Data Protection in OpenStack

Bruce Benjamin, Joel Coffman, Hadi Esiely-Barrera, Kaitlin Farr, Dane Fichter, Daniel Genin,
Laura Glendenning, Peter Hamilton, Shaku Harshavardhana, Rosalind Hom, Brianna Poulos, Nathan Reller
The Johns Hopkins University Applied Physics Laboratory
Laurel, MD
Email: {*forename.surname*}@jhuapl.edu

***Abstract*—As cloud computing becomes increasingly pervasive, it is critical for cloud providers to support basic security controls. Although major cloud providers tout such features, relatively little is known in many cases about their design and implementation. In this paper, we describe several security features in OpenStack, a widely-used, open source cloud computing platform. Our contributions to OpenStack range from key management and storage encryption to guaranteeing the integrity of virtual machine (VM) images prior to boot. We describe the design and implementation of these features in detail and provide a security analysis that enumerates the threats that each mitigates. Our performance evaluation shows that these security features have an acceptable cost—in some cases, within the measurement error observed in an operational cloud deployment. Finally, we highlight lessons learned from our real-world development experiences from contributing these features to OpenStack as a way to encourage others to transition their research into practice.**

I. Introduction

Cloud computing is increasingly prevalent as organizations look to slash IT costs and to simplify management of IT resources. Many organizations also expect cloud computing to increase productivity and give them a competitive market advantage. Despite these benefits, security and privacy concerns remain major inhibitors to adoption for more than 60% of companies [1]. One recent study claims that as few as 1 in 10 cloud providers use even basic security features such as encrypting data at rest [2].

In this paper, we provide a unique vantage point by describing the development of several major security features in OpenStack and lessons learned throughout this process. OpenStack is an open source Infrastructure as a Service (IaaS) cloud platform used by hundreds of companies. We have contributed a number of security features to OpenStack, including block storage encryption, verification of signed images, and key management. Block storage encryption secures data at rest in OpenStack, both for ephemeral storage that is local to a VM and for persistent volumes that may be shared among VMs. Signed image verification provides integrity protection for bootable VM images. Key management capabilities support the credentials necessary to accomplish these features securely.

The contributions of this paper are twofold. First, we describe several important data protection features available within OpenStack, including key management, block storage encryption, and verification of signed images. Second, we highlight the lessons learned from this experience. We expect this paper to facilitate discussion on effectively transitioning more research activities into operational products and to help other cloud researchers better understand this process.

The remainder of this paper is organized as follows. Section II provides background material—viz. an overview of OpenStack's services and contribution process. Section III describes our security features with a security analysis of the features following in Section IV. We evaluate the performance of those features in Section V. Section VI enumerates lessons learned throughout our work. Section VII reviews related work. Finally, we conclude in Section VIII.

II. Background

OpenStack was initiated in July 2010 as a collaboration between NASA and Rackspace. OpenStack's mission is "to produce the ubiquitous open source cloud platform that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable."[1] It is currently used by hundreds of companies worldwide thanks to its permissive Apache 2 license.

A. OpenStack Services

The basic architecture of OpenStack comprises many loosely-coupled services that support virtualized compute, storage, and network resources. OpenStack's services have flexible representational state transfer (REST) application programming interfaces (APIs) and plug-in configurations to support a variety of different product offerings, including those from commercial vendors.

The following list identifies the major OpenStack services (each service's codename appears in parentheses) and provides a brief description of the service.

Block Storage (Cinder) Cinder supports user-defined provisioning of block storage, typically based on commercial network-attached storage (NAS) or storage area network (SAN) products.

Compute (Nova) Nova provides on-demand, self-service provisioning and management of VMs. Many hypervisors are supported, including KVM, Xen, Hyper-V, and vSphere.

Dashboard (Horizon) Horizon provides a web interface to various features within OpenStack such as uploading VM images and booting a VM.

Identity (Keystone) Keystone provides client authentication, distributed authorization, and discovery of other OpenStack services.

[1] https://wiki.openstack.org/wiki/Main_Page

2159-6190/17 $31.00 © 2017 IEEE
DOI 10.1109/CLOUD.2017.77

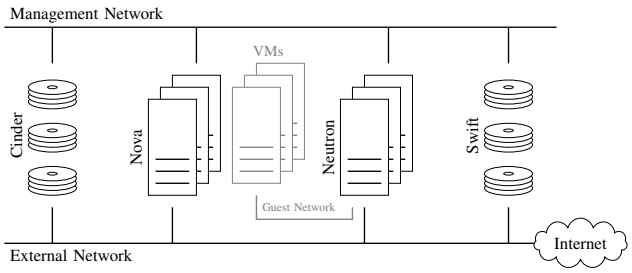

Fig. 1. Illustration of a subset of OpenStack's services. VMs are managed by Nova with Neutron providing network connectivity. Cinder and Swift provide persistent storage in the cloud.

Image (Glance) Glance allows users to upload VM images and to discover existing images for reuse.

Key Management (Barbican) Barbican securely stores, provisions, and manages various types of encryption keys and other private data.

Networking (Neutron) Neutron supports network connectivity for other OpenStack services (e.g., VMs managed by Nova) via a pluggable architecture that supports existing hardware and software-defined networking (SDN).

Object Store (Swift) Swift provides scalable, efficient, and redundant object storage.

Figure 1 provides a high-level depiction of several services previously described—in this case, those providing compute, networking, and storage resources.

B. Contribution Process

OpenStack follows a structured process for contributions as illustrated in Figure 2. The semi-annual OpenStack Summit initiates the development cycle and facilitates design discussions among contributors. Blueprints provide a high-level description of new features and track their development. Specifications add design information and technical details, allowing OpenStack's core reviewers (i.e., project maintainers) to provide direction prior to implementation. Merging code changes requires acceptance from two core reviewers in addition to passing automated unit and integration tests.

OpenStack requires a high level of commitment to its community, to support both one's own work and that of others. This involvement goes beyond the posting of blueprints, specifications, and code as previously mentioned. Participation also includes activities such as submitting bug reports, peer review of others' work (e.g., code reviews), and active participation at OpenStack events. Lastly, documentation must be written to reflect how to use new features, both from an end user's perspective and from a system administrator's perspective. This documentation step is critical if new features are to be used and maintained by the OpenStack community.

III. Security Features

This section describes three major security features in OpenStack. First, we describe OpenStack's key management service, and our related contributions to support Key Management

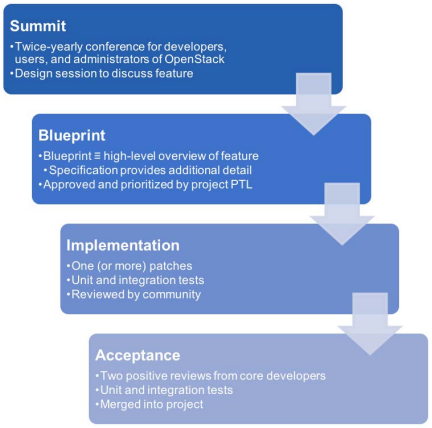

Fig. 2. Illustration of OpenStack's contribution process for new features.

Interoperability Protocol (KMIP)-compliant appliances for key storage. Second, we describe our block storage encryption feature, which provides confidentiality for data at rest and data in transit. Third, we describe our approach to image integrity that ensures VM images are not modified prior to boot. We provide a security analysis of our block storage encryption and image integrity features in the following section.

A. Key Management

Effective data protection requires strong cryptographic keys that are stored securely. A secure key manager is a critical part of cloud security. Key managers have a number of benefits, including

- the generation of cryptographically strong keys;
- performing cryptographic operations (e.g., signing) of data without the key leaving the device;
- centralization of sensitive data, which includes passphrases, keys, and certificates; and
- logging access information, creating an audit trail for administrators, compliance, and, when necessary, forensic investigations.

Hardened key management appliances include features like anti-tampering—if the device detects that someone is attempting to access the key storage, then all the data will be wiped.

OpenStack's key management service, Barbican, includes a REST interface for managing secrets, including symmetric

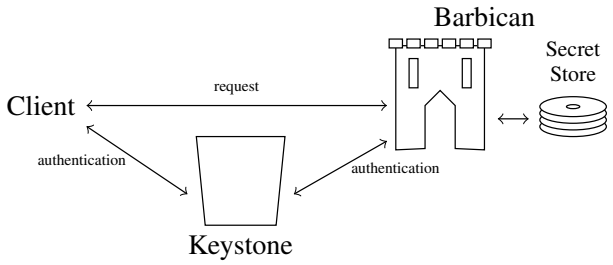

Fig. 3. Illustration of Barbican's components and handling of requests.

keys, asymmetric key pairs, X.509 certificates, passphrases, and opaque data objects. In addition, many features are supported such as quotas, user-defined metadata, and access control lists. Barbican has a plug-in storage system (the "secret store" in Figure 3) that we championed, allowing an administrator to configure one of several storage plug-ins, including KMIP or PKCS #11-compliant hardware security modules (HSMs).

OpenStack services integrate with Barbican via a simple Python interface. This interface abstracts the underlying key manager and enables interaction with existing enterprise key management appliances (e.g., in a private cloud deployment).

Our core contribution to Barbican was support for KMIP [3], a communication protocol defining how to encode key management messages and object data for exchange. The KMIP standard is maintained by the Organization for the Advancement of Structured Information Standards (OASIS), and is supported by many organizations. Many existing key management devices are KMIP-compliant, allowing easy integration for organizations with deployed key management solutions.

The protocol supports a variety of managed object types, including standard symmetric and asymmetric encryption keys and certificates as well as more generic object structures, like secret data (e.g., passwords) and opaque data objects [3]. By requiring the use of Transport Layer Security (TLS), KMIP-compliant devices and services can securely communicate across public and private networks [4]. With built-in support for algorithms compliant with Federal Information Processing Standard (FIPS), KMIP is well-suited for organizations working under strict security requirements [5].

The addition of KMIP support in Barbican required the creation of PyKMIP,[2] an open-source Python library, which implements the KMIP specification. Although equivalent proprietary software packages previously existed, these were untenable for use in OpenStack because they were not open source. PyKMIP provides a client utility that supports the core key management operations and object attributes required for the key management life cycle. The library also includes a KMIP server application that can be used for testing and demonstration [6]. This server allows developers to test the full key management life cycle, defraying the upfront investment required for production-level key management appliances.

B. Block Storage Encryption

OpenStack provides two types of block storage: ephemeral storage exists only for the lifetime of a VM whereas persistent volumes exist independently of any VM. Persistent volumes may be attached and detached from VMs; when attached to a VM the persistent volume appears as a block device that users may format or use as unpartitioned space. Not surprisingly, block storage is an attractive target for attackers because it contains data for many different cloud consumers. It is critical to provide controls that ensure the confidentiality of that data.

Our approach (shown in Figure 4) assumes that only the compute host that manages the user's VM is trustworthy. The

[2]https://github.com/OpenKMIP/PyKMIP

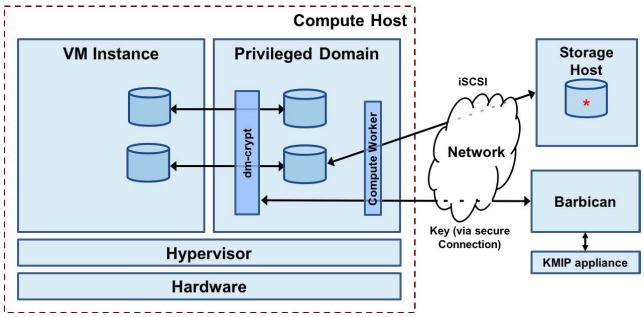

Fig. 4. Depiction of block storage encryption. The compute host has a single VM with ephemeral storage and a persistent volume connected via iSCSI to a storage host. dm-crypt intercepts reads and writes to both types of block storage to provide transparent encryption and decryption of the data. The dashed red line indicates the trust boundary.

implementation of encryption support for ephemeral storage and persistent volumes is essentially identical. The privileged domain of the compute host allocates block storage for a VM, either as a Logical Volume Manager (LVM) partition (ephemeral storage) or block device connected via iSCSI (persistent volume). Block devices may be encrypted either by a configuration option or by a specific request.

If the device is encrypted, then Nova retrieves an encryption key from Barbican. In the case of ephemeral storage, Barbican creates an encryption key for the VM, stores a copy of it,[3] and returns the key to the compute host. The encryption key for an encrypted persistent volume is created when the volume itself is created. In this case, an identifier for the encryption key is provided as part of the request, and Barbican returns that key in response to the request.

Before this block storage is provided to the VM, the privileged domain uses dm-crypt, part of Linux's device mapper infrastructure, to encrypt the block device. The interjection of dm-crypt is transparent to the VM. The VM itself has no knowledge of the encryption, making it impossible for a user or process within that VM to disable the encryption. Encryption of data in transit and encryption of data at rest are provided simultaneously using this scheme.

Because dm-crypt uses the Linux kernel's Crypto API, any cipher, mode, or key size supported by the kernel is available for use. Hardware improvements such as AES extensions (e.g., Intel's AES-NI) minimize the overhead introduced by the encryption. In addition, no changes to OpenStack are required to take advantage of these software and hardware features, as they are available via regular kernel updates to the compute host.

Performing encryption on the compute host does have some disadvantages. First, storage hosts cannot take advantage of techniques like compression and deduplication to reduce the amount of storage required. Deduplication alone provides more than 90% disk and bandwidth savings in common business

[3]A copy of the encryption key must be preserved to support operations like live migration, which transfers the VM's encrypted data to another compute host.

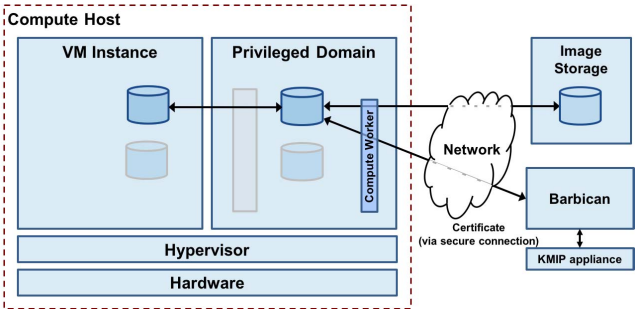

Fig. 5. Depiction of image signature verification. The root disk of the VM is derived from a base image that is retrieved from Glance, the image service. Signed images are verified against a certificate stored by the key manager; if the verification fails, the VM instance is not spawned and an error is reported to the user. The trust boundary is marked by the dashed red line.

settings [7]. The high entropy of encrypted data typically renders compression and deduplication ineffective although various schemes have been proposed to mitigate this issue (e.g., convergent encryption [8], [9]). In practice, we believe that the security benefits of encryption will outweigh these immediate costs. They may also be mitigated by allowing the storage hosts to access the encryption keys at the cost of increasing the trust in the cloud provider.

C. Image Integrity

Image integrity is an important component of cloud security. If an image is compromised, the modified image could contain malicious code and allow an attacker to gain a persistent foothold in the cloud. Prior to the implementation of the image integrity feature, the only verification for images was checking an MD5 hash of the image data, which is insufficient to protect against the malicious modification of images.

The image integrity feature adds digital signature verification to detect unauthorized modifications to images while they are in transit between the user and Glance, while they are at rest, and while they are in transit between Glance and Nova. Signature verification occurs in Glance upon the initial upload of the image, and again in Nova (see Figure 5) when the image is retrieved from Glance in order to boot an instance.

The following steps outline how image integrity is supported.

1) The user signs an image using a key pair. For maximum security, the private key should not be known to the cloud provider.
2) The user stores an X.509 certificate containing the public key in the key management service.
3) The user uploads the image and the signature to the cloud.
4) Glance verifies the signature of the image. This initial check detects invalid signatures immediately rather than waiting until the user wants to use the image.
5) The user requests that a VM be booted from the signed image. Nova retrieves the image and signature and verifies the signature prior to booting the VM.

Signature verification only occurs when the signature of the image is provided prior to or during the upload of the image. If no signature is provided, the signature verification does not occur when the image is uploaded.

It was important to the community that this functionality be able to be turned off, so that it would not affect performance for users who were uninterested in the feature. As such, in Nova this feature is off by default, but can be enabled with a configuration option, which only allows images with valid signatures to be booted.

IV. Security Analysis

Unfortunately, perfect security is likely an impossible objective when the user places data under the control of a third party cloud computing provider.[4] We want to limit that trust to the platform responsible for processing the data—i.e., the compute host that manages the user's VM.

Both the block storage encryption and image integrity features include the compute host in their trust boundary. There are tools available that provide awareness of the compute host's integrity. Trusted compute pools [11] is a feature that combines hardware-based security features (e.g., a Trusted Platform Module (TPM)) with remote attestation to ensure that a compute host is running in a known good configuration. Although OpenStack's trusted compute pools is currently based on OpenAttestation,[5] which only attests to the integrity of a compute host's Basic Input/Output System (BIOS), hypervisor, and operating system (OS), complementary technologies such as Linux's Integrity Measurement Architecture (IMA) [12] can be used to extend the attestation to running applications (i.e., the software executing on the compute host). Given these security controls, a user has a high degree of confidence that the platform is in a known good configuration. Moreover, the measurements provided by a TPM and IMA could be used as part of the authentication before releasing an encryption key or signing certificate.

A. Block Storage Encryption

Starting from the VM instance that actually initiates disk input / output (I/O), there are a number of places from which to compromise the data's confidentiality. The block storage encryption feature protects the confidentiality of data from the following threats:

- Unsanitized disk media: Although best practices dictate sanitization, data remanence remains a concern, particularly for a stolen or physically damaged disk.
- Reused disk sectors: Reuse of physical resources (i.e., multi-tenancy) without prior overwriting allows the prior data to be retrieved by other users.
- Compromised storage host: A storage host is a prime target for an attacker because many different volumes are accessible.
- Intercepted network traffic: iSCSI offers no intrinsic security, and the recommended practice to use IPsec [13] is frequently not followed due to its performance overhead.

[4]Techniques like fully homomorphic encryption [10] hold promise for provably secure computations in the cloud, but such methods have not yet been shown to be practical.

[5]https://01.org/openattestation

Assuming that a version of the data being written to the volume exists within the memory of a VM instance, the compute host must be trusted. Encrypting the data on the compute host ensures its confidentiality as it traverses the network to the storage host and is written to the physical media. Moreover, it is relatively simple to determine which cloud computing services should have access to the data; the encryption key is not provided to any host except the one managing the VM. Features like trusted compute pools [11] provide end users with assurance that the compute host has not been compromised.

An attacker with access to the storage host or network has access to the data in its encrypted form. In such cases, the attacker threatens the integrity and availability of the data—e.g., by overwriting portions of that data. While such attacks are a concern, they may be handled via appropriate protections by the user in many cases. For example, the user may take periodic snapshots for backups.

An obvious alternative is to allow the user to control the encryption directly—e.g., by encrypting data within the VM before it is written to disk. Although this idea has merit, it cannot be used for the root disk (because the user has no way to provide the encryption key to decrypt the disk when booting) and ultimately is no more secure than the implementation previously described. In particular, if the compute host has been compromised, then—short of advances in provable obfuscation—an attacker can extract the user's encryption key from the VM via brute force attacks. Moreover, delegating security to the end user raises the specter of an incorrect configuration or accidental omission of encryption that undermines the security completely.

B. Image Integrity

The image integrity feature secures OpenStack against the following threats:

- Man-in-the-Middle attack: An adversary with access to the network alters the image data when uploaded by the end user or downloaded by another OpenStack service.
- Untrusted Glance: In a hybrid cloud deployment, Glance is hosted in a physically insecure location or by a company with limited security infrastructure.

Image integrity ensures that the same signed images uploaded into Glance will be used in Nova. This feature gives the end user a high degree of assurance that the image has not been modified. We assume that an end user choosing to run a workload in a cloud environment places some trust in the cloud provider. As with the design for block storage encryption, our objective is to restrict the amount of trust placed in the cloud. Other cloud offerings currently lack the ability to verify the integrity of a VM images.

V. Evaluation

This section presents a performance evaluation of the security features previously described. Unless otherwise noted, the performance evaluation used an OpenStack deployment running the Mitaka release of each service. Table I provides basic information about the machines. The network backbone comprised 10 Gbps Arista switches. Cinder had a 14 TB LVM volume for persistent storage.

TABLE I
Hardware specifications of the OpenStack cloud

	CPU			Storage (GBs)	
Node	GHz	Cores	Threads	RAM	Disk
Cinder	2.1	12	24	64	14336
Compute controller	2.6	14	28	128	4096
Compute hosts	2.6	16	32	256	4096
Networking	2.0	12	24	64	900

A. PyKMIP

To act as a suitable testing replacement for a hardware key management device, the PyKMIP server must be able to efficiently create and return managed objects. To gauge server performance, timing measurements were made for bulk key creation and retrieval operations against the PyKMIP server (version 0.6.0). Each bulk operation set included 1000 requests per client with 1–8 clients operating simultaneously to identify client and server performance bottlenecks. These performance tests were conducted on an Ubuntu 16.04 LTS VM with 4 CPUs, 12.288 GB of RAM, and 60 GBs of storage. Python 3.5 was used to run the PyKMIP client and server. Both applications were hosted on the same VM to eliminate network delay.

Figure 6 shows the key creation throughput for different numbers of clients. Asymmetric key creation performance is limited by the computational intensity of generating asymmetric key pairs on the server. Increasing the number of clients issuing asymmetric key creation requests has little impact on performance. Symmetric key creation performance is approximately 50 keys created per second for an individual client. Increasing the client count corresponds to an initial increase in key creation rate, indicating that server performance approaches 75 keys created per second as operation frequency increases. Adding additional clients does not improve performance.

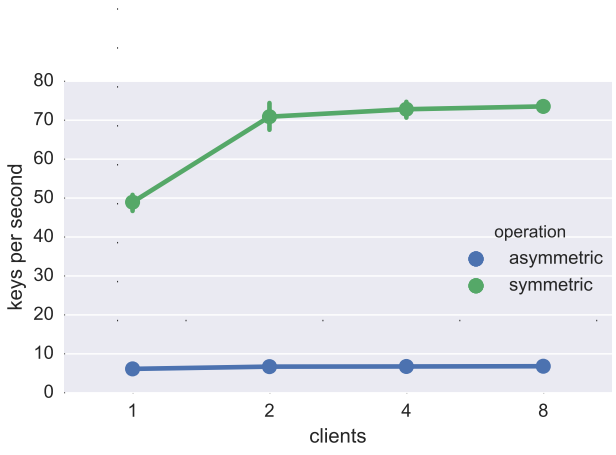

Fig. 6. Key creation performance for the PyKMIP server.

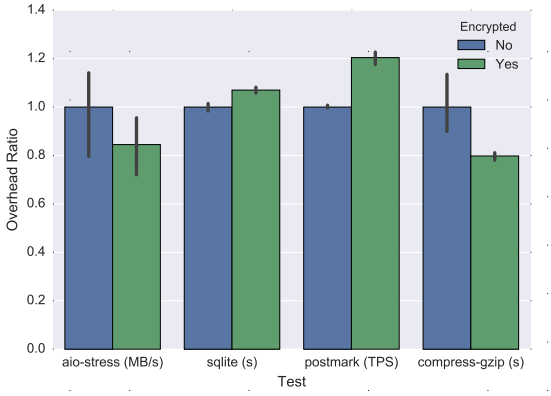

Fig. 7. Storage encryption overhead for benchmarks. The error bars represent the 95% confidence interval of the mean. Each test result has been normalized by the mean when using an unencrypted volume. The unit of measurement for each test is shown in parentheses after the test name.

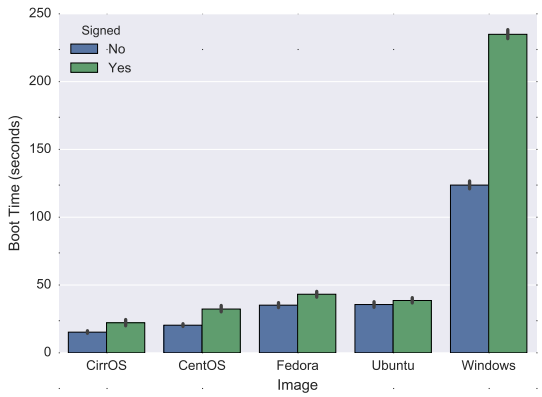

Fig. 8. Image signing overhead for different OS images. The error bars represent the 95% confidence interval of the mean. Verifying an image signature increases the boot time by a mean of 45% although the overhead varies considerably by image.

Key retrieval performance behaves similarly to that of symmetric key creation. An individual client is capable of completing approximately 70 key retrieval requests per second. Increasing the client count corresponds to an initial increase in key retrieval rate, with the server upper bound for key retrieval stabilizing at approximately 120 keys retrieved per second. Adding additional clients does not improve overall performance.

B. Block Storage Encryption

To evaluate the performance penalty imposed by encrypting block storage, we used a selection of I/O benchmarks from the Phoronix Test Suite.[6] We created an `m1.large` VM instance with 4 virtual CPUs, 8 GBs RAM, and 80 GBs of ephemeral storage. We attached two 20 GB persistent volumes, one of which was encrypted. The encrypted volume uses AES-XTS with a 256-bit key.

Figure 7 shows the performance overhead imposed by the encrypted block storage feature. As evidenced by the figure, the overhead varies widely, up to a 20% difference for Postmark and gzip compression. Of course, encrypting data should always decrease performance. We do not attempt to quantify the overhead more precisely or investigate possible sources of noise for two reasons. First, we observe that overhead varies significantly by workload. VMs dedicated to particular workloads likely require additional benchmarks to determine the impact of encryption for those workloads. Second, real-world environments are subject to significant noise such as network latency to the storage host. Although we could design experiments to eliminate such artifacts, those experiments would not be representative of actual cloud environments. Hence, we merely observe that encrypting block storage introduces a tolerable performance penalty.

[6]http://www.phoronix-test-suite.com

C. Image Integrity

To measure the performance overhead of image signature verification, we measured the length of time required to launch VM instances with and without signature verification. The timing is the wall clock time from submitting the request to boot the VM instance until Nova reported that the VM entered the "Active" state. Images were not reused for multiple tests because the compute hosts cache each image after its first use. Table II provides basic statistics about the images used in our evaluation.

Figure 8 illustrates the boot times for various images with and without image signature verification. Each bar indicates the mean of 100 timings. As evidenced by the figure, image signature verification imposes a measurable overhead when booting a VM image. The overhead varies considerably across images from a low of 8.4% for Ubuntu to a high of 89.9% for Windows. The mean overhead was 45%.

The overhead is largely attributable to computing the SHA256 hash of the image as part of the signature verification. Computing this hash requires reading all the image data, which otherwise may not be required to boot the image. The cause for the significant variance in the overhead is unclear. It could be that background processes and network traffic affected our timing, in which case collecting additional samples and using a trimmed mean might produce a more consistent result. Nevertheless, any noise introduced by such causes provides a

TABLE II
IMAGES USED TO EVALUATE IMAGE SIGNATURE VERIFICATION

OS	Image	Size (MBs)
CirrOS	cirros-0.3.4	12.6
CentOS	CentOS-7	856.6
Fedora	fedora-atomic-25	533.1
Ubuntu	ubuntu-16.04.1	300.3
Windows	windows_server_2012	15924.4

real-world picture of what should be expected in an operational cloud.

VI. Lessons Learned

Over the course of our involvement with OpenStack, we learned many lessons about how to contribute security features to an established open source project. We summarize these lessons for the benefit of other researchers interested in transitioning their work into operational settings. We hope that others can benefit from our experiences to ease the process.

Most of our team had no direct experience contributing to open source projects prior to our work with OpenStack. We initially expected the OpenStack community to embrace our security features as a matter of best practices. We found instead that successful contributions take time and many iterations of a feature are often required before it is deemed sufficiently mature for release.

Lesson 1: *Embrace open development practices.*

Early in our OpenStack involvement, we experienced a clash between organizational practices and those of the open source community. The best example was our attempt to deliver "final" features for acceptance by the community following internal design and code reviews. The community pushed back strongly: Having not been privy to the internal development activities, they naturally wanted input into many of the implementation decisions, and the timing of our contributions was often too close to code freezes to allow for effective community review.

Our internal review processes had to be modified to work effectively within an open source community with as much activity as OpenStack. We modified our approach to do as much as possible publicly, from code reviews of patches submitted by others on our team to posting "work-in-progress" features to give the broader community awareness of our ongoing work. In many cases, the community has modified our original designs, and receiving this feedback early in the development process (rather than waiting until a feature was "finished") eliminates a large amount of rework.

Lesson 2: *Contribute to the community's goals.*

Open source software requires a strong community to be successful. Limited progress ensues if everyone works on only those features that are of interest to them (as individuals or organizations). As outlined in Section II-B, successful contributions require commitment to support both one's own work and others' work. In practice, this commitment means submitting bug reports and bug fixes, performing code reviews, and sometimes assisting with the development of other features.

When we started contributing to OpenStack, we did few of these activities. Later, we learned that more active participation does improves the feedback time, and although the effect is not as dramatic as we might like, there are many community-oriented activities to backfill our time. Our participation ultimately makes OpenStack more successful, and in recognition of our increased involvement, we received leadership roles with greater influence over the community's direction.

Lesson 3: *Beware of stovepipes among teams.*

Just because a project follows an open development model does not mean that all stakeholders are involved in its process. The implementation of our block storage encryption and image integrity features required cross-project collaboration. For example, the development of the image integrity feature required collaboration with both the Glance and Nova teams. At times, we were the intermediary between two teams, each with limited interest in our work, which severely limited progress.

The Glance team originally stipulated that a signature of an image's MD5 hash be used instead of a signature of the image data. The former alleviates performance concerns regarding hashing the image data twice. Because this path appeared to be the only way forward, we developed a detailed plan to incrementally modify an initial implementation to improve the feature's security after its initial release. However, the Nova team rejected the initial implementation, requiring the use of a signature of the image data directly. The end result was a six-month delay in the feature being available due to the conflicting requirements for acceptance.

Lesson 4: *Acceptance takes time—don't rush!*

Anecdotally, we have found that a feature that requires 6 weeks to prototype requires 6–12 months for acceptance in OpenStack. Much of this time is spent working through implementation details. For example, is it possible to eliminate an API call to another service to improve performance? While this detail is not significant to a security engineer, to a project that emphasizes massive scalability, it is an important detail. Although such delays can seem lengthy, they ultimately improve our understanding of the OpenStack code base and the concerns of others in the community.

In an effort to satisfy deadlines for feature acceptance, the temptation is to use a stopgap design for initial functionality and to refine that design as part of future development cycles. The prior example of the conflicting design requirements of Glance and Nova is one cautionary tale. In the end, our initial implementation was reworked completely because many of the design decisions, which ultimately proved contentious, were made in concert with the Glance team due to an initial lack of interest from the Nova developers. Another example was a simple key management utility that we provided for the block storage encryption feature because the Barbican project was not yet ready for its initial release. Due to backward compatibility requirements, we must support that key management utility even though the block storage encryption feature did not appear to be used (outside of testing) prior to the availability of Barbican.

VII. Related Work

OpenStack supports additional security features that complement those described in this paper. For example, OpenStack's object storage service, Swift, supports encrypting data at rest on its storage nodes. Amazon Web Services (AWS) is the best-known cloud offering for comparison to OpenStack. Both OpenStack and AWS assume a shared responsibility model for security: OpenStack and AWS manage the security of services (e.g., compute, storage, and networking) with customers

responsible for the security of OSs, network configuration, and ultimately customer data. AWS Key Management Service (KMS) provides similar capabilities to Barbican, and Elastic Block Store (EBS) volumes can be encrypted using a similar approach to our block storage encryption feature. However, OpenStack's features allow greater customization by cloud operators. We are not aware of an AWS equivalent to the pre-boot VM integrity feature described in Section III-C.

Two adversarial models underlie most academic research in cloud computing security. The honest-but-curious (or semi-honest) model assumes an adversary who faithfully carries out a specified protocol but violates the confidentiality of any information gathered in the process. The malicious model assumes an adversary whose goal is to violate user privacy by modifying data, falsifying records, etc. Unlike these adversarial models, our security features clearly place some trust in the cloud provider. We feel that this trust is pragmatic: a cloud provider must maintain a favorable reputation to continue to do business.

Recent research specifically addresses OpenStack, including single points of failure [14]; an analysis and characterization of various SDN controllers and configurations [15]; and a framework and methodology to analyze the performance, scalability, and availability of compute resources [16]. Song et al. [17] propose the use of AES encryption algorithms that support homomorphic encryption techniques. Tari et al. [18] emphasize tenant isolation, which our encryption feature provides. A security model supporting similar encryption principles to our feature but with role-based access control has also been proposed [19]. To the best of our knowledge, none of this work is part of OpenStack's official code base.

VIII. Conclusion

This paper presents several data protection features available within OpenStack. These features include the PyKMIP library, block storage encryption, and image signature verification. We also describe the general steps needed to have such security features accepted by the OpenStack community. According to Huang et al. [20]:

> Current industry indicators suggest that OpenStack will likely become the dominant cloud hosting platforms, indicating that [...] OpenStack will have a big influence on which security mechanisms the public IaaS clouds of the future will support.

Our work serves as a model to foster more research into operational cloud deployments.

As part of our future work, we will investigate additional security features for OpenStack. However, much of our ongoing focus is to support and to improve the existing features that we described in this paper. In particular, we are gradually expanding PyKMIP's support for KMIP's Basic Baseline profile [3] and creating a dashboard plugin to enumerate a user's keys and to retrieve their certificates. We also continue to add enhancements to the image integrity feature (e.g., the ability to sign and verify snapshots of VM instances).

IX. Acknowledgments

We thank John and Susan Kelley for managing the OpenStack cloud used for our performance evaluation.

References

[1] C. Coles and J. Yeoh, "Cloud Adoptions Practices & Priorities Survey Report," January 2015.

[2] "Cloud Adoption & Risk Report," Skyhigh Networks, Tech. Rep., 2015.

[3] R. Griffin, S. Sankuratripati, R. Haas, and I. Fitzgerald, "Key Management Interoperability Protocol Specification 1.1," Tech. Rep., January 2013. [Online]. Available: https://docs.oasis-open.org/kmip/spec/v1.1/os/kmip-spec-v1.1-os.pdf

[4] R. Griffin and S. Sankuratripati, "Key Management Interoperability Protocol Profiles Version 1.1," Tech. Rep., January 2013, accessed: 13 December 2016. [Online]. Available: https://docs.oasis-open.org/kmip/profiles/v1.1/os/kmip-profiles-v1.1-os.pdf

[5] S. Saha, T. Cox, T. Hudson, and R. Lockhart, "KMIP Symmetric Key Foundry for FIPS 140-2 Profile Version 1.0," Tech. Rep., May 2015, accessed: 13 December 2016. [Online]. Available: https://docs.oasis-open.org/kmip/kmip-sym-foundry-profile/v1.0/os/kmip-sym-foundry-profile-v1.0-os.html

[6] P. Hamilton, "PyKMIP: Key Management for Python," Poster presentation at PyCon 2016, June 2016.

[7] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side Channels in Cloud Services: Deduplication in Cloud Storage," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40–47, November 2010.

[8] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," in *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, ser. ICDCS '02, 2002, pp. 617–624.

[9] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller, "Secure Data Deduplication," in *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*, ser. StorageSS '08, October 2008, pp. 1–10.

[10] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, ser. STOC '09, 2009, pp. 169–178.

[11] *OpenStack Administrator Guide*. [Online]. Available: http://docs.openstack.org/admin-guide/

[12] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture," in *USENIX Security Symposium*, vol. 13, 2004, pp. 223–238.

[13] K. Z. Meth and J. Satran, "Design of the iSCSI protocol," in *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, ser. MSST '03, April 2003, pp. 116–122.

[14] Y. Wang and X. Li, "Achieve High Availability about Point-Single Failures in OpenStack," in *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, vol. 01, Dec 2015, pp. 45–48.

[15] O. Tkachova, M. J. Salim, and A. R. Yahya, "An Analysis of SDN-OpenStack Integration," in *2015 Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology*, ser. PIC S&T '15, Oct 2015, pp. 60–62.

[16] G. Almsi, J. G. Castaos, H. Franke, and M. A. L. Silva, "Toward building highly available and scalable OpenStack clouds," *IBM Journal of Research and Development*, vol. 60, no. 2-3, pp. 5:1–5:10, March 2016.

[17] C. Song, Y. Park, J. Gao, S. K. Nanduri, and W. Zegers, "Favored Encryption Techniques for Cloud Storage," in *2015 IEEE First International Conference on Big Data Computing Service and Applications*, March 2015, pp. 267–274.

[18] Z. Tari, X. Yi, U. S. Premarathne, P. Bertok, and I. Khalil, "Security and Privacy in Cloud Computing: Vision, Trends, and Challenges," *IEEE Cloud Computing*, vol. 2, no. 2, pp. 30–38, March 2015.

[19] Y. Ghebghoub, O. Boussaid, and S. Oukid, "Security Model Based Encryption to Protect Data on Cloud," in *Proceedings of the International Conference on Information Systems and Design of Communication*, ser. ISDOC '14, May 2014, pp. 50–55.

[20] W. Huang, A. Ganjali, B. H. Kim, S. Oh, and D. Lie, "The State of Public Infrastructure-as-a-Service Cloud Security," *ACM Computing Surveys*, vol. 47, no. 4, pp. 68:1–68:31, June 2015.