Implementation of multi-domain isolation architecture and communication mechanism in Linux

Yuqing Lan*, Jianlun Zou#
School of Computer Science and Engineering, Beihang University, Beijing 100191, China

ABSTRACT

Multiple Independent Levels of Security (MILS) is widely used in the design of high assurance operating system. By separating the system into components, and making the components run in different domains, the kernel can control and monitor information flow between components to enhance the security and availability of system. However, due to the complexity and certification cost issue associate with large monolithic kernel, MILS architecture is mainly used in microkernel system. But we still want to use the idea of MILS in monolithic kernel system to improve the security. In the Linux, although there are some access control models based on the concept of domain (like SELinux). Limited by the feature of shared kernel, the security of system is affected by the vulnerabilities in itself. Therefore, this paper proposes a scheme of constructing multiple independent isolated domains based on virtualization technology in Linux. We developed on Linux kernel and QEMU/KVM hypervisor, exploiting the isolation feature brought by virtualization to achieve data isolation. We build domain from virtual machine, so that we can separate origin system into components and run them in domains. In the host, we take control of all domains and implements a secure communication mechanism between domains. By using this secure channel, we can monitor the data transmission between domains, and control the information flow according to the security level of the domain. Finally, we evaluated the effectiveness and efficiency of our communication mechanism.

Keywords: Multi-domain isolation, virtualization, communication mechanism.

1. INTRODUCTION

With the progressing of information technology, various information systems are becoming larger and more complex, which brings growing threats to the safety. Therefore, either individuals or enterprise raise higher demand on information systems and operating systems.

These threats can be generally divided into two categories: external threats and insider threats. External threats come from the malware and malicious attacks on Internet, and insider threats is derived from the vulnerabilities of the kernel or user programs. According to the statics from National Internet Emergency Centre (CNCERT/CC), in the first half year of 2021, about 23.07 million samples of malicious programs were captured, with average transmitting times reaching 5.82 million every day, involving 208 thousand malicious program families. The general security vulnerabilities recorded by China National Vulnerability Database (CNVD) totalled 13083, including 3719 high-risk vulnerabilities and 7107 "zero-day" vulnerabilities. The top three vulnerabilities were application program vulnerabilities, web application vulnerabilities and operating system vulnerabilities.

Nowadays, there are many products for enhancing the security of information systems, providing basic security guarantee from permission control to data monitoring. For example, the general-purpose operating system Linux adopted discretionary access control (DAC), mandatory access control (MAC) and role-based access control (RBAC) to ensure the security of system.

The discretionary access control allows users to pass their permission to others. The well-known scheme for DAC is access matrix[1], which is often implemented as a sparse matrix. In order to improve efficiency, Linux implements DAC using access control list (ACL) and capabilities. Compared to ACL, capabilities can grant authorization at a finer level.

* lanyuqing@buaa.edu.cn

zjl_2006209@buaa.edu.cn

Third International Conference on Computer Science and Communication Technology (ICCSCT 2022)
edited by Yingfa Lu, Changbo Cheng, Proc. of SPIE Vol. 12506, 1250610
© 2022 SPIE · 0277-786X · doi: 10.1117/12.2661795

Contrary to the DAC, the permission cannot be granted by user in mandatory access control (MAC). It uses the policies configured by manager to restrict user's access behaviours. The formal model proposed by LaPadula et al.[2] ensures multi-level security and data privacy by establishing two rules: only allowing "write up" and "read down". Not long after, the Biba security model[3] is proposed for protecting data integrity, which introduces integrity levels similar to the LaPadula model and only allows information flows from low level to high level. However, these security models lack consideration for multi-user scenarios, and McLean[4] showed how to remedy the limitations of BLP model.

The user in DAC and MAC cannot reflect the real organization of staff, which brings great inconvenience to personnel management. Therefore, the model of role-based access control[5] (RBAC) was proposed to solve this problem. The conventional RBAC model can reflect the structure of personnel organization, but it uses static permission which is difficult to satisfy the situation with frequently changed permission. Because of that, some studies introduced time[6], location[7] and priority[8] to RBAC model, expanding its applicable scenarios.

Although these access control mechanisms perform well in protecting malware and viruses, they still cannot eliminate the risks caused by system vulnerabilities and covert channel[9-12]. For example, the "WannaCry" exploits the vulnerabilities of file sharing in operating system to modify user's private files and spread in LANs. Covert channel is a communication channel that can be used to transfer data bypassing the prohibition of the access control policies. The previous studies have proposed some methods for identifying covert channels[13-15].

As the MILS architecture coming into sight, the concept of "separation" was introduced into the design of operating system[16]. By separating the entire system into components and constructing multiple independent isolated domains, components can run in domain without affecting each other. The kernel provides communication channels between domains, and it can control the information flows according to the privacy level protecting the data from leakage.

This paper refers to the idea of separation in MILS. By dividing the system into multiple domains, we enhance the security of general-purpose operating system. We developed on Linux kernel and QEMU/KVM hypervisor, and build the domain exploiting the isolation feature of virtualization. We developed a communication channel in kernel and hypervisor, and implement a control program to mediate the data transmission between domains, enforcing surveillance and inspections.

In summary, the contribution of this paper includes:

- An approach for building a multi-domain isolation system based on virtualization technology.
- A mechanism for transferring data from inner-domain to host environment.
- A secure mechanism for exchanging data between domains in the host kernel.
- The evaluation of the effectiveness and efficiency of our communication channel.

This paper is organized as follow. Section 2 illustrates the design of system architecture. Section 3 elaborates the implementation in detail. Section 4 is the evaluation. Section 5 concludes our work.

2. DESIGN

2.1 Principle

2.1.1 Data Isolation of Domain. The data exchange between different domains should not occur in an unpermitted way, it can only be taken in a controlled way by using the communication mechanism implemented by us. As considering the subject in two different domains, they should not communicate with each other by using conventional IPC, shared file or network.

2.1.2 Communication Mechanism Between Domains. The communication mechanism implemented by us should be the only way of data exchange. This communication mechanism supports the controller's mediation, so that the behavior of data transmission can be executing under the restriction of access control policies. In order to undertake different tasks in different scenarios, this communication mechanism should support multiple data formats and various transmission functions.

2.2 Architecture

The first thing we should consider about is the isolation capability of the domain. However, due to the complexity of operating system, it is almost impossible to enumerate all communication methods in system. Therefore, we could only assume that an isolation technology is reliable, like many existing research do, and proceed with our research based on it.

This paper assumed that by disabling the network service QEMU/KVM virtual machines are completely isolated with each other in the aspect of processor, memory and storage. On the basis of this assumption, we construct multiple domains on host and run a virtual machine in each of that. We also founded a control domain in the user space of host environment. And the entire architecture of our system is shown as Figure 1.

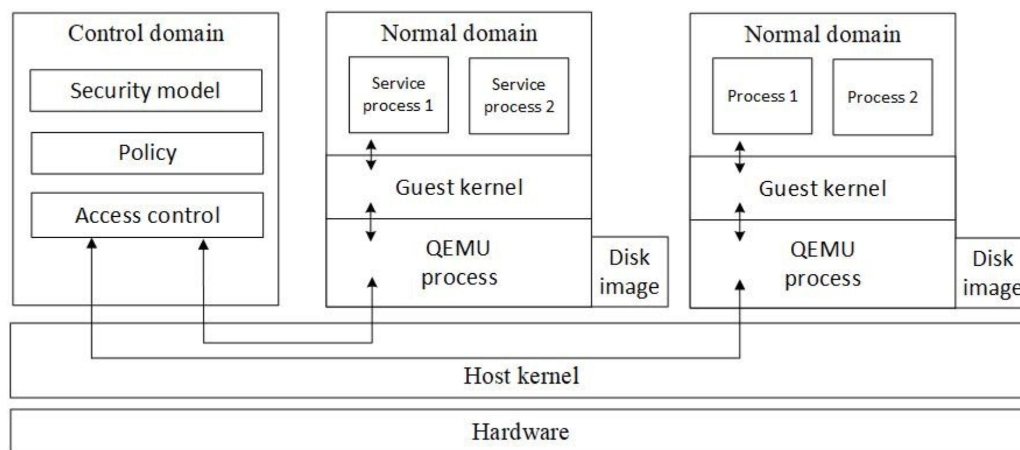

Figure 1. The Architecture of multi-domain operating system.

We developed on Linux kernel and implement domain concept in it. We define two kinds of domains: Normal domain and Control domain. The control program running in host user space will be associated to the control domain, and subsequently the QEMU process of each virtual machine will be attached to a normal domain. By setting up the concept of domain, the host kernel can manage the interdomain communication. All data transmission between Normal domains will be routed into control domain and enforce data inspection and access control policies.

The works we have done including:

- A kernel module with the function of data routing based on the concept of domain.
- A virtual device in QEMU and a frontend driver corresponding to it in guest kernel, which is used to deliver the data to the host.
- A group of control program, which is used to manage domains and inspect data flow.

3. IMPLEMENTATION

Our work has four parts, they are located in guest kernel, host kernel, QEMU user space program and host user space program respectively. The software version our project based on is Linux kernel 4.18, QEMU 3.1.0. The following sections describe our works in detail.

3.1 Virtual device in QEMU

QEMU need to simulate various external devices, it uses QEMU Object Model (QOM) to organize and manage the device simulation code. Therefore, we can define our own buses and devices in QEMU, to enable the capabilities of domain management and data transmission. By implementing these virtual devices, we can restrict data flowing out from domain only through it, so that we can achieve the goal of controlling and monitoring information flows.

First of all, we implement a PCI bridge device, inheriting the PCI bus class. This device has the capabilities of supporting others PCI device creating on it. In the run time, each domain runs a QEMU process and each QEMU process

has a PCI bridge. This PCI bridge device will specify domain through boot parameters, and get the information of that domain from host kernel at boot time.

In addition to the PCI bridge device, we also implement a special PCI device to support actual communication functions. By simulating the PCI device in QEMU, we can transmit data to other domains from a domain. At the frontend, the guest kernel within domain need to communicate with the QEMU process out of domain, and there are two types of communication: signal and block data transmission. The signal is bidirectional including inner domain to QEMU and QEMU to domain, which is respectively implemented by Memory mapped IO (MMIO) and device interrupt. As for the block data transmission, with the capabilities of managing and accessing the physical memory of guest virtual machine, QEMU can use shared memory to communicate with guest kernel.

Firstly, how domain communications with QEMU through the virtual PCI device? The PCI device support Memory mapped IO (MMIO). By using the MMIO, the processor can notify the PCI device simply by writing to the memory. Therefore, we use the MMIO to pass the parameters of signals. As shown in Figure 2, in order to support multiple functions in multiple scenarios, the configuration space of PCI device MMIO is divided into multiple fields align at 32 bits. These fields are divided into common configuration space and private configuration space, where the private configuration space refers to those fields specific to transmission function.

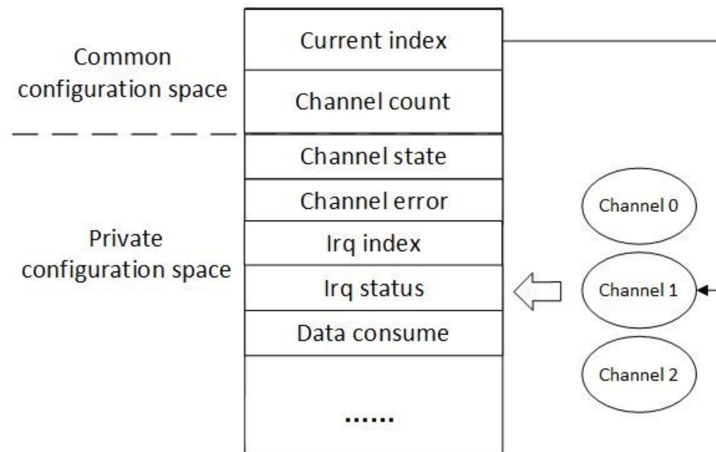

Figure 2. The fields of PCI device MMIO.

The guest kernel needs to write the configuration space twice to notify the QEMU. It will write the common configuration space first to choose a channel to activate, and write the private configuration space to tell which event is occurred. Using data transmission event as example, the guest kernel will write Data consume field to tell QEMU it wants to transfer data to the outside, and QEMU will then handle the data in shared memory. When the QEMU need to notify domain after processing data, it will fill the IRQ index field with channel number and IRQ status field with event number, and inject interrupt into virtual machine. The interrupt handler in guest kernel will read the value in these fields and trigger corresponding processing.

Secondly, how data is transferred from QEMU to host environment? This process is consisted of two steps: domain to QEMU and QEMU to host kernel. We use shared memory to support data transmission for both of them, but their implementation is different. The architecture is illustrated as Figure 3.

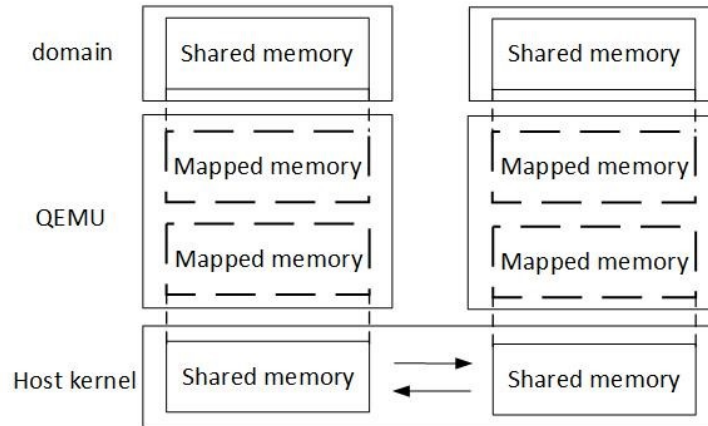

Figure 3. Shared memory model.

The shared memory from domain to QEMU is allocated continuously by guest kernel. We define two fields in the configuration space: the base address of shared memory and the length of shared memory. Guest kernel will convert virtual memory to physical memory after allocating it, and write the base and length of this physical memory into the configuration space. QEMU then translates the physical address in domain into the virtual address of QEMU process through the GPA to HVA mapping. Finally, QEMU checks the attributes of shared memory to ensure it is continuous and can be used normally.

The shared memory from QEMU to host kernel is allocated by host kernel. Similarly, the host kernel allocates continuous memory as well, and maps this area to QEMU process space by *mmap* system call.

However, these two shared-memory are pre-allocated in the initial stage of partition and channel set up, so their length will not change in the whole life cycle of partition, which is not a good way to transmit data with unknown length. Therefore, we use a ring queue to manage the memory space which is actually used for storing data. We only build the ring queue and descriptor array on the shared memory, and allocate memory dynamically to store the data. And we use descriptor to record the location and length of the memory. The following elaborates the design of ring queue in shared memory. We have three structures in shared memory.

- Descriptor table This is an array of descriptor struct. Each descriptor has four fields: address, the base address of memory segment; length, the length of memory segment; next, the index of next descriptor in array; flags, used to record this memory is the kind of writing in or reading out and whether it has successor.
- Available ring This is a circular linked list where each element points to an element of descriptor table. When adding memory segment, guest kernel will increase the element of this list. In contrast, QEMU will decrease the list after handling the data. It will use two pointers: next index points to the next empty index to be add in available ring; last index points to the last added element in available ring.
- Used ring Contrary to the Available ring, this list is increased by QEMU, and decreased by guest kernel. QEMU will remove the descriptor pointed by available ring and add it to used ring after handling it. There are also two pointers in this list, respectively pointing to next position and last position.

In summary, we have implemented a virtual PCI device in QEMU. Signals between guest kernel and QEMU is transferred by reading and writing MMIO configuration space and device interrupts. The actual data is delivered through shared memory, and the dynamically allocated memory is managed by ring queue data structure.

3.2 Front-end driver in guest kernel

3.2.1 The Organization of Frontend Drivers. The frontend drivers mainly consist of the following three parts: The PCI bus driver, which is used to handle the matching of guest drivers and data channel device; The PCI device driver, which is used to drive the PCI device and implement the common part of communication; A set of drivers specific to the transmission function, each kind of channel need a corresponding driver to implement the special part of communication.

In this paper, the kind of channel is identified by a 32 bits function value. Each function needs a private driver, we use an array to manage them and each driver need to be register in module initial stage.

3.2.2 The Atomicity of Reading or Writing Configuration Space. As described before, to ensure the scalability for new channel type, we provide same configuration space for different channels, resulting in two steps for each accessing configuration space behavior: write the current index field to switch channels, read or write the private configuration space. To ensure the atomicity for these two steps, we implement the accessing behavior as followed: acquire the spinlock to inhibit interrupts; write channel number to current index to activate the channel; access the target private configuration space; release the spinlock to resume the interrupts.

3.2.3 The Initialization Process of Drivers. At the initialization stage, the common PCI driver will initialize each channel by using the configurations gotten from configuration space. Each channel type has its own dedicated driver, and common driver will match them. At last, common driver requests an interrupt handler, which will invoke special interrupt handler implemented by dedicated driver according to the value of IRQ index field in configuration space. The dedicated drivers specific to channel type have these fields: type, a 32 bits function number to identify channel type; irq_handler, a call back define the special subprocess in interrupt context; probe, a call back define the special initial process of a channel type.

3.2.4 The Process of Data Transmission. The process of data transmission is synchronous, either write or read will block until getting the response from other side. We describe a complete transmission process as followed, using the write process as example: Firstly, when somebody request a transmission in domain, guest kernel will allocate a memory dynamically for storing data; Secondly, guest kernel will add the memory's physical address into descriptor table, and update available ring, and register a call back to an array at the same index of descriptor table; guest signal the QEMU by write the configuration space, and invoke wait to block itself; QEMU gets the memory from where the available ring point at and maps the GPA to HVA; QEMU adds the mapped host virtual address (HVA) to host shared memory, and notify host kernel; After host kernel handling data and return to QEMU, QEMU updates the guest used ring, fills the IRQ index and IRQ status field, and injects interrupt into virtual machine; At last, the interrupt handler in guest kernel calls the previously registered call backs to free the memory and wakeup the process blocked before.

3.3 Secure communication mechanism in host kernel

The initializing process of domain and channels in host kernel. In host kernel, we also create the concepts of domain and channel. Each domain has multiple types of channels, which is represented by the one-to-many relationship between domain and channel objects. At the beginning, we run a *startup* program to boot the system from a configuration file. This program will read domain configurations and create a control domain and multiple normal domains. Then, it will start a virtual machine for each domain and attach the QEMU process to the host kernel's domain object. Each QEMU user space process has three associations with kernel domain object: file descriptors, each domain and each channel in domain will create a file descriptor in QEMU process; a shared memory, a virtual memory allocated by host kernel will be mapped to QEMU process space using *mmap* system call; *eventfd*, an *eventfd* will be created by QEMU and set into host kernel, which is used for synchronization between QEMU and host kernel. When the attaching is start, these three initializing works are performed sequentially. And at the mapping stage, ring queue structures which is similar to what in guest kernel's shared memory will be built in host shared memory, acting as a mediation between user and kernel space.

When the transmission begins, QEMU sets the HVA's base address into host shared memory, and use channel's file descriptor to notify channel object in host kernel. QEMU will also start a new thread listening on the *eventfd*, which is used to perform the writing back to domain action after host kernel return.

The data routing between domains. When the data memory is added to the host kernel, we should face the next question: how to manage the data flow between domains? This paper implements it referring to the router. In host kernel, we create a central router across all domains, and encapsulate the data memory into package transferred in router. The packages are categorized into producer and consumer, producer indicates the memory contains the data, and consumer has an empty memory need to be write in. Each package has these fields: source domain and source channel; target domain and target channel; the base address and length of data memory; the flag shows this package is producer or consumer; a group of *callbacks* define how packages are matched and the behaviours on matching.

As shown in Figure 4, the central router stores the producer and consumer respectively into two hash tables using the target domain and target channel as key. When a new package coming in, it will be matched to the other kind of package

if it is the target of that package. After packages matching, the defined *callbacks* will be called to perform data moving from producer's memory to consumer's memory.

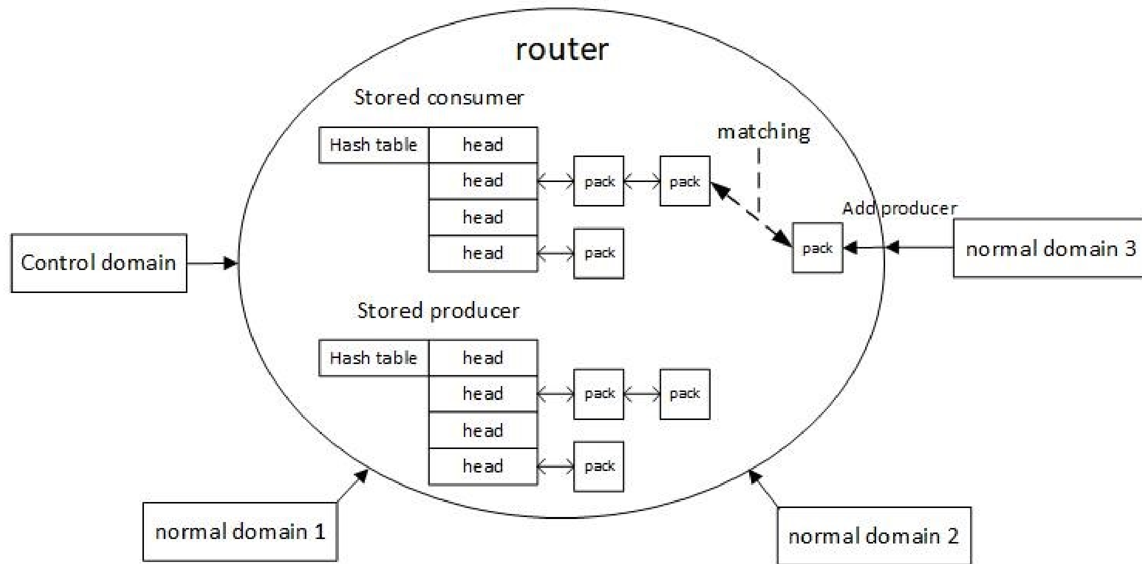

Figure 4. The data routing in host kernel.

4. EVALUATION

This paper developed multi-domain isolated system based on Linux system and QEMU/KVM hypervisor. We use the isolation feature of virtual machine to realize the isolation between domains. For example, each domain has its own storage image and accesses the data in its image independently using VIRTIO. It cannot get or modify the data in the other domain's image. We assume that the QEMU/KVM is secure itself and the virtual machine is isolated with each other completely. Under this assumption, we implement a secure communication mechanism, which enables data transfer between domains under monitoring. Therefore, this section mainly evaluates the effectiveness and performance of our inter-domain communication mechanism.

4.1 Experiment environment

The hardware and software environment we experiment on is listed in Table 1.

Table 1. Hardware and software environment.

Items	Detail information
Machine	i7-11800H
CPU logical cores	16
Memory	32GB
Linux kernel version	4.18
QEMU version	3.1.0

The hardware machine we used for evaluation is Dell G15 desktop with processor of 11th Gen Intel Core i7-11800H. It has 8 physical core, 16 SMT, with 32GB physical memory. The Linux kernel version is 4.18 and QEMU version is 3.1.0. We verified our design in Centos 8 system, compiling our work in host kernel as external module to enable them.

4.2 Effectiveness evaluation

The architecture, which is shown in Figure 1, was previously discussed. When we pass data from one domain to the other, the transmission process will be forced through the control domain, so that the control program can inspect the data. Therefore, we evaluate the effectiveness of communication mechanism by checking whether the data is same in sender, receiver and control program.

Table 2. The comparation of hash in three domains.

Package length	Sender domain	Receiver domain	Controller domain	match
64B	6b82472f	6b82472f	6b82472f	Yes
128B	33ae19da	33ae19da	33ae19da	Yes
256B	e79e28c3	e79e28c3	e79e28c3	Yes
512B	9e954cf7	9e954cf7	9e954cf7	Yes
1024B	0d5b1598	0d5b1598	0d5b1598	Yes
2048B	7ad2e533	7ad2e533	7ad2e533	Yes
4096B	9474c785	9474c785	9474c785	Yes

In different length, we generate strings randomly for sending to the other domain. We record the string respectively in sender domain, receiver domain and control program, and generate the hash from each string. As shown in Table 2, we record the first 8 bits of string's hash. We compare the string's hash and find they are same in three places, which verifies the effectiveness of our secure communication mechanism.

4.3 Efficiency evaluation

In the previous description, the transmission process starts from a common domain, through the control domain, and arrives at the target common domain. However, in contrast to the serial process of traditional request-return pattern, our data transmission process is parallel, which sender and receiver will add packages in the same time.

Figure 5 depicts this process. The consumer and producer add packages to router in parallel. Control domain intercept producer's package, inspect the content in it and add it back to router. The router will then match the consumer and producer, fill consumer's data memory with content. At last, the consumer package will return the content to domain. Therefore, we implement a test program, counting the add-return latency for sending and receiving process respectively.

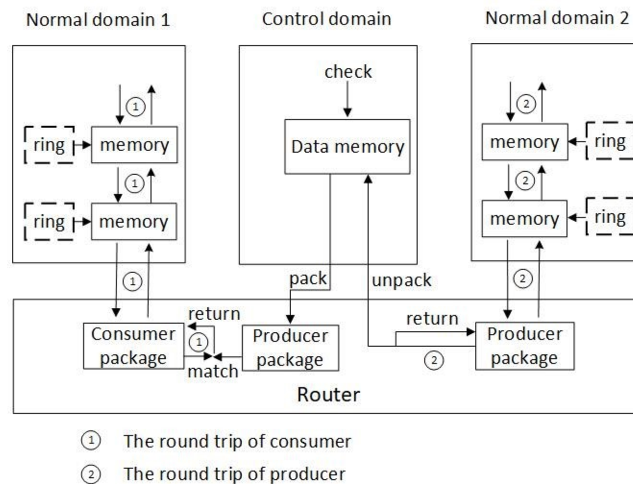

Figure 5. The double side data deliver process.

In our system, in order to make our communication channel the only way for domains to exchange data, we disable the virtual machine's network in host. However, for comparing the efficiency of our communication mechanism with traditional network in experiment, we use bridge device to establish a temporary LAN for VMs, and compare the network latency with our add-return latency. In Table 3, we count the max, min and average latency of sender, receiver and ping when the package length is 64 Bytes and 32 packages are sent consecutively. We can find that the average round-trip latency of both sender and receiver is much less than the ping latency.

Table 3. The time latency comparative experiment.

Latency per package (μs)	ping	sender	receiver
Max	505	472	398
Min	407	40	41
Total (32)	14829	2791	2398
Average	463	87	75

As shown in Figure 6, we also gather the statistics of average round-trip latency in different package length. It is found that the latency is not greatly affect by the package length. The ping latency ranges from 460 microseconds to 600 microseconds, the sender latency ranges from 65 microseconds to 100 microseconds, and the receiver latency ranges from 60 microseconds to 90 microseconds.

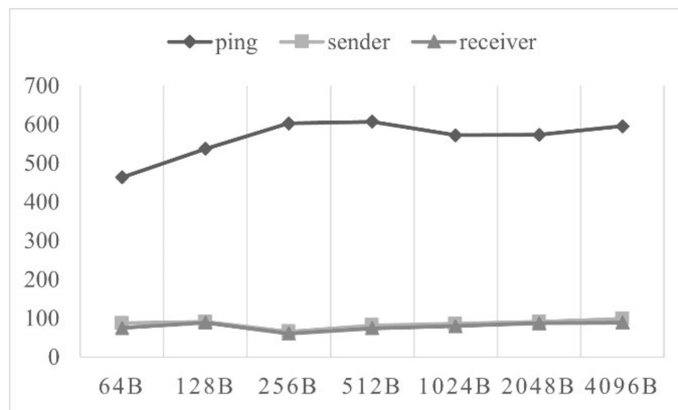

Figure 6. Latency varying with package length.

5. CONCLUSION

In this paper, we construct a multi-domain isolation system based on Linux and QEMU/KVM. And a secure communication mechanism is implemented for domains exchanging data under control. At last, we evaluate the effectiveness of inspection and efficiency of our communication mechanism. Proved by experiment, our communication mechanism can inspect the data flow effectively and the efficiency is better than network communication.

ACKNOWLEDGMENTS

Firstly, we need to thank Weihua Sun, for his early exploratory works. And we thank Wuxi Institute of Advance Technology and Shanghai HuaChengJinRui Co. Ltd. for the experiment equipment and their generous technical support. We also thank the anonymous reviewers for their valuable comments. This work is supported by project 2020-JCJQ-ZD-017.

REFERENCES

[1] Gautam, M., Jha, S., Sural, S., Vaidya, J. and Atluri, V., "Constrained policy mining in attribute based access control," Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies (SACMAT'17), 121-123 (2017).

[2] de Paoli, E. and di Biagio, C., "Multi-level Security Approach on Weapon Systems," Proceedings of 6th International Conference in Software Engineering for Defence Applications (SEDA 2018), Vol. 925, p. 36 (2019).

[3] Tian, X. and Song, H., "A zero trust method based on BLP and BIBA model," 2021 14th International Symposium on Computational Intelligence and Design (ISCID), 96-100(2021).

[4] McLean, J., "The specification and modeling of computer security," IEEE Computer, 23(1), 9-16(1990).

[5] Cruz, J. P., Kaji, Y. and Yanai, N., "RBAC-SC: Role-based access control using smart contract," IEEE Access, vol. 6, pp. 12240-12251(2018).

[6] Mitra, B., Sural, S., Vaidya, J. and Atluri, V., "Migrating from RBAC to temporal RBAC," IET Inf. Secur., 11, 294-300(2017).

[7] Kumar, A., Abhishek, K., Chakraborty, C. and Rodrigues, J. J. P. C., "Real geo-time-based secured access computation model for e-Health systems," Computational Intelligence, 1-18(2022).

[8] Thakare, Lee, E., Kumar, A., Nikam, V. B. and Kim, Y. -G., "PARBAC: Priority-attribute-based RBAC model for azure IoT cloud," IEEE Internet of Things Journal, 7(4), 2890-2900(2020).

[9] Block, K., Narain, S. and Noubir, G., "An autonomic and permissionless Android covert channel," Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'17), 184-194(2017).

[10] Tian, J., Xiong, G., Li, Z. and Gou, G., "A survey of key technologies for constructing network covert channel," Security and Communication Networks, 2020, 8892896(2020).

[11] Ge, Q., Yarom, Y., Cock, D., et al., "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," Journal of Cryptographic Engineering, 8(1), 1-27(2018).

[12] Ge, Q, Yarom, Y, Chothia, T., et al., "Time protection: The missing OS abstraction," Proceedings of the Fourteenth EuroSys Conference, 1-17(2019).

[13] Wang, S., Wang, P., Liu, X., Zhang, D. and Wu, D., "CacheD: Identifying cache-based timing channels in production software," 26th USENIX Security Symposium (USENIX Security 17), 235-252(2017).

[14] Hielscher, J., Lamshöft, K., Krätzer, C., and Dittmann, J., "A systematic analysis of covert channels in the network time protocol," 16th International Conference on Availability, Reliability and Security (ARES 2021), Article 69, 1-11(2021).

[15] Caviglione, L., "Trends and challenges in network covert channels countermeasures," Applied Sciences, 11(4), 1641(2021).

[16] Heinrich, M., Vateva-Gurova, T., Arul, T., Katzenbeisser, S., Suri, N., Birkholz, H., Fuchs, A., Krauß, C., Zhdanova, M., Kuzhiyelil, D., Tverdyshev, S. and Schlehuber, C., "Security requirements engineering in safety-critical railway signalling networks," Security and Communication Networks, 2019, 8348925(2019).