measurement for paging a code segment, the measurement value page_digest is generated. In addition, after each completed paging measurement, the hash of the current paging measurement is concatenated with the hash of the previous measurement, and the hash is calculated again, resulting in an aggregated measure of all process paging segments loaded into physical memory, digests_aggregate.

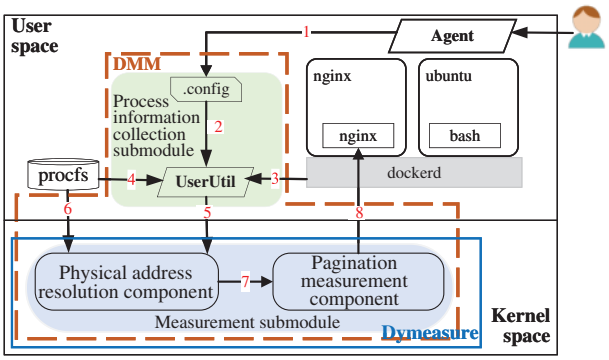

Figure 2: Paginated dynamic measurement for code segment of the process

5.2 Dynamic Measurement for Container Runtime

Because the physical memory data initialization of a process comes from its corresponding ELF file loaded into memory. Therefore, the paging measurement result of the process code segment is the same as the paging measurement result of the code segment in its corresponding ELF file. Consequently, we construct the baseline value library by performing the paged measurement on ELF code segments and obtaining the metric values for each of their paging. Each cloud server has one baseline value library. The construction method can be divided into three steps.

(1) *Obtaining the ELF file in the container*

The creation of containers is divided into two cases: manually building a container image by oneself and building an image based on a public image or someone else's image. If one builds the image manually, getting the ELF file in the container is relatively simple and performing measurement operations directly on the ELF file code segment. If it is built based on a public image or a third-party image, then we need to traverse the file system path of the base image, find the target ELF file and copy the target file from the container to the host or from the host to the container.

(2) *Paginated measurement ELF code segment*

Algorithm 1: Pagenation Measurement Algorithm for Code Segment of a ELF File

Input: elf_path, offset, code_seg_size
Output: elf_page_digests

(Continued)

Algorithm 1: (Continued)

```
struct page_digest{
    char digest [HASH_ALG_LEN];
}*elf_page_digests; // Define elf_page_digest, store Metric Value
struct crypto_shash *t fm // Synchronize hash algorithm and the arithmetic
struct shash_desc *shash = kmalloc()
*elf_page_digests = kmalloc (code_seg_size >> 12) // Allocate memory
pos = offset
el_fd = filp_open (elf_path)
struct page_digest * ret = elf_page_digests
t fm = crypto_alloc_shash (HASH_ALG_NAME)
shash–> t fm = t fm
while pos < (offset + code_seg_size) do
    kernel_read (elf_fd, buffer, PAGE_SIZE, &pos) // Read one page of code segment
    crypto_shash_digest (shash, buffer, PAGE_SIZE, elf_page_digests)
    elf_page_digests ++
end while
crypto_free_shash (t fm)
kfree (shash)
return ret
```

After the ELF file is obtained, the paging measurement algorithm is performed on the code segment of the ELF file, as shown in Algorithm 1. Firstly, the data structure page_digest is defined to hold the metric values, and the array elf_page_digests is used to hold all the paging metric values. Meanwhile, declare both the hash algorithm and the arithmetic. Next, open the binary file to be metricized, get the file descriptor, and initialize the hash algorithm. Then, each page of the ELF segment is measured, and the result is stored in the elf_page_digests pointer. Last, the occupied space is freed, and the first pointer to the array holding the paging metric value is returned. The final metric value for each paging of the ELF code segment was obtained and used as a record in the baseline value library.

(3) *Storage of baseline value library data*

The Consortium Blockchains features tamper-proof data on the chain, traceability of transactions, and member access mechanism. Therefore, the Consortium Blockchains can be an excellent solution to the lack of integrity protection for baseline value data. We connect all the cloud servers and a verification server to the blockchain network as user nodes of the blockchain, as shown in Fig. 3. The cloud server gets the metric values by executing Algorithm 1 on all ELFs in the container. Then, these metric values are used as the baseline value data and uploaded to the block. As a data user, the verification server has access to the baseline value data in the blocks of other user nodes. Meanwhile, the integrity of the baseline value data is protected by the consensus algorithm of the Consortium Blockchains.

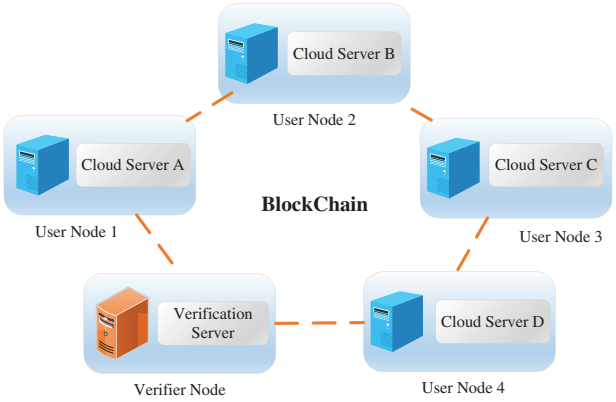

Figure 3: Storage of baseline value data

5.3 Storage of Metric Values and Generation of Measurement Log

5.3.1 Storage of Metric Values

The metric values need to be stored in tamper-proof hardware to improve the security of the PIMS system in this paper and to prevent tampering with the dynamic metric results. The storage of process integrity metric values depends on the measurement of the processes in Section 5.1. The DMM is required to provide the final aggregated metric value and information about the process. Meanwhile, the TFBM selects PCR No. 11 as the register for saving dynamic metric values to prevent the metric results from being tampered. The description of the storage of the metric value execution process is shown in Fig. 4.

(1) When inserting the virtual character device Dymeasure, Step 1a), TFBS reads the first record boot_aggregate of the IMA storage measurement log imaSML. Next, Step 1b), the template-hash in the boot_aggregate record is extended to PCR No. 11 as a base. Furthermore, in Step 1c), at first, the extended PCR 11 value pcr11_value is read; second, record the pcr11_value along with the boot_aggregate information into the integrity information log file of the process code segment; lastly, generate the first log record of the PIMS in the container. Eventually, credible evidence of the computer startup process was applied to PIMS.

(2) In the subsequent dynamic metric operation, DMM passes parameters to TFBS, including aggregated metric value digests_aggregate, code segment paging bitmap page_bitmap, container image name image_name, and the corresponding ELF binary file path elf_path of the measured process.

(3) Extend the aggregation metric digests_aggregate to PCR 11 and read the extended PCR11 value pcr11_value.

(4) Record pcr11_value, digests_aggregate, image_name, and elf_path to the integrity information log of the process code segment of storing measurement log idmsSML, and record the code segment paging bitmap page_bitmap to the process paging bitmap log file proc_pages_bitmap of idmsSML.

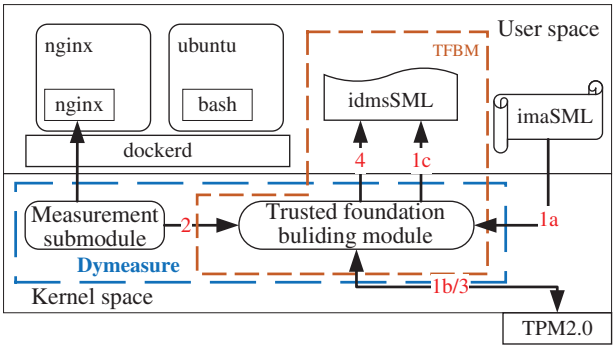

Figure 4: Process segment integrity trusted foundation construction

5.3.2 Generate Measurement Log

The measurement log needs to be generated to facilitate the reproduction of the measurement process in remote attestation and to verify the trustworthiness of the container and the processes running in it. Simultaneously, we have implemented a storage measurement log idmsSML format design for dynamic measurement, in order to distinguish the same name files in different containers in the measurement log. In this paper, idmsSML is divided into two categories: the integrity information log of the process code segment and the paging bitmap log, as shown in Fig. 5.

(1) *Process segment integrity information log*

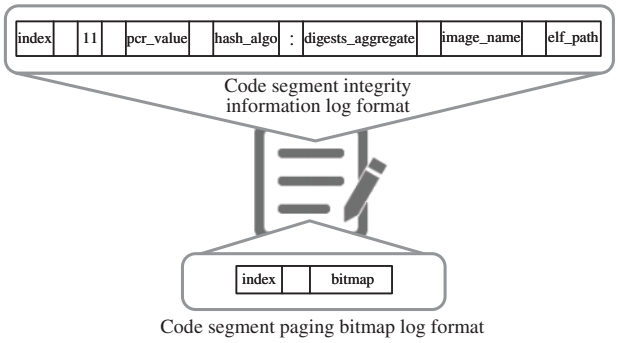

Figure 5: Store measurement log idmsSML

The integrity information log of the process segment contains two files: binary_proc_measurements and ascii_proc_measurements. Both of them record the results and related information of dynamic measurement. The former directly stores the metric value in binary format. The latter converts the measurement result, stores it in ASCII encoding, and is readable. The recorded information is shown in Fig. 5, where each field represents the meaning of each as follows:

- index: Log index, starting from 0. The first record is the integrity information of the underlying computer environment, from the template-hash field of the boot_aggregate record of imaSML. Starting from 1 is the official record of the dynamic measurement of the process code segment.

- 11: PCR index number for storing integrity information.
- pcr_value: The value of PCR11 after performing the extended operation.
- hash_algo: The name of the hashing algorithm used by the measurement.
- digests_aggregate: The aggregate metric value of the process segment paging metric values from a single dynamic measurement.
- image_name: The name of the image of the container to which the process belongs.
- elf_path: The path of the corresponding ELF file in the container of the process.

(2) *Paging bitmap log of process code segment*

The PIMS uses a paging measurement for process code segments due to the characteristic of computer memory management. Since the integrity information is saved, we have selected its aggregated metric value for saving. Therefore, it makes the verification of integrity difficult. For this reason, we have designed and implemented a paging bitmap log of process code segments to hold the paging measurement process. Also, it can distinguish whether the paging is measured or not. The format of the paging bitmap log for the code segment is shown in Fig. 5. The paging bitmap log of the process code segment is saved as a file proc_pages_bitmap with the field meanings described as follows:

- index: Index of the log. Each record, starting from 1, corresponds to a record with the same index in the process integrity information log.
- bitmap: Paging bitmap. The page_bitmap generated in Section 5.1 is filled in to represent the paging of the process code segment in the current dynamic measurement.

5.4 Integrity Verification

The Agent collects the integrity information generated by TFBM and sends it to the verifier. The verifier reproduces the measurement process based on the process paging measurement log, and compares it with the baseline value library information stored on the blockchain to verify whether the process integrity is broken. The remote attestation process is shown in Fig. 6.

(1) First, find the paging baseline[i] of the corresponding code segment based on the image_name and elf_path information of the ith record in the ascii_proc_measurements file.

(2) Second, after returning the corresponding baseline[i], find the corresponding paging bitmap record pages_bitmap[i] based on the index.

(3) Then, combine baseline[i] and pages_bitmap[i] to calculate the digests_aggregate[i] corresponding to the ith record.

(4) Next, hash the obtained digests_aggregate all in sequence to get the aggregated value pcr11_value_compare.

(5) Finally, the PCR11_Value obtained by the TPM_Quote command is compared with the pcr11_value_compare to verify that the integrity is not broken.

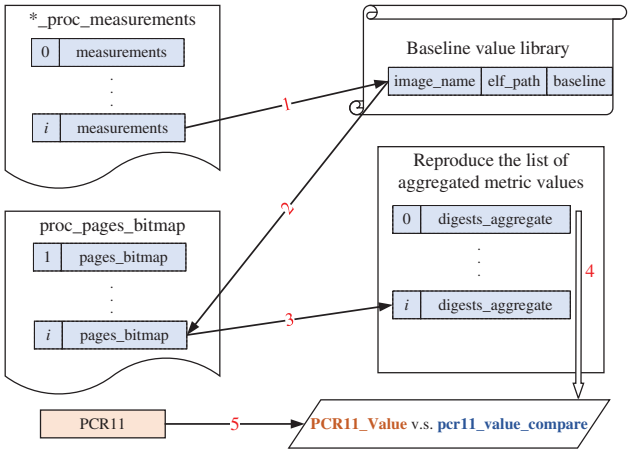

Figure 6: Integrity information verification process for PIMS

6 Evaluation

In this section, a baseline value integrity analysis and a series of experiments are provided to evaluate the performance of the proposed PIMS in a container environment.

6.1 Integrity Analysis of Baseline Values

The baseline value library data is stored explicitly on the validation server or downloaded locally and protected by a local protection mechanism. Therefore, the integrity of the baseline values relies on strong security assumptions. In the PIMS design, we implement integrity protection of baseline value data through blockchain.

Blockchain technology leverages the hash function's collision resistance to ensure the data's immutability. Hash functions satisfy three properties:

(1) *Unidirection ality*. For a given y, it is computationally infeasible to find an that can make $H(x) = y$ hold.

(2) *Weak collision resistance*. For a given y, it is computationally infeasible to find another x' that can make $H(x) = H(x')$ hold.

(3) *Strong collisionality*. It is computationally infeasible to find dissimilar x and x' that can make $H(x) = H(x')$ hold.

In the blockchain, each data block is a transaction, and the resulting Merkle tree root value is stored in the block header. Merkle tree is a binary tree structure built using a hash function [43]. Its leaf nodes at the lowest level are data blocks, and the content (label value) of each non-leaf node is the hash value of its children nodes when concatenated. Following this approach, a Merkle tree root is obtained, as shown in Fig. 7. Let us analyze the integrity protection of baseline values with a simple example.

Assume that the letters A, B, C, and D in Fig. 7 represent the leaf nodes, respectively, indicating different baseline values. The initialization phase starts generating a Merkle tree by hashing algorithm and returns the Merkle tree root value to the challenger. In the execution phase, the challenger randomly selects nodes in the Merkle tree (e.g., K) to challenge the attacker. The attacker responds to the challenger with the label value of node K and the hash value of the root path of the Merkle

tree generated by that node. The challenger calculates the Merkle root based on the value returned by the attacker. The baseline value integrity is not broken if it is the same as the Merkle root in the initialization phase. If it is not the same, according to the three properties of the hash function, it means that the integrity of the baseline value is broken.

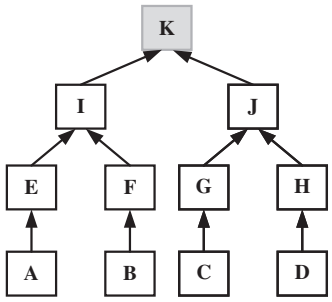

Figure 7: Merkle tree

In a nutshell, the integrity of the baseline value library data can be well protected by storing it on the blockchain.

6.2 Experiment

This section analyzes PIMS in terms of functionality and performance, respectively. PIMS prototype system developed on the Ubuntu 20.04 operating system, and by loading a virtual character device, named Dymeasure, in kernel space to achieve paging dynamic measurement of process code segments within docker in user space. The configuration information of the test environment of the prototype system is shown in Table 1. One of the servers has no trusted chip, while the others have the same configuration and all have TPM 2.0 trusted chips.

Table 1: Experimental environment configuration

OS	Kernel	CPU	Memory	Disk	TPM
Ubuntu 20.04	5.4.9	Intel core (TM) i3-4150	16 GB	1 TB	None
Ubuntu 20.04	5.4.9	Intel core (TM) i7-9700	16 GB	1 TB	TPM 2.0

6.2.1 Functional Analysis

We designed an experiment on malicious tampering with container processes to demonstrate that PIMS can accurately detect integrity tampering problems while the container is running. Four containers were run on the cloud platform in the experimental setup, and different web applications were run in each container separately. Users access the web application through the open port of the container, and administrators can log in to the container through the docker exec command to perform related management operations. The attacker tries to attack the web service of one of the containers and run a tampered bash process in it, and the experimental results are shown in Fig. 8.

In Fig. 8, we can see that three processes are running in this container, among which the python process and a bash process have the same measurement value as the initial measurement. However, the other bash process does not match the initial measure value of the bash progress measure. Therefore,

it can be determined that the container has been tampered with, and an unfamiliar process has been started.

(a) Process baseline value with container ID 3b819373754d

(b) Discover abnormal processes

Figure 8: Measurement at container runtime

6.2.2 Performance Analysis

We have used the UnixBench performance testing tool to perform performance loss tests before and after the deployment of PIMS. Ten container instances are run for the test, and the attacker is assumed to tamper with one of the container processes every 20 s. Then, PIMS performs an integrity measurement operation on the process code segment of that container. Finally, the integrity of the process is compared and analyzed by remote attestation. The performance test results are shown in Fig. 9.

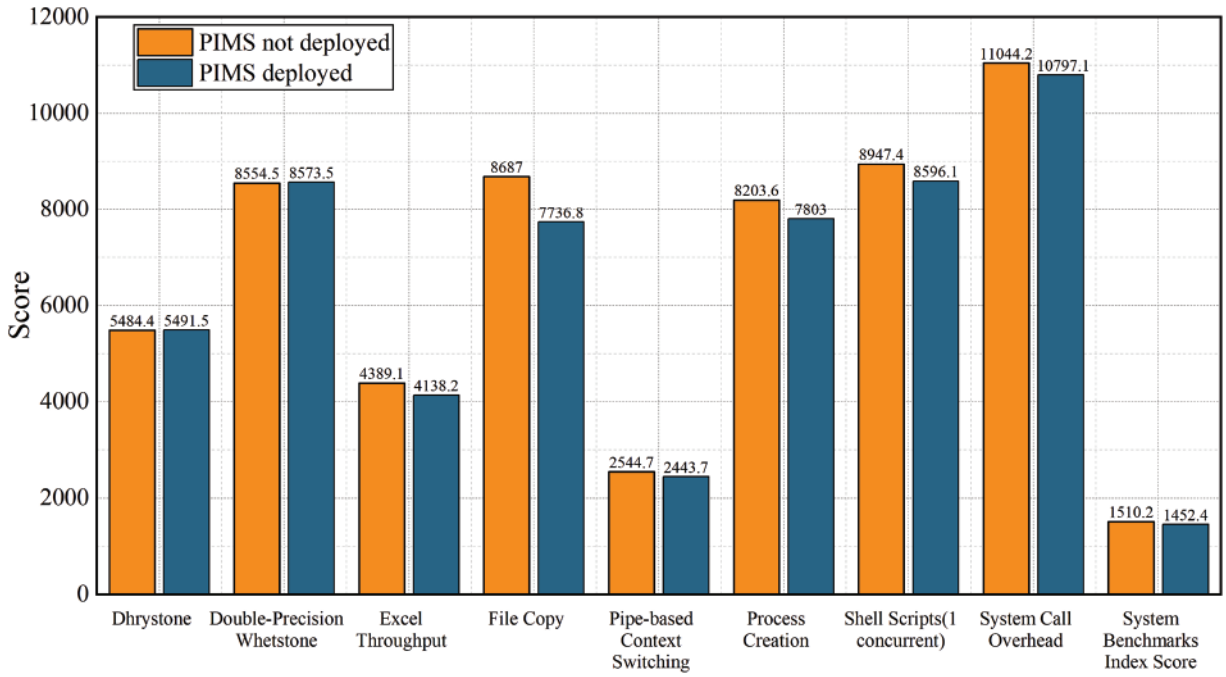

Figure 9: Comparison of system performance overhead

From Table 2, we can see that by comparing the two test environments, the performance loss caused by PIMS to the container is only 3.57%, which is less than 5% and within the acceptable range.

The results in Fig. 9 show that PIMS only has a relatively large impact on the File Copy subscore. By analysis, the reason for the high-performance loss in this sub-item is that the PIMS system performs paged measurement operations on process code segments. Moreover, the measurement results are extended into PCR11 after each measure, and the extended results are subsequently read. Therefore, many files read/write and copy operations are generated during this process, resulting in an impact on File Copy.

Table 2: Overall performance evaluation

PIMS not deployed	PIMS deployed	Average performance loss
1539.5	1484.6	3.57%

7 Conclusion

In this paper, we propose an integrity monitoring method for processes of containers from the perspective of process integrity monitoring. The proposed approach: (1) A dynamic measurement method for paging process code segments in containers has been implemented based on the process runtime information provided by the procfs file system. (2) Build a trusted foundation for process integrity with the trusted chain technology of trusted computing and the tamper-proof features of the hardware TPM 2.0 module. (3) A baseline value library for paging code segments is created using the ELF files corresponding to the processes and stored on the blockchain to provide easy access to baseline data while maintaining integrity. (4) It also enables recoverable traceability of metric results in the integrity verification phase. Simulation experiments evaluate the performance of the proposed system. The results show that the proposed approach can accurately verify the trusted state of all containers in the cloud environment.

Funding Statement: The work was supported by China's National Natural Science Foundation (U19A2081; 61802270; 61802271); Ministry of Education and China Mobile Research Fund Project (MCM20200102; CM20200409); Sichuan University Engineering Characteristic Team Project 2020SCUNG129.

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

References

1. Chae, M., Lee, H., Lee, K. (2019). A performance comparison of linux containers and virtual machines using docker and KVM. *Cluster Computing, 22(1),* 1765–1775. DOI 10.1007/s10586-017-1511-2.
2. Srivastava, P., Khan, R. (2018). A review paper on cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering, 8(6),* 17–20. DOI 10.23956/ijarcsse.v8i6.711.
3. He, X., Tian, J., Liu, F., Management, S. O., University, H. (2019). Survey on trusted cloud platform technology. *Journal on Communications, 40(2),* 154–163.
4. Zhang, X., Yang, J., Sun, X., Wu, J. (2018). Survey of geo-distributed cloud research progress. *Ruan Jian Xue Bao/Journal of Software, 29(7),* 2116–2132.
5. Yu, J., Chuan, J., Dong, P. (2019). A review of docker security research. *Computer Science and Application, 9(5),* 926–933.

6. de Benedictis, M., Lioy, A. (2019). Integrity verification of docker containers for a lightweight cloud environment. *Future Generation Computer Systems, 97,* 236–246. DOI 10.1016/j.future.2019.02.026.
7. Rad, B. B., Bhatti, H. J., Ahmadi, M. (2017). An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS), 17(3),* 228.
8. Compastié, M., Badonnel, R., Festor, O., He, R. (2020). From virtualization security issues to cloud protection opportunities: An in-depth analysis of system virtualization models. *Computers & Security, 97,* 101905. DOI 10.1016/j.cose.2020.101905.
9. Liu, C., Lin, J., Tang, B. (2013). A dynamic trustworthiness verification mechanism for trusted cloud execution environment. *Journal of Software, 24(1),* 1240–1252.
10. Shringarputale, S., McDaniel, P., Butler, K., La Porta, T. (2020). Co-residency attacks on containers are real. *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pp. 53–66. New York.
11. Dirty, C. (2021). Cve-2016-5195. https://dirtycow.ninja/.
12. Casalicchio, E., Iannucci, S. (2020). The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience, 32(17),* e5668. DOI 10.1002/cpe.5668.
13. Demigha, O., Larguet, R. (2021). Hardware-based solutions for trusted cloud computing. *Computers & Security, 103,* 102117. DOI 10.1016/j.cose.2020.102117.
14. TCG (2021). Trusted platform module library, part 1: Architecture.
15. Tian, D., Choi, J. I., Hernandez, G., Traynor, P., Butler, K. R. (2019). A practical intel sgx setting for linux containers in the cloud. *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pp. 255–266. New York.
16. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L. (2004). Design and implementation of a tcg-based integrity measurement architecture. *USENIX Security Symposium*, vol. 13, pp. 223–238. San Diego.
17. Wang, Q. X., Chen, X. S., Jin, X., Li, X., Chen, D. J. et al. (2021). Enhancing trustworthiness of Internet of Vehicles in space–air–ground-integrated networks: Attestation approach. *IEEE Internet of Things Journal, 9(8),* 5992–6002. DOI 10.1109/JIOT.2021.3084449.
18. Cheng, J., Zhang, K., Tu, B. (2021). Remote attestation of large-scale virtual machines in the cloud data center. *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 180–187. Shenyang, China.
19. Wang, W., Chen, X. S., Lan, X., Jin, X. (2018). VMI-based virtual machine remote attestation scheme. *Chinese Journal of Network and Information Security, 4(12),* 32.
20. Shen, X., Jiang, S., Zhang, L. (2021). Mining bytecode features of smart contracts to detect ponzi scheme on blockchain. *Computer Modeling in Engineering & Sciences, 127(3),* 1069–1085. DOI 10.32604/cmes.2021.015736.
21. Chen, J., Zhang, C., Yan, Y., Liu, Y. (2022). Filewallet: A file management system based on ipfs and hyperledger fabric. *Computer Modeling in Engineering & Sciences, 130(2),* 949–966. DOI 10.32604/cmes.2022.017516.
22. Yang, J. C., Wen, J., Jiang, B., Wang, H. (2020). Blockchain-based sharing and tamper-proof framework of big data networking. *IEEE Network, 34(4),* 62–67. DOI 10.1109/MNET.65.
23. Hosseinzadeh, S., Laurén, S., Leppänen, V. (2016). Security in container-based virtualization through vtpm. *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pp. 214–219. Shanghai, China.
24. Guo, Y., Yu, A., Gong, X., Zhao, L., Cai, L. et al. (2019). Building trust in container environment. *2019 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)*, pp. 1–9. Rotorua, New Zealand.

25. Docker, I. (2021). Docker storage drivers.
26. Liu, Z. W., Feng, D. G. (2010). Tpm-based dynamic integrity measurement architecture. *Journal of Electronics & Information Technology, 32(4),* 875–879. DOI 10.3724/SP.J.1146.2009.00408.
27. Pan, G. (2020). *Research on docker platform protection technology based on trusted computing (Master's Thesis)*. Southeast University, China.
28. Jin, X., Wang, Q. X., Li, X., Chen, X. S., Wang, W. (2019). Cloud virtual machine lifecycle security framework based on trusted computing. *Tsinghua Science and Technology, 24(5),* 520–534. DOI 10.1109/TST.5971803.
29. Chen, D., Wang, H., Zhang, N., Nie, X., Dai, H. N. et al. (2022). Privacy-preserving encrypted traffic inspection with symmetric cryptographic techniques in IoT. *IEEE Internet of Things Journal, 9(18),* 17265–17279. DOI 10.1109/JIOT.2022.3155355.
30. Chen, D., Zhang, N., Wu, H., Zhang, K., Lu, R. et al. (2022). Audio-based security techniques for secure device-to-device (D2D) communications. *IEEE Network,* 1–8. DOI 10.1109/MNET.005.2100336.
31. Han, Y., Ji, T., Wang, Z., Liu, H., Jiang, H. et al. (2021). An adversarial smart contract honeypot in ethereum. *Computer Modeling in Engineering & Sciences, 128(1),* 247–267. DOI 10.32604/cmes.2021.015809.
32. Shao, Q. F., Jin, C. Q., Zhang, Z., Qian, W., Zhou, A. Y. (2018). Blockchain: Architecture and research progress. *Chinese Journal of Computers, 41(5),* 969–988.
33. Ritzdorf, H., Wüst, K., Gervais, A., Felley, G., Capkun, S. (2018). TLS-N: Non-repudiation over TLS enabling-ubiquitous content signing for disintermediation. *Network and Distributed Systems Security (NDSS) Symposium 2018*, pp. 18–21. San Diego.
34. Pavithran, D., Al-Karaki, J. N., Shaalan, K. (2021). Edge-based blockchain architecture for event-driven IoT using hierarchical identity based encryption. *Information Processing & Management, 58(3),* 102528. DOI 10.1016/j.ipm.2021.102528.
35. Huang, C. L., Chen, W., Yuan, L., Ding, Y., Jian, S. et al. (2021). Toward security as a service: A trusted cloud service architecture with policy customization. *Journal of Parallel and Distributed Computing, 149,* 76–88. DOI 10.1016/j.jpdc.2020.11.002.
36. Shaw, A. L., Attak, H. (2017). Exploring granular flow integrity for interconnected trusted platforms. *2017 IEEE Trustcom/BigDataSE/ICESS*, pp. 594–601. Sydney, Australia, IEEE.
37. TCG (2018). TCG guidance for securing network equipment using TCG technology version 1.0 revision 29.
38. Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review,* 21260. DOI 10.2139/ssrn.3440802.
39. Han, D., Chen, J., Zhang, L., Shen, Y., Gao, Y. et al. (2021). A deletable and modifiable blockchain scheme based on record verification trees and the multisignature mechanism. *Computer Modeling in Engineering & Sciences, 128(1),* 223–245. DOI 10.32604/cmes.2021.016000.
40. Chen, H. F., Wu, H. L., Chang, C. C., Chen, L. S. (2019). Light repository blockchain system with multisecret sharing for industrial big data. *Security and Communication Networks, 2019*. DOI 10.1155/2019/9060756.
41. Angelis, S. D., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A. et al. (2018). pbft vs proof-of-authority: Applying the cap theorem to permissioned blockchain. *Italian Conference on Cyber Security*, Milan, Italy.
42. Ma, C., Zhang, Y., Fang, B., Zhang, H., Jin, Y. et al. (2022). Ripple plus: An improved scheme of ripple consensus protocol in deployability, liveness and timing assumption. *Computer Modeling in Engineering & Sciences, 130(1),* 463–481. DOI 10.32604/cmes.2022.016838.
43. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K. (2015). Proofs of space. *Annual Cryptology Conference*, pp. 585–605. Berlin, Heidelberg, Springer.