Monitoring and Attestation of Virtual Machine Security Health in Cloud Computing

Cloud customers need assurances regarding the security of their virtual machines operating within an infrastructure-as-a-service cloud system. This article presents an architecture that can monitor a virtual machine's security health and communicate this to the customer in an unforgeable manner. The authors demonstrate a concrete implementation of property-based attestation and a full prototype based on the OpenStack open source cloud software.

Tianwei Zhang
Ruby B. Lee
Princeton University

In an infrastructure-as-a-service cloud, a customer can request to launch a virtual machine (VM) in the cloud system. The cloud provider places the VM in a virtualized cloud server and allocates a specified amount of physical resources (for example, processors, memory, disk, and networking) to this VM. Previous research has suggested a security-on-demand service model, in which secure computing platforms are dynamically provisioned to customers according to their specific security needs.[1] During the VM's lifetime, the customer would like to know if the VM has good security health. A healthy VM satisfies the security properties (for example, confidentiality, integrity, availability, or auditability) the customer requested for his or her leased VM. A VM's security health should take into account attacks from within the VM (such as malware or guest OS root kits), as well as those from other coresident VMs on the same server. Past research has shown that bad neighbor VMs can steal critical information through side-channel attacks,[2,3] thus compromising the VM's confidentiality health. Resource contention between different VMs on the same server motivates malicious VMs to perform resource-freeing attacks,[4] thus compromising the victim VM's availability health. Large cloud management software, including the hypervisor, will also have bugs[5] that can be exploited to compromise a VM's security health. Thus, a VM's security health depends on not only the activities inside the VM, but also on the VM's interactions with its environment.

Monitoring the VMs' security health poses a series of challenges in a cloud system. First, the customer's desired security requirements are expressed in high-level terms for his or her VM, but the security measurements usually involve low-level measurements of the physical server, hypervisor, and other entities related to this VM. This creates a semantic gap between what the customer wants to monitor and the type of measurements that can be collected. Second, the VMs go through different life-cycle stages and can migrate to different host servers. A seamless monitoring mechanism throughout the VMs' lifetime is therefore highly desirable. Third, there are numerous entities between the customer and the VM. It is important to collect, filter, and interpret the measured information securely to attest—that is, pass on to the customer in an unforgeable way—only the requested information.

Traditional binary attestation based on the Trusted Platform Module (TPM) lets remote customers verify the start-up integrity of the attested platform and establish a chain of trust from the bootloader to applications for measured boot up.[6–8] The VM will then execute on these verified trusted components. Some research introduced a privileged trusted party to perform integrity and configuration attestation for customers.[9,10] However, a VM's security health can also be affected by untrusted components (such as colocated VMs) outside of the trusted domains at runtime. The traditional attestation cannot verify if these colocated VMs will conduct runtime attacks (such as side-channel, covert-channel, or denial-of-service attacks) and compromise the monitored VM's security health (for example, its confidentiality, availability, and integrity). Past work proposed the concept of property-based attestation to attest different properties, functions, and behaviors of the system.[11] However, the specification and implementation of properties to be measured and attested remain challenging open problems.

We designed a flexible architecture called CloudMonatt to monitor the security health of customers' VMs within a cloud system. CloudMonatt is built on the property-based attestation model, and it provides several novel features. First, it defines VM security health for several different security properties and provides a framework for monitoring different aspects of security health. Second, it shows how to interpret and map actual collected measurements to security properties that can be understood by the customer. These features bridge the semantic gap between requested VM properties and the platform measurements for security health. Third, to the best of our knowledge, this is the first concrete realization of property-based attestation for a VM. Finally, CloudMonatt provides remediation response strategies based on the monitored results.

Architecture

Figure 1a shows an overview of the CloudMonatt architecture, which includes four entities: the cloud customer, cloud controller, attestation server, and cloud servers. We assume that the cloud controller and attestation server are trusted—that is, they are correctly implemented, with secure boot up, and are protected during runtime. These servers can be redundancy-protected for reliability and security, and they are only a small percent of all the servers in the cloud's datacenter. However, the numerous cloud servers need not be trusted, except for the trust module and monitor module in each server.

Cloud Customer

The customer is the system's initiator and end-verifier. He or she places a request for leasing VMs with specific resource and security requirements to the cloud controller. CloudMonatt gives customers two modes of operation: one-time attestation, in which the customer can request an attestation at any time during the VM life cycle, and periodic attestation, in which the customer can ask for periodic attestations with a prespecified or random frequency.

Cloud Controller

The cloud controller acts as the cloud manager and is responsible for taking VM requests and servicing them for each customer. The policy validation module in the controller selects qualified servers for customers' requested VMs. These servers need to both satisfy the VMs' demanded physical resources and support the requested security properties and their property monitoring services. The deployment module allocates each VM on the selected server.

During the VMs' life cycle, customers can request the cloud controller to monitor the security properties associated with their VMs. The cloud controller will entrust the

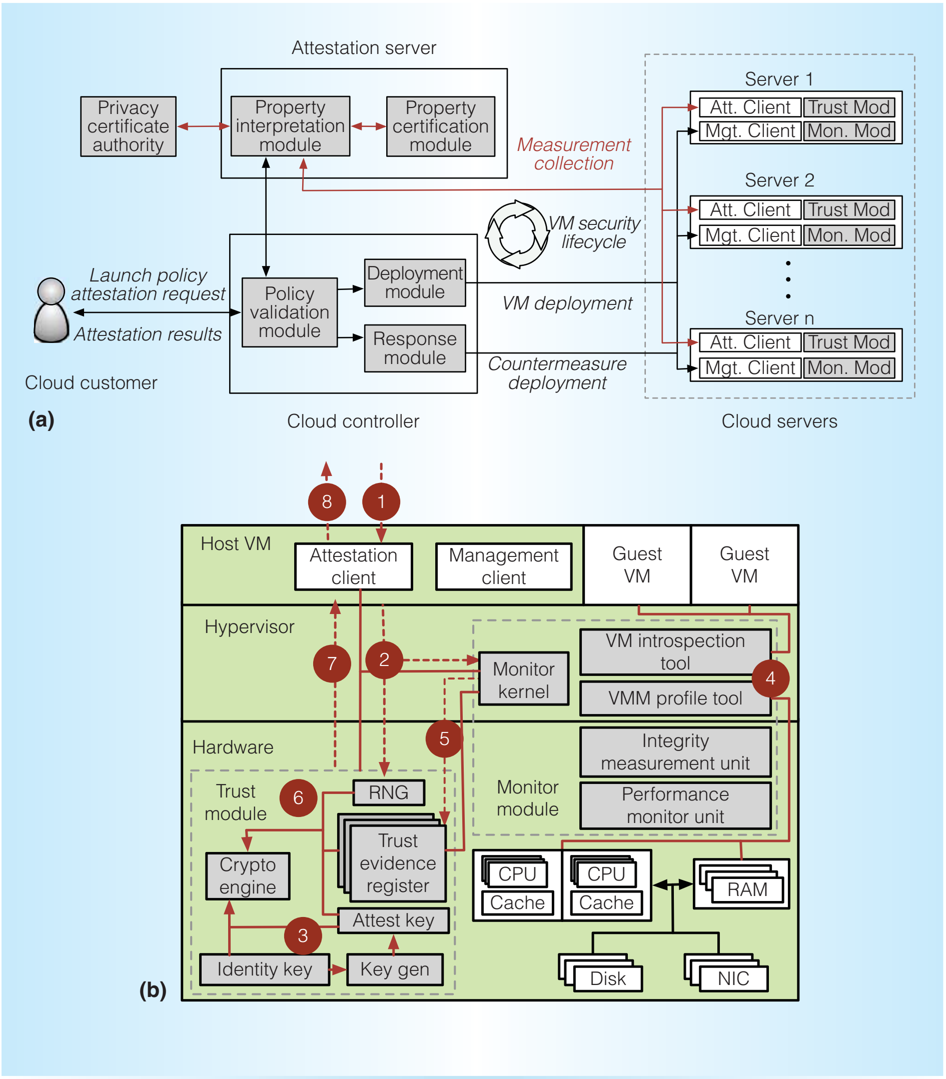

Figure 1. CloudMonatt. (a) Overview of the architecture, which includes a cloud customer, cloud controller, attestation server, and cloud servers. (b) New modules in a cloud server. Gray blocks are trusted.

attestation server to collect the monitored security measurements from the correct VMs and send a report back to it. It then sends the attestation results back to the customers to keep them informed of the VMs' security health. When these results reveal potential vulnerabilities for the VMs, the response module in the controller carries out appropriate remediation responses.

One such response, for example, is termination—the cloud controller can shut down the VM to protect it from attacks. Another possible response is suspension—the controller can temporarily suspend the VM when it detects that the platform's security health might be questionable. Meanwhile, it can initiate further checking and also continue to attest the platform. If the attestation results show the cloud server has returned to the desired security health, the controller can resume the VM from the saved state. And finally, another response is migration, which occurs when the current server's security health is questionable or the server has been compromised. In this case, the controller tries to find another secure cloud server that can satisfy the VM's security property requirements. The VM is suspended until a suitable server is found, then the controller migrates the VM to that server.

Attestation Server

The attestation server acts as the attestation requester and appraiser and comprises two essential modules. The property interpretation module validates measurements, interprets properties, and makes attestation decisions. It obtains a list of measurements that can indicate the security health with respect to the specified property. It needs a certificate from a privacy certificate authority (PCA) to authenticate cloud servers. The PCA could be a separate trusted server already used by the cloud provider for standard certification of public-key certificates that bind a public key to a given machine. The second essential module is the property certification module, which issues an attestation certificate for the properties monitored. There can be different attestation servers for different clusters of cloud servers, enabling the CloudMonatt architecture's scalability.

We introduce the attestation server for security monitoring and attestation, whereas the cloud controller is responsible for management. This job split achieves better scalability, because attestation servers can be added to handle more cloud servers. It consolidates property interpretation in the attestation servers, rather than replicating this in each cloud server or burdening the cloud controller. This also achieves better separation-of-duties security, because the cloud controller needs only to focus on cloud management while the attestation server focuses on security. It also improves performance by preventing a bottleneck at the cloud controller if it had to handle management as well as myriad attestation requests and security property interpretations.

Cloud Server

The cloud server is the computer that runs the VMs in question. It is the system's attester, and it provides measurements for different security properties. Figure 1b shows the structure of a cloud server with a Type-I hypervisor (for example, Xen). This has the hypervisor sitting on bare metal and a privileged VM called the host VM (or Dom0) running over the hypervisor. To support CloudMonatt's goals, a cloud server must include a monitor module and a trust module.

The monitor module contains different types of monitors to provide comprehensive and rich security measurements. These monitors can be software modules or existing hardware mechanisms such as a performance monitor unit or the TPM chip. For example, the performance monitor unit (present ubiquitously in Intel x86 and ARM processors) has numerous hardware performance counters to collect runtime measurements of the VMs' activities. An integrity measurement unit (which could use a TPM chip) can measure accumulated hashes of the system's code and static data configuration. In the hypervisor, a VM introspection tool can collect the information inside the specified VM, and the virtual machine monitor (VMM) profile tool can be used to collect dynamic information about each VM's activities.

We define a new hardware trust module in Figure 1b. This trust module is responsible for server authentication using its identity key, crypto operations using the crypto engine, key generation and random number generation, and secure measurement storage using the trust evidence registers. By using new hardware registers to store the security health measurements (trust evidence), we do not need to include the main DRAM memory in our trusted computing base, although we also could use trusted RAM instead of trust evidence registers in the trust module.

Figure 1b also shows the functional steps taken by the monitor module and the trust module. The cloud server includes an attestation client in the host VM that takes requests from the attestation server to collect a set of measurements (step 1 in the figure). It invokes the monitor module to collect the measurements (step 2) and the trust module to generate a new attestation key for this attestation session (step 3). This new attestation key is signed by the trust module's private identity key. The required measurements of suspicious events or evidence of trustworthy operation are then collected from the monitor module (step 4) and stored into new trust evidence registers (step 5). The trust module then invokes its crypto engine to sign these measurements (step 6) and forwards the data to the attestation client (step 7), which sends it to the attestation server (step 8). The trust module contains a key generator and a random number generator for generating keys and nonces (that is, random numbers that can be used only once).

Monitoring and Attestation Protocols

An adversary who has full control of the network between different servers can eavesdrop or falsify the attestation messages to make the customer receive a forged attestation report without detecting anything suspicious. So, secure communications are required among the four entities in Figure 1a. CloudMonatt expects the cloud controller, attestation server, and secure cloud servers to have long-term public–private key pairs for identification. Then, the entities in CloudMonatt talk with each other using SSL protocols. Nonces are used to prevent replay attacks. Our previous work offers more details about the protocols.[12]

Case Studies

CloudMonatt can support traditional TPM-based integrity attestation,[6] as well as attestations of various other properties.

Integrity

When customers launch VMs in CloudMonatt, they might want to check the integrity of both the host platform and the VM image. CloudMonatt will check the VM start-up integrity in two phases. First, it measures (that is, hashes) the server's platform configuration (including its hypervisor, host OS, and so on) during server boot up, and the respective hash values are stored securely in the TPM's platform configuration registers or in the trust evidence registers if a TPM is not used. Second, the VM image is measured before VM launch. The attestation server can have full knowledge of the attested software and have the correct precalculated hash values of its executable files. It can use these correct hash values to check the hash measurements sent back by the cloud server, and it can issue the integrity property attestation if the hash values match. Alternatively, the attestation server can use a trusted appraiser system (such as the integrity measurement architecture[8]) to check if the measured hash values conform to the correct values for a pristine, malware-free system before sending the start-up integrity property attestation back to the customer.

In addition to start-up integrity, CloudMonatt lets customers check if their VMs are infected with malware during runtime. Different VM introspection (VMI) tools can be integrated seamlessly as hypervisor-level tools to monitor the VM from outside the VM and examine the states of the target VM.

Confidentiality

We give an example of detecting confidentiality vulnerabilities via covert channels to show an interesting use of CloudMonatt's trust evidence registers. The detection method is based on detecting probability distributions characteristic of some covert communications, similar to the method proposed by Jie Chen and Guru Venkataramani.[13] Our threat model considers simple attacks. There are many types of covert channels,[13–17] and this detection method cannot cover all of them. The real purpose of this example is to show how CloudMonatt can use trust evidence registers to collect security measurements—for example, to build an empirical probability distribution for attack detection. The detection of various attacks is orthogonal to our work, and new methods can be integrated into CloudMonatt.

Although VMs are isolated from each other by the hypervisor, it could still be possible to leak confidential data via a cross-VM covert channel at VM runtime. A covert channel exists when a colluding insider (such as a program inside the victim VM) can use a medium not normally used for communications to leak secret information to an unauthorized party in another VM. When VMs on the same server share physical resources, the contention for these shared resources can be exploited to encode and transmit information, for example, in the form of timing features. Such characteristics can be different cache operations (hit or miss),[2,16] memory bus activities (locked or unlocked bus),[17] or DRAM controller states (bandwidth saturated or not). Figure 2a shows the covert channel information observed by the receiver VMs, using each of the last-level cache (LLC), bus, and DRAM as the covert channel communication

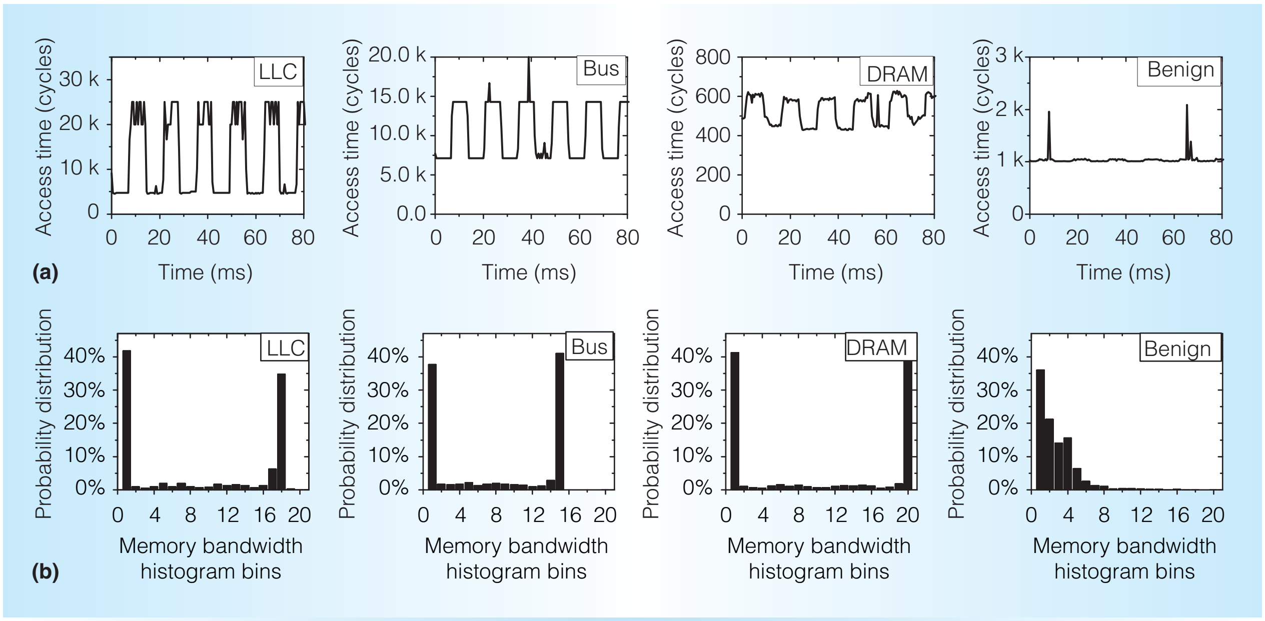

Figure 2. Detection of covert channel attacks. (a) Average access time of the receiver virtual machine (VM) when it keeps reading a preallocated data array. The first three graphs from left to right show covert channel communications, whereas the rightmost graph shows the victim VM running a benign application. (b) Memory bandwidth histogram of the attested victim VM measured by CloudMonatt to detect these covert channels.

medium. A fast access time indicates the sender is sending a 0, whereas a slow access time indicates a 1. The figure also shows the receiver's observations when the sender VM runs a benign application (apache) with some noise.

A key idea to detect these covert channels is that programs involved in covert channel communications give unique patterns of the events happening on these hardware.[13] If a customer requests covert channel protection and periodic attestation as to whether potential covert communications are being sent out of his or her VM, CloudMonatt can use hardware performance counters to monitor the attested VM's memory bandwidth at every time interval (for example, 0.1 ms). After a certain monitoring period, CloudMonatt calculates the frequency distribution histogram for the memory bandwidth used. Specifically, it divides the entire range of observed bandwidth values into a number of bins (for example, 20 in our experiment) with equal size and then counts how many bandwidth values fall into each bin. Then CloudMonatt uses 20 trust evidence registers to store the number of values in each bin to represent the memory bandwidth distribution. These 20 values are sent as the security health measurements for detecting these LLC, bus, or DRAM covert channels. We use 20 bins in our experiment, but we could also use a different number to save space or increase accuracy.

When the attestation server receives the 20 values, the property interpretation module calculates the probability distribution (shown in Figure 2b) of the attested victim VM's memory bandwidth. If a covert channel exists, the distribution graph gives two peaks, each representing the activity of transmitting a 0 or a 1. On the contrary, a benign application with no covert communications tends to give multiple smaller peaks. The attestation server can use machine learning techniques to conduct pattern recognition of covert channels. Once the attestation server identifies the existence of attacks, it can perform further actions (for example, sending warnings to the customers and asking them to double-check malicious processes) to reduce false positives. This detection method might also introduce some false negatives because it cannot cover all covert-channel attacks. More

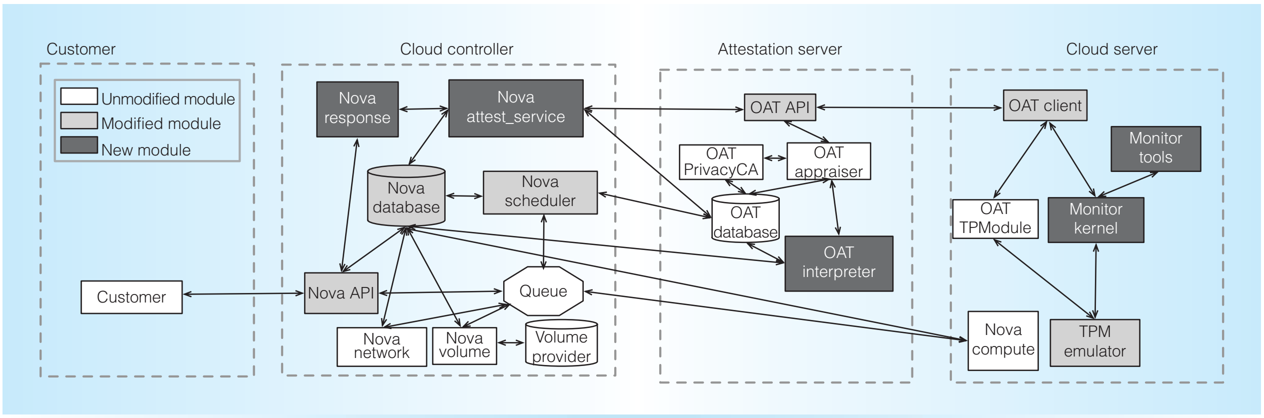

Figure 3. Implementation of attestation architecture. We implemented CloudMonatt on the OpenStack platform with the OpenAttestation and Trusted Platform Module (TPM) emulator software.

sophisticated detection methods can be integrated into CloudMonatt to detect other types of attacks.

Availability

Some denial-of-service attacks can significantly change the probability distribution of the victim VM's microarchitectural characteristics (for example, instructions executed per cycle or memory bandwidth). To detect if a VM is under such attacks, CloudMonatt can collect the empirical probability distribution of this VM in the trust evidence registers using the method described earlier, and then compare them with the expected typical distribution (which can be collected when the VM runs in a dedicated server, so there are no attacks). A significant difference between these two distributions indicates this VM's availability property is compromised, and corresponding responses can be triggered (for example, migrating it to another server).

Implementation

Figure 3 shows the implementation of our prototype for property-based cloud attestation on the OpenStack Havana platform. We integrated the OpenAttestation (OAT) software for host remote attestation protocols. We integrated the TPM-emulator and leveraged it to emulate the functions of the trust module in the hardware. Our evaluation shows that the emulation of the trust module has little impact on the system performance.

Cloud Controller

The cloud controller is implemented by the OpenStack Nova. We modified the Nova API to extend the VM launch command with the security properties the customers want for their VMs and added new commands for customers to monitor the VM's health. We modified the Nova database to store the customers' specifications of the security properties required for their VMs and the monitoring and attestation capabilities supported by each server. We modified the Nova scheduler to implement the policy validation module and deployment module in Figure 1a. We also added two new modules (shown in dark gray in Figure 3): one is the Nova attest service, which manages the attestation services. Another one is the Nova response, which provides some responses if the attestation fails.

Attestation Server

The attestation server and client are realized by OpenAttestation. We modified the OAT API to extend the APIs with more parameters, that is, security properties and VM ID. We added a new module OAT interpreter to interpret the VM's security health and make attestation decisions based on the information of the cloud server from the Nova database and the security measurements from the OAT database.

Cloud Servers

We modified the OAT client, the client side of OpenAttestation, to receive attestation

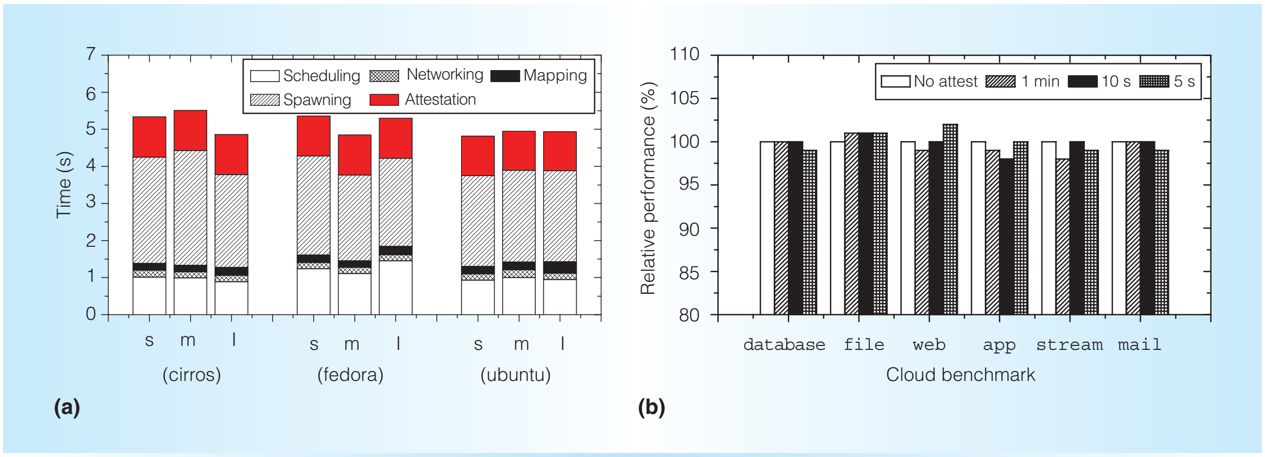

Figure 4. Performance for (a) VM launching and (b) runtime attestation. The attestation introduces about 20 percent performance overhead during VM launch and negligible overhead at VM runtime.

requests. We modified the TPM emulator to provide secure storage and crypto functions. We added two new modules: monitor kernel can start the security measurements and store the values in the TPM emulator, and monitor tools can integrate different software VMI tools, VMM profile tools, or other logging or provenance tools into the server to perform the monitoring and take measurements.

Evaluation

Our testbed includes three Dell PowerEdge R210II servers, each with a quad-core 3.30-GHz Intel Xeon processor, 32 Gbytes of RAM, and an on-board dual gigabit network adapter with a speed of 1 Gbit per second. We selected one server as the cloud controller, equipped with the Nova controller and OpenAttestation server. We implemented the other two servers as cloud server nodes.

Performance Evaluation

We consider two performance issues: the overhead of VM launching due to the new security requirements, and the overhead of attestation during runtime. We exploited the OpenStack Ceilometer for timing measurements. In the original OpenStack platform, the VM launch involves four stages:

1. Scheduling: allocate VMs to appropriate servers based on customer requirements and server workloads.
2. Networking: allocate the networks for VMs.
3. Block device mapping: set up block devices for VMs.
4. Spawning: start VMs on the selected servers.

CloudMonatt involves five steps for VM launching. At the scheduling stage, the controller needs to check the OAT database to find qualified servers with security features that support the customer's desired security properties. Steps 2, 3, and 4 are the same as the previous list. We add a fifth stage, attestation, which will check if the VM has been launched securely. Figure 4a shows the time for each stage of VM launching. We test three VM images (cirros, fedora, and ubuntu) with three VM flavors (small, medium, and large). The overhead of the attestation stage is about 20 percent, which is acceptable for VM launching. The main overhead of an attestation is for the message transmission across the network.

During VM runtime, customers can monitor the VM at any time, or periodically at a given frequency. To test the performance effect of periodic runtime attestation, we ran different cloud benchmarks in one VM while the customer issues the periodic runtime attestation request at different frequencies. Figure 4b shows the effect of periodic runtime attestation at frequencies of 1 minute, 10 seconds, and 5 seconds, on an ubuntu-large VM. This figure indicates that there is no performance degradation due to the execution of runtime attestation. This is for covert-channel monitoring, in which the measurements are taken from hardware performance counters. Whether runtime attestation causes performance degradation to the VM execution time depends on the measurement collection mechanism. However, if the periodic attestation frequency is low, then the performance effect is negligible.

Security Evaluation

The attestation architecture's trusted entities include the cloud controller and the attestation server. It is important that these machines have secure boot up and are secured for runtime protection. Traditional approaches can be taken to protect these servers, such as establishing firewalls, disabling VM launching on these central servers, and data hashing and encryption in the database. In addition, the trust module and monitor module in the cloud servers also need to be protected against software, OS, or physical attacks via existing protection mechanisms such as secure enclaves.[18,19]

We used ProVerif to verify the cryptographic protocol to ensure the customers can receive unforgeable attestation reports. For more information, see our previous work.[12]

CloudMonatt increases assurance in cloud systems by enabling secure monitoring and attestation of security features provided by a cloud server for the customer's VMs. It is the first concrete implementation of property-based attestation for security properties other than integrity checking, with examples for attesting different security health properties. We hope that CloudMonatt can lay the foundation for future work on investigating more security properties, how they can be implemented and monitored, and whether they can be integrated seamlessly into the CloudMonatt framework. Hardware and software platform vendors are encouraged to implement the architectural support identified by CloudMonatt to enable secure trust evidence monitoring and attestation in cloud servers. This can lead to significantly better security health in cloud computing environments. MICRO

Acknowledgments

This work was supported in part by the National Science Foundation Computer and Network Systems award no. 1218817.

References

1. P. Jamkhedar et al., "A Framework for Realizing Security on Demand in Cloud Computing," *Proc. IEEE 5th Conf. Cloud Computing Technology and Science*, 2013, pp. 371–378.
2. T. Ristenpart et al., "Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds," *Proc. 16th ACM Conf. Computer and Communications Security*, 2009, pp. 199–212.
3. Y. Zhang et al., "Cross-VM Side Channels and Their Use to Extract Private Keys," *Proc. ACM Conf. Computer and Communications Security*, 2012, pp. 281–292.
4. V. Varadarajan et al., "Resource-Freeing Attacks: Improve Your Cloud Performance (at Your Neighbor's Expense)," *Proc. ACM Conf. Computer and Communications Security*, 2012, pp. 281–292.
5. D. Perez-Botero, J. Szefer, and R.B. Lee, "Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers," *Proc. ACM Workshop Security in Cloud Computing*, 2013, pp. 3–10.
6. *TCG Design, Implementation, and Usage Principles*, ver. 3.0, Trusted Computing Group, 2011; www.trustedcomputinggroup.org/tcg-design-implementation-usage-principles-best-practices.
7. T. Garfinkel et al., "Terra: A Virtual Machine-Based Platform for Trusted Computing," *Proc. 19th ACM Symp. Operating Systems Principles*, 2003, pp. 193–206.
8. R. Sailer et al., "Design and Implementation of a TCG-Based Integrity Measurement Architecture," *USENIX Security Symp.*, 2004, pp. 223–238.
9. J. Schiffman et al., "Seeding Clouds with Trust Anchors," *Proc. ACM Workshop Cloud Computing Security*, 2010, pp. 43–46.
10. N. Santos et al., "Policy-Sealed Data: A New Abstraction for Building Trusted Cloud Services," *Proc. 21st USENIX Security Symp.*, 2012, article 10.
11. A.-R. Sadeghi, and C. Stüble, "Property-Based Attestation for Computing Platforms: Caring about Properties, Not Mechanisms,"

Proc. Workshop New Security Paradigms, 2004, pp. 67–77.

12. T. Zhang and R.B. Lee, "CloudMonatt: An Architecture for Security Health Monitoring and Attestation of Virtual Machines in Cloud Computing," *Proc. 42nd Ann. Int'l Symp. Computer Architecture*, 2015, pp. 362–374.
13. J. Chen and G. Venkataramani, "CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware," *Proc. 47th Ann. IEEE/ACM Int'l Symp. Microarchitecture*, 2014, pp. 216–228.
14. R.A. Kemmerer, "Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels," *ACM Trans. Computer Systems*, 1983, pp. 256–277.
15. V.D. Gligor, *A Guide to Understanding Covert Channel Analysis of Trusted Systems*, report NCSC-TG-030, Nat'l Computer Security Center, 1994.
16. Y. Xu et al., "An Exploration of L2 Cache Covert Channels in Virtualized Environments," *Proc. 3rd ACM Workshop Cloud Computing Security*, 2011, pp. 29–40.
17. Z. Wu, Z. Xu, and H. Wang, "Whispers in the Hyper-Space: High-Speed Covert Channel Attacks in the Cloud," *Proc. 21st USENIX Security Symp.*, 2012, article 9.
18. D. Champagne and R.B. Lee, "Scalable Architectural Support for Trusted Software," *Proc. 16th Int'l Symp. High Performance Computer Architecture*, 2010; doi:10.1109/HPCA.2010.5416657.
19. F. McKeen et al., "Innovative Instructions and Software Model for Isolated Execution," *Proc. 2nd Int'l Workshop Hardware and Architectural Support for Security and Privacy*, 2013, article 10.

Tianwei Zhang is a PhD candidate in the Department of Electrical Engineering at Princeton University. His research interests include cloud computing, security detection, and mitigation. Zhang received an MA in electrical engineering from Princeton University. Contact him at tianweiz@princeton.edu.

Ruby B. Lee is the Forest G. Hamrick Professor in the Department of Electrical Engineering at Princeton University. Her research interests include security-aware computer architecture, secure cloud computing, smartphone security, side channels, improving security without sacrificing performance, and security validation. Lee received a PhD in electrical engineering from Stanford University. She is a Fellow of IEEE and ACM and is on the advisory board of *IEEE Micro*. Contact her at rblee@princeton.edu.

cn *Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.*

IEEE micro

Call for Papers | General Interest

IEEE Micro seeks general-interest submissions for publication in upcoming issues. These works should discuss the design, performance, or application of microcomputer and microprocessor systems. Summaries of work in progress and descriptions of recently completed works are most welcome, as are tutorials. *IEEE Micro* does not accept previously published material.

www.computer.org/micro