Auditing Cloud Administrators Using Information Flow Tracking

Afshar Ganjali
a.ganjali@utoronto.ca

David Lie
lie@eecg.utoronto.ca

Department of Electrical and Computer Engineering
University of Toronto

ABSTRACT

In the last few years, cloud computing has evolved from being a promising business concept to one of the fastest growing segments of the IT industry. However, one impediment to widespread adoption by enterprise customers is the threat of attack by a malicious cloud administrator. To address this security and privacy challenge, we propose *H-one*, a new auditing mechanism for cloud. *H-one* uses information flow tracking techniques to implement complete, efficient and privacy-preserving logs that will enable the auditing of the administrators of the cloud infrastructure, thus increasing the customer's trust in cloud services.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Information flow controls*; K.6 [**Computing Milieux**]: Security and Protection

Keywords

Cloud Computing, Information Flow Tracking, Security, Privacy

1. INTRODUCTION

Infrastructure-as-a-Service (IaaS) clouds provide a service where users can rent virtual machines (VMs) from a cloud provider instead of having to own the hardware themselves. Such a model reduces the capital costs of deploying compute infrastructure and allows for much needed elasticity; allowing users to only pay for the compute cycles they need and allowing cloud providers to easily shift hardware resources between customers. However, despite these apparent advantages, many companies and organizations have been reluctant to embrace cloud computing for their compute infrastructure. An IDG survey of industry leading CIOs and information-technology decision makers in 2011 and 2012, showed that security was the top concern for using cloud computing in both years [5]. As a result, a key advantage for the cloud provider will be to provide technologies that instill confidence in their customers that VMs running on the cloud provider's infrastructure are safe from tampering and data leakage.

One of the new threat vectors that IaaS clouds introduce is exposure of customer VMs and their data to the system administrators of the cloud provider. Because the cloud administrators must have privileges to access all aspects of the cloud infrastructure, they pose a significant threat to the security of any VM running in an IaaS cloud. With root privileges on the hypervisor, the administrator has the ability to install and execute arbitrary software. For example, in the Xen hypervisor, tools like LibVMI [2] allow an administrator to observe the contents of a VM's memory at run time. While there have been proposals to attest and protect the integrity of the hypervisor [16, 17, 24], attestation of the large, privileged management stack above the hypervisor is impractical because of the diversity of legitimate software and configurations that can exist in the management stack.

To protect customer VMs from tampering by a malicious administrator, the cloud provider must guarantee that an administrator does not abuse their ability to access the private memory, disk or processor state of a VM, regardless of whether it is running or suspended to disk [16]. One way to achieve this is to attenuate the administrator's privileges. For example, the cloud provider might disable shell access for its administrators and only permit them to perform tasks through a limited web interface such as OpenStack [3] or a management client such as VMware's vSphere. However, removing the privileges of the administrator ultimately makes them less efficient and impedes their ability to manage the cloud infrastructure. For example, the limited interface prevents the administrator from using full-featured scripting languages such as Perl or Python to automate her tasks. Administrators benefit from being able to use commodity software, but due to the complexity of commodity software stacks, correctly implementing the fine line between removing unnecessary privileges and keeping the useful privileges for an administrator is very challenging.

In this paper, we advocate a fundamentally different approach. Rather than attenuating the capabilities of the administrator to prevent abuse, we use secure auditing to record any unauthorized use of administrator privileges. Auditing has been long employed with great success in other domains such as banking, taxation and public service. Instead of enforcing correct behavior through restrictions, auditing deters individuals from misbehaving by creating an indisputable log of their actions. In a cloud environment,

STC'12, October 15, 2012, Raleigh, North Carolina, USA.

auditing will also enable the correct assigning responsibility, which is crucial in the enforcement of the service level agreements (SLA). We aim to use a similar approach for logging data manipulation by administrators. By just logging events we do not hinder a legitimate administrator from fulfilling his duties, but at the same time are able to deter malicious administrators from attacking customer VMs.

Our proposal, called *H-one* has three main goals. First, the audit log produced by H-one should be *complete*, i.e., enough information must be recorded during the audit process to reconstruct what information was read from the customer's VM or what modifications were made to the customer VM by the administrator. Second, the recording of the audit log must be *efficient* – imposing minimal performance and storage overheads on the cloud provider. Finally, audit logs will be made available to the customer and in the event of a dispute, be broadly shared with arbitrators or to courts in the legal system. As a result, it is important that an audit log for a VM be *privacy-preserving* and thus contain only the information pertinent to that VM.

To achieve these goals, H-one uses information flow tracking. Having the entire path gives H-one several options to try and impose minimal performance and storage overhead for logging. For example, for an administrator performing a backup of a VM image, ultimately, the administrator may only need to see the total number of bytes archived by the backup operations. By identifying the entire flow of information, rather than recording the entire backup operation, which would touch every bit in the VM image, we only need to record the information that finally flowed to the administrator. Similarly, if an administrator injects data into a VM via a malicious virtual CD-ROM image, having knowledge of the entire information flow path allows us to record only the bytes read by the VM rather than the entire contents of the CD-ROM. In both examples, information flow also ensures that separate audit logs can be maintained for each customer. Even if the administrator backs up the VMs of several customers, information flow allows H-one to tear apart the flows of data into separate audit logs for each customer.

2. ATTACK MODEL AND PROBLEM DESCRIPTION

Our goal is to log all flows of information both from an administrator to a customer VM and vice versa. Recording information flows from VMs to administrators will capture violations of VM confidentiality. For example, a misbehaving administrator on a Xen hypervisor can use various existing tools like LibVMI [2] or VIX [14] for introspecting VM's memory, CPU, disk and network to reach user's private information. However, completely preventing any information flow from VMs to administrators is not desirable because administrators may have legitimate reasons for such flows. For example an administrator may have to observe VM's disk for backup purposes or its memory for virus scanning. Another common example is that sometimes administrators are supposed to help customers for troubleshooting and need to log into the user's VM. All these examples highlight a legitimate flow of information from VMs to administrators.

We also have to track all the information flows from administrators to VMs to assure the integrity of the guest VMs. As an example, most of the existing cloud management tool stacks provide APIs for attaching a CD-ROM to a user VM by administrators. A malicious administrator can abuse this API by inserting a malicious CD and rebooting the system, causing it to boot from the CD-ROM and allowing the administrator to then access the VM's disk using the software on the CD-ROM. Auditing flows from administrators to VMs can deter such malicious actions. However, not every flow from administrators to VMs are malicious. For example, to resolve customer issues, it is not uncommon for an administrator to ping a customer's VM or even log into the customer's VM to make changes.

We assume that the administrator has the equivalent of root or superuser access on the management stack of the hypervisor. Examples of such access would be root in Domain 0 of Xen, the host OS in KVM or SSH access to an ESX server. As a result, the administrator is able to install and execute arbitrary user code in the management stack, as well as change the configuration of any of the running services. Also the underlying network layer is in control of the administrator and any kind of traditional network attacks have to be expected.

We assume that the hypervisor and management stack kernel are protected from tampering by the administrator. There has been several works in the literature that deal with integrity and attestation of the hypervisor [16,17] and we assume a similar mechanism in our system has been used to secure those components. Unfortunately, because of the variety of configurations and software that must run above the management stack kernel, it is not realistic to extend attestation above that of the kernel. We also assume that some access control policy prevents administrators from corrupting the kernel by loading unauthorized modules, tampering with the kernel binary or directly accessing kernel memory. A Linux Security Module (LSM) or an enhancement to POSIX capabilities may be ways to achieve this.

We also assume physical security of the hardware running the hypervisor, so that hypervisor and management stack kernel are protected from tampering through hardware. This is fairly straightforward to fulfill since most administrative tasks do not require physical access to machines. In addition, methodologies for securing physical machines using cages, locked doors and security cameras are fairly well understood and can be transferred over from a variety of other areas. Hence, any physical attack such as cold boot attacks are beyond the scope of this paper.

Another attack vector on guest VMs is covert channels. Data from a customer VM may be leaked indirectly through a covert channel or a side-channel. Such channels may exist either because our information flow tracking is incomplete and misses certain flows, or information is steganographically encoded by a compromised process so that the user does not realize information has been leaked during an audit even though the flow is caught and recorded by our system. We currently consider such attacks against our system out of scope.

H-one's goal is to produce audit logs that will capture the tampering of customer VMs by a malicious administrator. This is an orthogonal and complementary goal to systems that try to protect VMs against vulnerabilities in the hypervisor or management stack [8,24]. These systems protect against vulnerabilities by reducing the size of the TCB on which the security of customer VMs depend, while H-one only requires that the logging infrastructure be protected

from an administrator. As a result, so long as the logging infrastructure is secure, H-one is agnostic to the size of the TCB for customer VMs.

3. USING INFORMATION FLOW

We expect H-one to be applicable to any hypervisor architecture, but for convenience will describe a possible implementation on the Xen hypervisor. We assume an *administrative VM* (domain 0 in Xen), which has special privileges and implements a management stack that contains a commodity OS. Operations in the management stack, such as VM creation, require the management stack to directly access the state of a VM [13]. The administrator has root privileges on the management stack and thus has the ability to use all privileges conferred on the administrative VM.

As we mentioned earlier, logging infrastructure must be protected from tampering (i.e. it must be self-protecting). H-one implements logging in the kernel of the management stack. Thus, to protect the logging infrastructure, H-one needs only to protect the integrity of the kernel, which can be protected by restricting the administrator from tampering with the kernel image or loading unauthorized modules. This can be achieved by using Linux Security Modules (LSM) [20] and is the only attenuation of privileges that H-one imposes on the administrator.

Our solution has two components. The first is the ability to track all flows of information emanating from both a customer VM and an administrator. It is important that this tracking is complete; otherwise a malicious administrator will be able to steal information from a VM or tamper with its state without being logged. By tracking all flows, we are able to detect paths of "tainted" processes in the management stack along which information flowed. The second component is selecting the best point along these tainted paths to log the information. Since the goal is to log only the information that is finally consumed, this implies that the best logging point will usually be as close to the consumer of the information as possible, given that logging at that point does not incur too high of a performance penalty. In another words, information tainted with the VM should be recorded as close to the administrator, and information tainted with the administrator should be recorded as close to the VM as possible.

3.1 Tracking All Information Flows

When information enters the management stack, it must be "seeded" with a label for tracking purposes. If the information comes from a VM then H-one seeds it with a label corresponding to the customer who owns the VM and if the information comes from an administrator, H-one seeds it with a label corresponding to the identity of the administrator.

Information from a customer VM can enter the management stack either through direct interaction with components in the management stack that virtualize hardware for guest VMs, or through indirect interaction via the hypercalls made by the management stack. As a result, the seeding of labels is performed by both the hypervisor and management stack kernel. If information returned from a hypercall is derived from a guest VM, the hypervisor passes this information to the management stack kernel, which then labels the return value as containing information from the particular VM.

Xen implements virtual hardware using either a backend driver for paravirtualized guest VMs or hardware QEMU emulation for fully virtualized guests. In both these cases, the virtual hardware emulation is performed in the management stack and the management stack kernel is responsible for labeling information from customer VMs and tracking the information flow appropriately. A typical example is a write to a virtual disk whose image is stored in the management stack. The backend driver or QEMU would be seeded with the customer VM's label and would thus taint the disk image. Any process that then accesses the disk image will also be tainted with the customer VM's label.

Administrator information enters the management stack through a terminal process. To properly label terminal processes with the identity of the administrator that controls them, we modify remote terminal services, such as the SSH daemon in the management stack to seed the child terminal process with the label of the administrator that authenticated and logged in. Any child processes started by the administrator's shell are also seeded with the administrator's label.

After seeding initial labels, we have to track propagation of taint. Again we handle this in management stack's kernel. Table 1 lists various propagation channels that we need to consider for applying taint analysis. We have to address two issues: i) How to intercept all these channels of communication? ii) How to propagate labels?

Three of these taint propagation paths can be caught by intercepting hypercalls in the management stack kernel. However, other paths are entirely within the management stack and have to be intercepted using some hooks in the kernel. We are considering using a customized LSM to monitor and interpose on the operations that propagate taint within the management stack kernel.

As Table 1 indicates, whenever a process in the management stack kernel maps memory belonging to a VM, we conservatively assume that the process will read management stack memory and propagate taint from the VM to the process. Even though the mapping operation can be intercepted by monitoring hypercalls in the management stack kernel, we are more precise for propagating administrator taint from a process to a VM since such a flow requires logging. To detect flows from a process to a VM via shared memory, we can map the shared region as read-only inside the process's address space, causing the management stack kernel to take a fault if the process writes to the region. This is similar to how the HyperLock project [19] prevents the KVM hypervisor from writing into any domain other than its own or how the SecVisor project [18] prevents the kernel from executing user pages.

For propagating taint, when a communication event happens, we propagate the labels from the source to the destination, if it is a read operation; or the other way around, if it is a write operation.

3.2 Selecting the Best Point to Log

In order to reduce the storage cost of logging, H-one attempts to log as close as possible to the consumer of the data (i.e. the administrator or the VM). For finding the points closest to the administrator, we have to detect when VM-tainted information is about to leave the management stack kernel through network or any other I/O channel. This is observable using LSM I/O hooks when a process writes to

Channel	Flow	Interception Mechanism	Label Propagation
Mapping a guest VM's memory image.	VM → Process	Intercepting Hypercalls	VM → Process
Reading VCPU state of a guest VM.	VM → Process	Intercepting Hypercalls	VM → Process
Reading an arbitrary file.	File → Process	LSM Filesystem Hooks	File → Process
Inter Process Communication (IPC)	Process → Process	LSM IPC Hooks	Process → Process
Writing into memory image of a guest VM.	Process → VM	Paging Mechanism	Process → VM Should be logged.
Writing VCPU state of a guest VM.	Process → VM	Intercepting Hypercalls	Process → VM Should be logged.
Writing into an arbitrary file.	Process → File	LSM Filesystem Hooks	Process → File May need to be logged.
Writing to network.	Process → Network	LSM Network Hooks	Process → Exporter May need to be logged.
Writing to an output device.	Process → Admin	LSM I/O Hooks	Should be logged.

Table 1: List of taint propagation channels; how they get intercepted and how the labeling gets effected.

network or any other I/O device. In these cases we have to record all the exchanged information for auditing purposes.

The other constraint for selecting the point at which to log is that H-one must be able to determine which VM the information pertains to. Thus, if a process reads information from several VMs, H-one must log before the information is read and mixed together by the process. Similarly, if an administrator uses a single process to administer several VMs, to be able to unambiguously determine which administrator actions pertain to which VMs, H-one must log the outputs of the program to the individual VMs as opposed to the inputs the administrator sends to the process.

Capturing data at the network interface is not effective if the process, such as an SSH shell, encrypts the data before sending it over the network. Since we already modify SSH for seeding labels, we can have the modified SSH log unencrypted data before transmission. However, administrators may install their own application that encrypts data before writing to a network socket. To address this case, we plan to implement exporter daemons similar to those in DStar [23], which will label outgoing network data. The remote host will then assign those labels to the received network data and continue to track the information until it is consumed by the administrator's terminal. We will use DStar-like access control to restrict network connections by administrator-installed programs only to verified hosts that also implement H-one logging.

For finding points closest to VM, we have to detect when information is about to be written to the VM. When a process writes into the memory image, VCPU state or disk image of a VM, it should be considered as consumption of information by the guest VM. In these cases all the exchanged information has to be recorded. For example when an administrator attaches a virtual CD-ROM, it will be visible to the VM as a block device. Communication between block device and VM is over shared memory, which is expensive to monitor. Instead, we can monitor what bytes are read from the CD-ROM image by the QEMU device emulator that the back end driver communicates with. Note that in this case, QEMU would be part of the logging infrastructure so we must protect it to make sure it is not modified. Otherwise, a misbehaving administrator could modify QEMU to directly write malicious data into VM instead of reading it from the CD-ROM image.

4. DISCUSSION

4.1 Auditing Instead of Restricting

One of the key differences between H-one and previous approaches is that H-one performs no access control and provides no active protection to customer VMs. Instead, H-one provides tamper-resistant, complete and efficient logging of cloud administrator activities. The advantage of auditing is that since it is passive, it guarantees that administrators will not be unnecessarily restricted in any way. For instance, we could prevent tainted processes from generating logs to be seen by administrators to avoid privacy violations. However, sometimes specific kinds of data need to be passed to an administrator for specific purposes. For example, data to be used in log files or some information provided by daemons inside guest VMs for the purpose of monitoring, measuring or provisioning are legitimate types of data that we can pass to administrators. Restricting all these channels can guarantee privacy for the user but conflicts with our goal of not to inhibit administrative duties. Since appropriate security policies in such cases are context dependent, applying a policy that restricts administrators will almost inevitably lead to a situation where an administrator is prevented from performing some legitimate action. In our approach we just audit all the classified information passed to administrators and defer the further decisions to the user, who owns the data and is aware of the consequences of such leakage.

Our approach for letting administrators get access to classified data is equivalent to declassification techniques used in DStar [23] and other information flow control systems. In our system, any process can declassify information and pass it to an entity without required clearance level. By just auditing all the channels, we do not have to deal with the problem of identifying processes that can be trusted as declassifiers, which is context dependent.

4.2 Realtime Filtering of Logs

Using information flow techniques, we can save extensively on the amount of data we log, which reduces cost at the end. For further reduction of the log size, we are considering implementing a realtime filtering daemon that filters some unnecessary logs before committing to disk and the end user. For example, in our information flow system, we intercept all the data written to the memory image of a guest VM. However, operations like VM creation have to map the whole memory address space of the VM in the beginning to copy the memory image from disk to VM's memory before

executing the VM. Since VM creation happens though user space management processes, our system would intercept all those operations and would log the whole memory image as exchanged data between an administrator and the guest VM. By knowing beforehand that this type of information exchange is legitimate, our filtering system can reduce the logs by simply comparing the data written to the memory with the existing memory image file on the disk. If those sets of data are exactly the same, we can easily skip the logging and trust the operation. Otherwise, we log the data and notify the user. We believe there are other scenarios that we can predict beforehand and our filtering system can use them to narrow down the illegitimate data.

5. RELATED WORK

Trusted Cloud Computing. Over the past several years, there has been a lot of work on trusted cloud computing. The management stack has traditionally been a monolithic privileged virtual machine. Different projects such as Qubes OS [4], Citrix XenClient [1], Xen Disaggregation [13] and Xoar [8] have reduced the TCB size of Xen by breaking the management stack into smaller isolated components. However, all these solutions require a complete reorganization and significant changes in the management stack architecture.

Santos et al. [16] propose a trusted cloud architecture to protect the confidentiality and integrity of customer VMs. Li et al.'s paper [11], CloudVisor [24], and Excalibur [17] are more recent efforts that share the high-level vision proposed in that work. CloudVisor uses nested virtualization to protect guest VMs from malicious hypervisor and management stack. Excalibur proposes a system that implements policy-sealed data to address some of the limitations of trusted computing on the cloud. However, we are not aware of any previous work using information flow for providing more security at cloud computing environments.

Information Flow Control. Prior work on information flow control can be broadly categorized into static language-based techniques [15], which seek to detect and prevent information leakage at compile time, and dynamic runtime enforcement. Asbestos [9] and HiStar [22] are two example operating systems that track information flow dynamically using a relatively small, trusted kernel.

Recently, there has been significant work on incorporating DIFC (Decentralized Information Flow Control) mechanisms into operating systems and runtime environments with the goal of enforcing end-to-end information flow policies that are set by individual applications instead of a central administrator. As an example, DStar [23] extends the OS-level information flow control architecture to a distributed environment with the goal of mitigating the effects of untrustworthy distributed applications and compromised machines. It uses "exporters" to map machine-level security policies into a distributed context, allowing hosts to define the degree of trust between host nodes. The dynamic enforcement in DStar and in earlier OS-level systems tracks information flows at a process granularity, which often requires monolithic applications to be split into several processes to make information flows within an application explicit to the OS. Flume [10], one of the more recent efforts, demonstrates that runtime DIFC does not require a clean-slate redesign of the software stack and can be retrofitted into an existing UNIX-based operating system.

Some other works [6] employ IFC for providing security in other contexts. SilverLine [12] is another system that uses IFC to improve both data and network isolation for cloud-based services without requiring cloud application developers to rewrite their applications, and without requiring the deployment of specialized network hardware.

6. CONCLUSION AND FUTURE WORK

One of the biggest obstacles in the way of the adoption of cloud-based services is the risk that data stored in the cloud will be compromised by a misbehaving administrator. To deter such attacks, we proposed H-one, an auditing mechanism for cloud that logs information leakage by tracking data using information flow tracking techniques. By using auditing, H-one allows administrators the freedom to use whatever tools they wish to perform and automate administrative tasks.

Our next steps would be to have a fully implemented prototype to evaluate the performance overhead and log storage costs of H-one. We aim to use the Xen [7] hypervisor for implementation. As we mentioned in Section 3.1 we have to implement some hooks in the management stack kernel for intercepting sensitive operation done by administrators and for protecting the log system. For these cases we are planning to employ a customized Linux Security Module (LSM) module. LSM provides a wide range of hooks in the Linux kernel, which can be used to monitor and interpose on the operations that propagate taint within the management stack kernel. LSM is lightweight and designed to impose little performance overhead. We plan to also employ the LSM hooks both to prevent administrator tampering of the management stack kernel and H-one logging infrastructure and to track information flows in the management stack kernel. For assuring completeness of this method, previous study [21] has used automatic static analysis of the kernel code to verify that all of the necessary hooks have actually been inserted into the Linux kernel.

Another challenge will be to ensure that the logging is complete enough for a third party to reconstruct the actual events that led to the log being produced and make an unequivocal judgment as to whether the administrator acted maliciously or not. As an example, assume a user receiving logs related to her VM's execution on the cloud. She can check the logs for various types of violations. For instance, since mapping of VM memory should only occur on guest VM boot, she can check if the mapping of her VM's memory to the management stack has a one-to-one relation with VM starts happened by her. If data has been read from the VM's file system or from a certain process, she can determine which files or process were accessed. If the administrator installs or runs some code on the VM, she can determine the binary of the code and as a result, possibly run the code again to ascertain its purpose or identity. While it will be difficult to prove that recorded logs will be able to result in an unambiguous determination in every instance, we hope to show that for a large number of scenarios, the logs will allow an impartial third party to prove the guilt or innocence of the cloud administrator.

Acknowledgements

We thank the anonymous reviewers for their helpful comments. Funding for this work was provided by the NSERC ISSNet Strategic Network, an NSERC Discovery Grant, an MRI Early Researcher Grant and a gift from AT&T.

7. REFERENCES

[1] Citrix XenClient. http://www.citrix.com/xenclient.

[2] LibVMI. http://code.google.com/p/vmitools/.

[3] OpenStack: Open source cloud computing software. http://www.openstack.org/.

[4] QubeOS. http://qubes-os.org/Home.html.

[5] IDG Enterprise - Cloud Computing. http://www.idgenterprise.com/report/idg-enterprises-cloud-computing, Apr. 2012.

[6] Abadi, M., Birrell, A., and Wobber, T. Access control in a world of software diversity. In *Proceedings of the USENIX Workshop on Hot Topics in Operating Systems (HotOS)* (2005).

[7] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, USA, 2003), ACM, pp. 164–177.

[8] Colp, P., Nanavati, M., Zhu, J., Aiello, W., Coker, G., Deegan, T., Loscocco, P., and Warfield, A. Breaking up is hard to do: Security and functionality in a commodity hypervisor. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)* (2011), pp. 189–202.

[9] Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazieres, D., Kaashoek, F., and Morris, R. Labels and event processes in the Asbestos operating system. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)* (New York, NY, USA, 2005), ACM, pp. 17–30.

[10] Krohn, M., Yip, A., Brodsky, M., Cliffer, N., Kaashoek, M. F., Kohler, E., and Morris, R. Information flow control for standard OS abstractions. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)* (Oct. 2007), pp. 321–334.

[11] Li, C., Raghunathan, A., and Jha, N. K. Secure virtual machine execution under an untrusted management OS. In *Proceedings of the IEEE 3rd International Conference on Cloud Computing* (2010), pp. 172–179.

[12] Mundada, Y., Ramachandran, A., and Feamster, N. SilverLine: Data and network isolation for cloud services. In *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)* (2011).

[13] Murray, D. G., Milos, G., and Hand, S. Improving Xen security through disaggregation. In *Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)* (Seattle, WA, USA, 2008), pp. 151–160.

[14] Nance, K., Bishop, M., and Hay, B. Virtual machine introspection: Observation or interference? *IEEE Security & Privacy Magazine 6*, 5 (Sept. 2008), 32–37.

[15] Sabelfeld, A., and Myers, A. C. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications 21*, 1 (2003), 5–19.

[16] Santos, N., Gummadi, K. P., and Rodrigues, R. Towards trusted cloud computing. In *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)* (2009), USENIX Association.

[17] Santos, N., Rodrigues, R., Gummadi, K. P., and Saroiu, S. Policy-Sealed data: A new abstraction for building trusted cloud services. In *Proceedings of the USENIX Security Symposium* (2012).

[18] Seshadri, A., Luk, M., Qu, N., and Perrig, A. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *Proceedings of ACM SIGOPS Symposium on Operating Systems Principles* (Stevenson, Washington, USA, 2007), ACM, pp. 335–350.

[19] Wang, Z., Wu, C., Grace, M., and Jiang, X. Isolating commodity hosted hypervisors with HyperLock. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)* (New York, NY, USA, 2012), ACM, pp. 127–140.

[20] Wright, C., Cowan, C., Smalley, S., Morris, J., and Kroah-Hartman, G. Linux security modules: General security support for the linux kernel. In *Proceedings of the USENIX Security Symposium* (2002), pp. 17–31.

[21] Xiaolan Zhang, Antony Edwards, and Trent Jaeger. Using CQUAL for static analysis of authorization hook placement. In *Proceedings of the USENIX Security Symposium* (2002).

[22] Zeldovich, N., Boyd-Wickizer, S., Kohler, E., and Mazieres, D. Making information flow explicit in HiStar. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2006).

[23] Zeldovich, N., Boyd-Wickizer, S., and Mazieres, D. Securing distributed systems with information flow control. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2008), pp. 293–308.

[24] Zhang, F., Chen, J., Chen, H., and Zang, B. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of ACM SIGOPS Symposium on Operating Systems Principles (SOSP)* (2011).