Secure Privacy Control inside Clouds with AMD SEV and Nested Virtualization

Naoya Ando
Kyushu Institute of Technology
naoya@ksl.ci.kyutech.ac.jp

Kazuki Takiguchi
Kyushu Institute of Technology
takiguchi@ksl.ci.kyutech.ac.jp

Kenichi Kourai
Kyushu Institute of Technology
kourai@csn.kyutech.ac.jp

***Abstract*—The leakage of personal data from public clouds has been a major issue in recent years. As cloud services become increasingly complex, e.g., using microservices and multicloud, personal data can be distributed to various services. However, the details of data flow inside clouds are not disclosed to users. Therefore, users cannot know how their personal data is processed and stored. To regain control of personal data, users need a privacy control mechanism for clouds, but the mechanism provided by clouds cannot be trusted. This paper proposes *SEV-tracker* for enabling secure privacy control inside clouds using a processor-based trusted execution environment (TEE). SEV-tracker injects a user hypervisor into a cloud virtual machine (VM). Using nested virtualization, the user hypervisor runs a cloud service in a user VM created on top of it and tracks and controls the data flow of the service. To mutually protect the user hypervisor and the cloud from each other, SEV-tracker applies AMD SEV to both VMs. We have implemented SEV-tracker using BitVisor as a lightweight user hypervisor and unikernels as cloud services to mitigate the overhead of nested virtualization. We conducted several experiments and examined the effectiveness of SEV-tracker.**

***Index Terms*—TEE, AMD SEV, VM, nested virtualization, data flow**

1. Introduction

Recent public clouds provide various services and inevitably deal with a huge amount of personal data. In addition, they provide more and more complex services by coordinating multiple services, e.g., using microservices and multicloud. As a result, personal data can be distributed to various services within a cloud and even across multiple clouds. Only if one of the services has a vulnerability or if there is one malicious cloud administrator, leakage of personal data can occur. For example, information on more than a hundred million customers leaked out from Captial One by the misconfiguration of a web application firewall [1]. Currently, end users have to use cloud services with the risk of such information leakage.

Although clouds must obey regulations such as GDPR, they do not disclose the details of how they internally process personal data to users. Therefore, users usually cannot know which cloud services their personal data is transferred to. Even if personal data is transferred to unintended services or leaked to attackers, users have no means to notice that. To regain control of personal data from clouds, users need a privacy control mechanism that allows them to track and control data flow inside clouds. Using such a mechanism, they could grasp the region of data distribution and the data store. This can deter clouds from illegally transferring personal data. Furthermore, they could prevent information leakage in advance by limiting data flow. Such a privacy control mechanism needs to run inside clouds, but users cannot trust the mechanism provided by clouds.

This paper proposes *SEV-tracker* for enabling secure privacy control inside clouds using a trusted execution environment (TEE), which is provided by recent processors. SEV-tracker injects a user's own hypervisor into a cloud's virtual machine (VM) used to provide a cloud service for the user. Using nested virtualization [2], the injected user hypervisor creates a user's VM on top of it and runs a cloud service in the VM. The privacy control mechanism in the user hypervisor monitors the user VM and tracks and controls the data flow of the cloud service. To protect the privacy control mechanism from the cloud, SEV-tracker encrypts the memory of the cloud VM using AMD Secure Encrypted Virtualization (SEV). For mutual protection, it also applies SEV to the user VM running in the cloud VM using Nested SEV [3] and protects the cloud service from the user hypervisor.

To mitigate the overhead of running a user VM in a cloud VM with nested virtualization, SEV-tracker uses a lightweight hypervisor and unikernels [4]. As a user hypervisor, it uses BitVisor [5], which runs only one VM and efficiently virtualizes only necessary device access. We modified BitVisor to capture network packets of a user VM and send communication logs to a user's log servers. SEV-tracker can visualize the communication logs to show data flow intuitively. To run a cloud service as a unikernel, SEV-tracker uses Unikraft [6], which enables developers to construct a unikernel with minimum functionalities of the operating system (OS) and runs only one application in a VM. We modified Unikraft to support SEV and run on top of BitVisor. We conducted several experiments and examined the effectiveness of SEV-tracker.

The organization of this paper is as follows. Section 2 describes the necessity of a privacy control mechanism for clouds. Section 3 proposes SEV-tracker for enabling

secure privacy control using SEV. Section 4 explains its implementation, and Section 5 shows experimental results. Section 6 describes related work, and Section 7 concludes this paper.

2. Privacy Control in Clouds

Recent public clouds deal with a huge amount of personal data in various services provided to users. An example of personal data is information that can identify and distinguish individuals. To protect personal data, regulations such as GDPR have been established in many countries. Since clouds must obey such regulations, unintended use of personal data is strongly restricted. However, it is not easy to prevent leakage of personal data from clouds because that happens unexpectedly even if clouds make great efforts to protect personal data.

One of the reasons for increasing information leakage is that clouds provide more and more complex services. In clouds, it is common to coordinate multiple services, especially in microservice architectures. Microservices are a method of developing software by dividing one complex service into multiple smaller services. Since a cloud service exchanges data between internal services, personal data can be distributed to various services if one service processes personal data. Furthermore, one cloud service can use several services provided by multiple clouds. In this multicloud deployment, personal data is transferred not only within a cloud but also across clouds.

In public clouds, the detailed flow of personal data that cloud services deal with is basically not disclosed to the users. Usually, users cannot know which cloud services their personal data is transferred to. They often cannot determine in which data centers their personal data is stored around the world. Hence, it is more difficult to control the flow of personal data. Private clouds and government clouds can be used to protect personal data. They can limit the flow of personal data to within one organization, country, or region. However, they inherently increase costs, compared to public clouds. In addition, users cannot confirm whether the flow of personal data is actually restricted as expected.

To regain control of personal data from clouds, a privacy control mechanism is needed for public clouds. It would enable users to track and control the flow of their personal data that cloud services deal with. Users could ask the privacy control mechanism which cloud services their personal data has been transferred to and which data centers it has been stored in. As such, they could always grasp the region of the distribution of their personal data and all the stores of their personal data. This can deter clouds from transferring personal data to unexpected regions. In addition, they could limit data flow in advance to decrease the possibility of information leakage. This allows users to use public clouds as private clouds in a sense.

However, users cannot fully trust the privacy control mechanism provided by public clouds themselves. This is because users have no means of confirming that clouds' privacy control mechanism tracks and controls personal data properly. Clouds can easily bypass their own privacy control mechanism and transfer personal data to data collection servers in secret. Even if cloud providers are trusted, insiders such as malicious administrators can exist in clouds. In fact, it is reported that 28% of cybercrimes are committed by insiders [7] and that 35% of administrators have stolen confidential information [8]. In addition, part of a cloud can be compromised by external attackers. Such insiders and attackers can easily disable the private control mechanism.

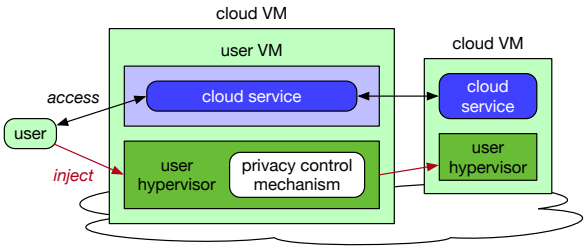


Figure 1: The system overview of SEV-tracker.

3. SEV-tracker

In this paper, we propose *SEV-tracker* for enabling secure privacy control inside public clouds using a processor-based trusted execution environment (TEE). Fig. 1 illustrates the system overview of SEV-tracker. It is desirable to run a user's privacy control mechanism in the cloud hypervisor included in the cloud infrastructure, but this is not realistic. Therefore, a cloud provides a dedicated VM to run its service, called a *cloud VM*, for each user. The cloud VM runs a hypervisor injected by the user, called a *user hypervisor*, using nested virtualization [2]. The user hypervisor creates a VM, called a *user VM*, on top of it and runs a cloud service in the VM. A privacy control mechanism runs in the user hypervisor and tracks and controls the data flow of the cloud service, e.g., network and disk access. The user accesses the cloud service running in the user VM. If necessary, the cloud service accesses other cloud services running on top of other user hypervisors.

Using SEV-tracker, individual end users could inject their own user hypervisors into clouds, but this is too costly. Since at least one cloud VM is required per user, the number of VMs would increase dramatically in clouds. Therefore, we assume that end users can trust their organizations such as their companies, universities, etc. Instead of end users, the organization injects its own hypervisor and shares a cloud service running on top of that user hypervisor among its constituent members. Then, the members share the data flow tracked by the privacy control mechanism for that cloud service. Since the privacy control mechanism does not expose personal data itself, privacy is kept among the members. If end users cannot trust their organizations, they may be able to rely on trusted third parties. They can use a cloud service running on the user hypervisor injected by a trusted third party. In any case, we believe that the cost is much less than using private clouds.

In our threat model, we assume that a cloud and its users do not trust each other. Therefore, an injected user hypervisor and a cloud running the user hypervisor have to be protected from each other inside the cloud. Thanks to the isolation of a cloud VM, cloud infrastructure can be protected from the user hypervisor. However, the user hypervisor can suffer from attacks by the cloud because cloud infrastructure has higher privileges than the user hypervisor running in a cloud VM. In such a case, the cloud can bypass the user's privacy control mechanism and tamper with tracking information. This makes it impossible to track and control data properly. Similarly, the user hypervisor is protected from a cloud service by the isolation of a user VM. In contrast, the user hypervisor can eavesdrop on and tamper with the data of the cloud service in a user VM. The data includes not only the user's but also the cloud's.

SEV-tracker provides mutual protection by applying AMD SEV to these two VMs. SEV is a security feature of AMD EPYC processors. It transparently encrypts the memory of a VM using a hardware key. SEV-ES also encrypts the register state of the virtual CPUs, and SEV-SNP checks the integrity of the memory. Thus, SEV prevents even the hypervisor from accessing unencrypted data in a VM. To protect a user hypervisor from cloud infrastructure, SEV-tracker applies SEV to a cloud VM. The cloud cannot disable the privacy control mechanism in the user hypervisor or alter the policy used for privacy control. SEV-tracker also applies SEV to a user VM running in the cloud VM using Nested SEV [3] to protect a cloud service from the user hypervisor. The user hypervisor cannot attack the cloud service. Remote attestation for SEV can verify that unmodified user hypervisor and cloud service run in cloud and user VMs, respectively.

Since SEV-tracker uses nested virtualization to inject a user hypervisor between cloud infrastructure and a cloud service, it increases the overhead of the cloud service. To mitigate this overhead, SEV-tracker uses a lightweight hypervisor as a user hypervisor. A hypervisor usually supports multiple VMs, but it is sufficient to support only one VM because the user hypervisor can be specialized to monitor only one cloud service. The user hypervisor can eliminate the functionalities required to run multiple VMs. In addition, it basically does not need to virtualize devices, although a usual hypervisor needs device virtualization to share one device with multiple VMs. A user VM can directly access the devices provided by a cloud VM using device passthrough. Then, the user hypervisor virtualizes only devices that are required to track and control data flow.

Furthermore, SEV-tracker runs a unikernel [4] as a cloud service in a user VM to eliminate the overhead of the general-purpose OS. A unikernel is a specialized appliance constructed using a library OS. It links only necessary OS functionalities provided as libraries to an application. Thus, it efficiently runs in a single address space and requires less memory. Thanks to minimal initialization, a unikernel can also reduce the boot time. This is suitable for our usage of running a cloud service per user. Using a unikernel makes the overhead of a user VM close to that of an OS process. SEV-tracker is expected to achieve the performance between single-level virtualization and fully nested virtualization.

To enable users to track and control data flow, SEV-tracker monitors all the communications of a cloud service. Whenever a cloud service in a user VM sends or receives a network packet, the user hypervisor captures it and records its information in a communication log. If that communication is not permitted by the user's policy, the privacy control mechanism in the user hypervisor drops that packet. For example, the user can limit the communications of the cloud service within one cloud. To examine the communications performed by the cloud service, the user can obtain the communication logs from the privacy control mechanism. SEV-tracker visualizes data flow recorded in the communication logs to show it to the user intuitively. The obtainable information is coarse-grained due to the protection of a cloud service, but it is useful for the user to check the possibility of illegal data flow. In addition, it can deter clouds from illegal data flow.

4. Implementation

We have implemented SEV-tracker using BitVisor as a user hypervisor and Unikraft as a unikernel to run a cloud service. BitVisor is a lightweight hypervisor and has no capabilities necessary to run multiple VMs. In SEV-tracker, it runs in a cloud VM created on top of KVM and creates only one user VM. It basically provides the virtual devices of the cloud VM directly to the user VM using device passthrough. If necessary, it can efficiently virtualize only part of device access using a para-passthrough mechanism. Unikraft is a unikernel development kit to facilitate the development of applications using a library OS. Developers select and compile only libraries that provide the OS functionalities needed to run their applications. In SEV-tracker, a unikernel runs in a user VM on top of BitVisor.

We added SEV support to BitVisor and Unikraft. In addition, we addressed several issues to run Unikraft on BitVisor.

4.1. Fast Deployment of Cloud Services

To execute an injected BitVisor into a cloud VM, the user creates a disk image for BitVisor. The disk image contains the boot loader needed to boot BitVisor in UEFI and the ELF binary of BitVisor. In addition, it contains a UEFI shell script to automatically boot BitVisor and a cloud service. The user transfers the created disk image to a server in a cloud when he uses a cloud service. If the cloud service accesses other services in the same cloud or different clouds, the cloud server forwards the disk image to other cloud servers recursively.

Fig. 2 illustrates the detailed system architecture of SEV-tracker. SEV-tracker requires KVM that supports SEV as cloud infrastructure. It uses OVMF as UEFI BIOS, which is provided by a cloud. A cloud VM uses both a user's disk image for BitVisor and the disk image for booting a cloud service, which is provided by a cloud. The latter disk image

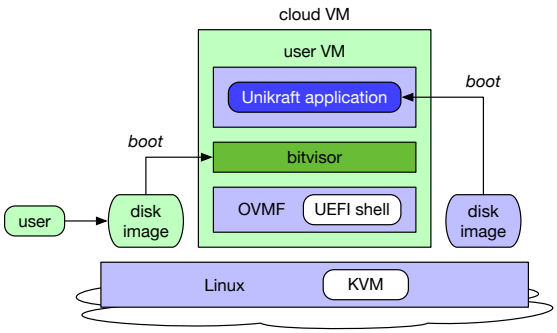


Figure 2: The system architecture of SEV-tracker.

contains a Unikraft application and a UEFI shell script for its automatic boot. The running cloud service uses the host filesystem in a cloud. SEV-tracker confirms that OVMF, BitVisor, and the cloud service are legitimately booted by using the remote attestation of SEV.

SEV-tracker creates a cloud VM on KVM and runs OVMF in the VM when it boots a per-user cloud service. After OVMF executes the UEFI shell, the UEFI shell executes the shell script contained in the user's disk image and automatically boots BitVisor. This shell script first executes the boot loader of BitVisor as a UEFI application. The boot loader boots BitVisor in the cloud VM, and BitVisor creates a user VM in the cloud VM. After that, the control is returned to the UEFI shell, but the UEFI shell is executed in the execution context of the user VM. Next, SEV-tracker boots a cloud service in the user VM. To run the cloud service provided by a cloud, the user's UEFI shell script invokes the UEFI shell script contained in the cloud's disk image.

4.2. Tracking and Controlling Communication

In SEV-tracker, the injected BitVisor captures all the packets sent to and received from a user VM and analyzes packets to track and control the communication. To capture packets, SEV-tracker uses the pass module of BitVisor. The pass module allows a Unikraft application in the user VM to directly use the virtual NICs of the cloud VM, whereas it enables BitVisor to hook access to the virtual NICs. BitVisor analyzes the Ethernet, IP, TCP, and UDP headers in a packet and collects information on the source and destination IP addresses and port numbers. If the collected information exactly matches the previous one, it just forwards that packet. If the user's policy denies forwarding that packet, BitVisor discards it.

BitVisor sends the recorded communication log to the user's log server using the syslog protocol with UDP. SEV-tracker boots a log server in another cloud VM protected by SEV. BitVisor creates UDP packets using the IP address of the log server and the MAC address of the log server or the gateway. Then, it directly passes the packets to the virtual NIC. Note that BitVisor does not use the network stack provided by lwIP [9] included in it because the pass module does not enable lwIP. BitVisor adds a sequence number to the communication log to detect packet drop by the cloud. It encrypts the log using AES to prevent the cloud from eavesdropping on it. Also, it calculates a message authentication code (MAC) and sends it with the log.

When the log server receives the encrypted communication log, it decrypts that log and checks the sequence number. If the sequence number is not contiguous, it records that the cloud may discard communication logs. Note that the cloud did not always discard it because UDP packets can be dropped or received out of order. Next, the log server recalculates a MAC from the received communication log and compares it with the received MAC. If the two MACs do not match, it records that the communication log may be tampered with. If the integrity of the communication log is preserved, the log server saves the log in its database.

4.3. Visualizing Data Flow

SEV-tracker visualizes the communication logs and shows data flow inside clouds to the user intuitively. It uses net-glimpse [10] to visualize the communication between cloud services. Net-glimpse is a tool to obtain packets from a NIC in real time and visually shows communication in a browser. It shows a host as a node with an IP address and packet sending and receiving as an arrow. If already-shown nodes send or receive packets again, net-glimpse emphasizes the nodes. It shows the protocol name or port number on an arrow.

However, SEV-tracker cannot use net-glimpse as it is because communication logs do not contain the entire packets and are obtained from log servers instead of a NIC. Therefore, SEV-tracker creates a TAP device as an alternative to a NIC. TAP is a device that can simulate an Ethernet device and manipulate the data link layer. SEV-tracker reconstructs a packet header from the IP address, port number, and protocol number contained in communication logs and generates an Ethernet frame. It fills arbitrary values in the other fields such as the MAC address. Then, it writes the generated Ethernet frame to the TAP device. Consequently, net-glimpse can capture the packet from the device and visualize communication by that packet.

SEV-tracker supports three modes of data-flow visualization. It can visualize communication logs after the user finishes using cloud services. Using this mode, the user can grasp the overview of the communication performed by the cloud services by showing information at once. If SEV-tracker shows communication information one by one, the user can grasp the data flow of the cloud services more easily. In addition, SEV-tracker can periodically collect communication logs from log servers and visualize them. This allows the user to grasp data flow in near real time.

5. Experiments

We conducted several experiments to examine the effectiveness of SEV-tracker. First, we confirmed whether the

TABLE 1: The used cloud servers.

	host 1	host 2
CPU	AMD EPYC 7402P	AMD EPYC 7262
memory	256 GB	128 GB
NIC	Broadcom 57416	
OS	Linux 5.4	
hypervisor	QEMU-KVM 4.2.1	

user could easily grasp data flow in a cloud by visualization. Then, we inspected the performance of SEV-tracker. For cloud servers, we used two hosts described in Table 1. We created one cloud VM per host and assigned one virtual CPU, 400 MB of memory, and a virtual e1000 NIC to each cloud VM. In a cloud VM, we ran BitVisor as a user hypervisor. We created one user VM in a cloud VM and ran Unikraft 0.15 as a guest OS. For comparison, we ran KVM in Linux 5.4 and QEMU-KVM 4.2.1 as a user hypervisor in a cloud VM and Linux 5.4 as a guest OS in a user VM. In this case, we assigned 4 GB of memory to a cloud VM and 400 MB of memory to a user VM. Due to the instability of our implementation, we applied SEV to cloud and user VMs only for the measurement of the deployment time.

5.1. Visualization of Data Flow

We have developed a cloud service as a web application using the Flask web application framework. Using SEV-tracker, we ran the cloud service on the two servers. The two cloud services communicated with each other. The user accessed cloud service 1 on host 1 from the client PC using HTTPS. Cloud service 1 first accessed a DNS server for name resolution and then accessed cloud service 2 on host 2 using HTTPS. Cloud service 2 accessed an external service using HTTPS. We assumed that this communication with the external service was unintended. In this experiment, we applied SEV-tracker only to the two cloud services. We ran a log server on each host and obtained communication logs from the log servers.

Fig. 3 shows the visualized data flow of the cloud services. For ease of understanding, we added the roles of the hosts using red labels. We confirmed that the obtained data flow matched the actual data flow. Using this data flow, the user could detect illegal communication with the external service. Note that the communication logs for the DNS server and the external service may be different from actual communication because the cloud can change the destinations after BitVisor records those logs. To guarantee the correctness of the logs, we need to apply SEV-tracker to those services as well and crosscheck the logs of both the source and destination.

5.2. Performance

We measured the time needed to deploy the developed cloud service. In this experiment, we defined the deployment time as the time until the network in the user VM was activated. For comparison, we measured the deployment

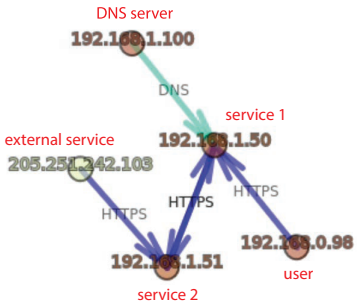
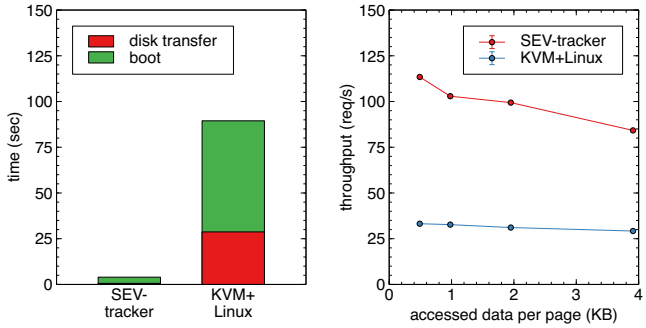


Figure 3: The visualized data flow of cloud services.



(a) Deployment time (b) Service performance

Figure 4: The performance of SEV-tracker.

time when transferring a disk image for KVM and booting KVM and Linux. Fig. 4(a) shows the deployment time and its breakdown. The deployment time was 4.0 seconds in SEV-tracker. It took 0.6 seconds to transfer the disk image and 3.4 seconds to boot BitVisor and the Unikraft application. SEV-tracker could deploy the cloud service 22x faster than the combination of KVM and Linux. In more detail, SEV-tracker could transfer the disk image 49x faster and boot the cloud service 18x faster. When we used KVM and Linux, the transfer time increased because the size of the disk image became 328x larger. The increase in boot time was caused by booting general-purpose Linux in both cloud and user VMs.

Next, we measured the performance of cloud services using SEV-tracker. We used the cloud service that simulated big data analysis, which processed requests using a large amount of data. This cloud service allocated a large amount of memory per HTTP request, accessed it, and released it. Fig. 4(b) shows the throughput when the cloud service allocated 100 MB of memory and changed the data size accessed per page. SEV-tracker always outperformed the combination of KVM and Linux, whereas its performance degraded gradually when the accessed data size increased. In Linux, the performance was slightly affected by the accessed data size. This is because the impact of the accessed data size was relatively small because heavyweight memory swaps and page faults were bottlenecks.

6. Related Work

Several monitoring systems have been proposed using Intel SGX, another popular TEE. SGX is a security feature of Intel processors and enables a program to be securely executed in a protection domain called an enclave. Ryoan [11] creates a sandbox with Google NaCl inside an SGX enclave and runs a cloud service in it. NaCl checks the code executed in a sandbox and performs the runtime check of its behavior. Similarly, AccTEE [12] uses WebAssembly to create a sandbox in an SGX enclave. Like NaCl, WebAssembly can confine a cloud service to a sandbox and execute it securely. Unlike Ryoan and AccTEE, SEV-tracker creates a sandbox with a VM created in an SEV-enabled VM.

Nested enclave [13] extends SGX hardware to enable the creation of inner enclaves in an outer enclave. An outer enclave cannot access its inner enclaves, whereas inner enclaves can access their outer enclave. Inner enclaves are isolated from each other. Since an inner enclave needs to invoke its outer enclave to perform I/O, the outer enclave can monitor the data flow of a cloud service running in the inner enclave. Unlike SEV-tracker, the monitoring system is not protected from the cloud service because an inner enclave can access the memory of its outer enclave.

CloudVisor [14] uses nested virtualization to deploy the security monitor below cloud infrastructure to protect VMs from cloud operators. The security monitor restricts access to VMs by the cloud hypervisor. It also protects the virtual disks of VMs by encryption and integrity checking. If users can inject their security monitor into clouds, they could easily monitor the data flow of cloud services in VMs. However, that is not acceptable to clouds because the security monitor has too high privileges. In SEV-tracker, an injected user hypervisor is confined to a cloud VM.

Xen-Blanket [15] runs a user hypervisor in a cloud VM with nested virtualization to enable the user to migrate a user VM on top of it across clouds. To reduce the overhead of network communication in a user VM, it uses a paravirtual network driver in the host OS of a cloud VM and efficiently accesses the back-end driver running in the host OS underlying the cloud VM. This mechanism achieves network throughput comparable to single-level virtualization. Since Xen-Blanket does not need to modify cloud infrastructure, it can be used in existing clouds. SEV-tracker is also applicable to existing clouds if cloud infrastructure supports SEV for cloud and user VMs.

7. Conclusion

This paper proposes SEV-tracker for enabling secure privacy control inside clouds by protecting VMs using SEV. Using a user hypervisor injected into a cloud VM, SEV-tracker allows the user to securely track and control the data flow of a cloud service. To mitigate the overhead of nested virtualization, SEV-tracker uses a lightweight hypervisor as a user hypervisor and unikernels for running cloud services. We have implemented SEV-tracker using BitVisor and Unikraft and examined its capability and performance.

One of our future work is to examine the performance of SEV-tracker using SEV-enabled VMs. We need to make SEV support for BitVisor and Unikraft more stable. Then, we are planning to run various microservices and multicloud services and track and control their data flow. In addition, it is necessary to visualize the data flow of cloud services more intuitively. Also, it is needed to track and control other types of data flows such as disk access.

Acknowledgments

This work was supported by JST, CREST Grant Number JPMJCR21M4, Japan.

References

[1] Capital One. (2019) Information on the Capital One Cyber Incident. [Online]. Available: https://www.capitalone.com/digital/facts2019/

[2] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour, "The Turtles Project: Design and Implementation of Nested Virtualization," in *Proc. USENIX Conf. Operating Systems Design and Implementation*, 2010, pp. 423–436.

[3] K. Takiguchi and K. Kourai, "Protecting Nested VMs with AMD SEV," Poster at ACM SIGOPS Asia-Pacific Workshop on Systems, 2024.

[4] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library Operating Systems for the Cloud," in *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2013, pp. 461–472.

[5] T. Shinagawa, S. Hasegawa, T. Horie, Y. Oyama, S. Chiba, H. Eiraku, K. Tanimoto, M. Hirano, E. Kawai, Y. Shinjo, K. Omote, K. Kourai, K. Kono, and K. Kato., "A Thin Hypervisor for Enforcing I/O Device Security," in *Proc. Int. Conf. Virtual Execution Environments*, 2009, pp. 121–130.

[6] S. Kuenzer, V. Badoiu, H. Lefeuvre, S. Santhanam, A. Jung, G. Gain, C. Soldani, C. Lupu, S. Teodorescu, C. Raducanu, C. Banu, L. Mathy, R. Deaconescu, C. Raiciu, and F. Huici, "Unikraft: Fast, Specialized Unikernels the Easy Way," in *Proc. European Conf. Computer Systems*, 2021, pp. 376–394.

[7] PwC, "US Cybercrime: Rising Risk, Reduced Readliness," PwC, Tech. Rep., 2014.

[8] CyberArk Software, "Global IT Security Service," CyberArk Software, Tech. Rep., 2009.

[9] A. Dunkels. lwIP – A Lightweight TCP/IP Stack. [Online]. Available: https://savannah.nongnu.org/projects/lwip/

[10] K. Lange. net-glimpse. [Online]. Available: https://github.com/kristian-lange/net-glimpse

[11] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "A Distributed Sandbox for Untrusted Computation on Secret Data," in *Proc. USENIX Symp. Operating Systems Design and Implementation*, 2016, pp. 533–549.

[12] D. Goltzsche, M. Nieke, T. Knauth, and R. Kapitza, "AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting," in *Proc. Int. Middleware Conf.*, 2019, pp. 123–135.

[13] J. Park, N. Kang, T. Kim, Y. Kwon, and J. Huh, "Nested Enclave: Supporting Fine-grained Hierarchical Isolation with SGX," in *Proc. ACM/IEEE Annual Int. Symp. Computer Architecture*, 2020, pp. 776–789.

[14] F. Zhang, J. Chen, H. Chen, and B. Zang, "CloudVisor: Retrofitting Protection of Virtual Machines in Multitenant Cloud with Nested Virtualization," in *Proc. ACM Symp. Operating Systems Principles*, 2011, pp. 203–216.

[15] D. Williams, H. Jamjoom, and H. Weatherspoon, "The Xen-Blanket: Virtualize Once, Run Everywhere," in *Proc. ACM European Conf. Computer Systems*, 2012, pp. 113–126.