OpenStack Security Guide

master (2013-10-02)

This book provides best practices and conceptual information about securing an OpenStack cloud.

Table of Contents

Preface

Document change history

This version of the guide replaces and obsoletes all previous versions. The following table describes the most recent changes:

Revision Date	Summary of Changes
July 2, 2013	• Initial creation...

1. Acknowledgements

The OpenStack Security Group would like to acknowledge contributions from the following organizations who were instrumental in making this book possible. These are:


redhat.


hp


nebula.

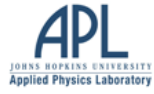

NATIONAL SECURITY AGENCY
UNITED STATES OF AMERICA


intel


nicira
by vmware


rackspace
the open cloud company


cloudscaling


APL
JOHNS HOPKINS UNIVERSITY
Applied Physics Laboratory


FLOSS MANUALS
'free manuals for free software'


CloudPassage

2. Why and How We Wrote This Book

As OpenStack adoption continues to grow and the product matures, security has become a priority. The OpenStack Security Group has recognized the need for a comprehensive and authoritative security guide. The **OpenStack Security Guide** has been written to provide an overview of security best practices, guidelines, and recommendations for increasing the security of an OpenStack deployment. The authors bring their expertise from deploying and securing OpenStack in a variety of environments.

The guide augments the *OpenStack Operations Guide* and can be referenced to harden existing OpenStack deployments or to evaluate the security controls of OpenStack cloud providers.

Objectives

- Identify the security domains in OpenStack
- Provide guidance to secure your OpenStack deployment
- Highlight security concerns and potential mitigations in present day OpenStack
- Discuss upcoming security features
- To provide a community driven facility for knowledge capture and dissemination

How

As with the OpenStack Operations Guide, we followed the book sprint methodology. The book sprint process allows for rapid development and production of large bodies of written work. Coordinators from the OpenStack Security Group re-enlisted the services of Adam Hyde as facilitator. Corporate support was obtained and the project was formally announced during the OpenStack summit in Portland, Oregon.

The team converged in Annapolis, MD due to the close proximity of some key members of the group. This was a remarkable collaboration between public sector intelligence community members, silicon valley startups and some large, well-known technology companies. The book sprint ran during the last week in June 2013 and the first edition was created in five days.

The team included:

- **Bryan D. Payne**, Nebula

  Dr. Bryan D. Payne is the Director of Security Research at Nebula and co-founder of the OpenStack Security Group (OSSG). Prior to joining Nebula, he worked at Sandia National Labs, the National Security Agency, BAE Systems, and IBM Research. He graduated with a Ph.D. in Computer Science from the Georgia Tech College of Computing, specializing in systems security.

- **Robert Clark**, HP

  Robert Clark is the Lead Security Architect for HP Cloud Services and co-founder of the OpenStack Security Group (OSSG). Prior to being recruited by HP, he worked in the UK Intelligence Community. Robert has a strong background in threat modeling, security architecture and virtualization technology. Robert has a master's degree in Software Engineering from the University of Wales.

- **Keith Basil**, Red Hat

  Keith Basil is a Principal Product Manager for Red Hat OpenStack and is focused on Red Hat's OpenStack product management, development

and strategy. Within the US public sector, Basil brings previous experience from the design of an authorized, secure, high-performance cloud architecture for Federal civilian agencies and contractors.

- **Cody Bunch**, Rackspace

  Cody Bunch is a Private Cloud architect with Rackspace. Cody has co-authored an update to "The OpenStack Cookbook" as well as books on VMware automation.

- **Malini Bhandaru**, Intel

  Malini Bhandaru is a security architect at Intel. She has a varied background, having worked on platform power and performance at Intel, speech products at Nuance, remote monitoring and management at ComBrio, and web commerce at Verizon. She has a Ph.D. in Artificial Intelligence from the University of Massachusetts, Amherst.

- **Gregg Tally**, Johns Hopkins University Applied Physics Laboratory

  Gregg Tally is the Chief Engineer at JHU/APL's Cyber Systems Group within the Asymmetric Operations Department. He works primarily in systems security engineering. Previously, he has worked at SPARTA, McAfee, and Trusted Information Systems where he was involved in cyber security research projects.

- **Eric Lopez**, Nicira / VMware

  Eric Lopez is Senior Solution Architect at VMware's Networking and Security Business Unit where he helps customer implement OpenStack and Nicira's Network Virtualization Platform. Prior to joining Nicira, he worked for Q1 Labs, Symantec, Vontu, and Brightmail. He has a B.S in Electrical Engineering/Computer Science and Nuclear Engineering from U.C. Berkeley and MBA from the University of San Francisco.

- **Shawn Wells**, Red Hat

  Shawn Wells is the Director, Innovation Programs at Red Hat, focused on improving the process of adopting, contributing to, and managing open source technologies within the U.S. Government. Additionally, Shawn is an upstream maintainer of the SCAP Security Guide project which forms virtualization and operating system hardening policy with the U.S. Military, NSA, and DISA. Formerly an NSA civilian, Shawn developed SIGINT collection systems utilizing large distributed computing infrastructures.

- **Ben de Bont**, HP

  Ben de Bont is the CSO for HP Cloud Services. Prior to his current role Ben led the information security group at MySpace and the incident response team at MSN Security. Ben holds a master's degree in Computer Science from the Queensland University of Technology.

- **Nathanael Burton**, National Security Agency

  Nathanael Burton is a Computer Scientist at the National Security Agency. He has worked for the Agency for over 10 years working on distributed systems, large-scale hosting, open source initiatives, operating systems, security, storage, and virtualization technology. He has a B.S. in Computer Science from Virginia Tech.

- **Vibha Fauver**

  Vibha Fauver, GWEB, CISSP, PMP, has over fifteen years of experience in Information Technology. Her areas of specialization include software engineering, project management and information security. She has a B.S. in Computer & Information Science and a M.S. in Engineering Management with specialization and a certificate in Systems Engineering.

- **Eric Windisch**, Cloudscaling

  Eric Windisch is a Principal Engineer at Cloudscaling where he has been contributing to OpenStack for over two years. Eric has been in the trenches of hostile environments, building tenant isolation and infrastructure security through more than a decade of experience in the web hosting industry. He has been building cloud computing infrastructure and automation since 2007.

- **Andrew Hay**, CloudPassage

  Andrew Hay is the Director of Applied Security Research at CloudPassage, Inc. where he leads the security research efforts for the company and its server security products purpose-built for dynamic public, private, and hybrid cloud hosting environments.

- **Adam Hyde**

  Adam facilitated this Book Sprint. He also founded the Book Sprint methodology and is the most experienced Book Sprint facilitator around. Adam founded FLOSS Manuals—a community of some 3,000 individuals developing Free Manuals about Free Software. He is also the

founder and project manager for Booktype, an open source project for writing, editing, and publishing books online and in print.

During the sprint we also had help from Anne Gentle, Warren Wang, Paul McMillan, Brian Schott and Lorin Hochstein.

This Book was produced in a 5 day book sprint. A book sprint is an intensely collaborative, facilitated process which brings together a group to produce a book in 3-5 days. It is a strongly facilitated process with a specific methodology founded and developed by Adam Hyde. For more information visit the book sprint web page at http://www.booksprints.net.

How to Contribute to This Book

The initial work on this book was conducted in an overly air-conditioned room that served as our group office for the entirety of the documentation sprint.

Learn more about how to contribute to the OpenStack docs: http://wiki.openstack.org/Documentation/HowTo.

3. Introduction to OpenStack

This guide provides security insight into OpenStack deployments. The intended audience is cloud architects, deployers, and administrators. In addition, cloud users will find the guide both educational and helpful in provider selection, while auditors will find it useful as a reference document to support their compliance certification efforts. This guide is also recommended for anyone interested in cloud security.

Each OpenStack deployment embraces a wide variety of technologies, spanning Linux distributions, database systems, messaging queues, OpenStack components themselves, access control policies, logging services, security monitoring tools, and much more. It should come as no surprise that the security issues involved are equally diverse, and their in-depth analysis would require several guides. We strive to find a balance, providing enough context to understand OpenStack security issues and their handling, and provide external references for further information. The guide could be read from start to finish or sampled as necessary like a reference.

We briefly introduce the kinds of clouds: private, public, and hybrid before presenting an overview of the OpenStack components and their related security concerns in the remainder of the chapter.

Cloud Types

OpenStack is a key enabler in adoption of cloud technology and has several common deployment use cases. These are commonly known as Public, Private, and Hybrid models. The following sections use the National Institute of Standards and Technology (NIST) definition of cloud to introduce these different types of cloud as they apply to OpenStack.

Public Cloud

According to NIST, a public cloud is one in which the infrastructure is open to the general public for consumption. OpenStack public clouds are typically run by a service provider and can be consumed by individuals, corporations, or any paying customer. A public cloud provider may expose a full set of features such as software defined networking, block storage,

in addition to multiple instance types. Due to the nature of public clouds, they are exposed to a higher degree of risk. As a consumer of a public cloud you should validate that your selected provider has the necessary certifications, attestations, and other regulatory considerations. As a public cloud provider, depending on your target customers, you may be subject to one or more regulations. Additionally, even if not required to meet regulatory requirements, a provider should ensure tenant isolation as well as protecting management infrastructure from external attacks.

Private Cloud

At the opposite end of the spectrum is the private cloud. As NIST defines it, a private cloud is provisioned for exclusive use by a single organization comprising multiple consumers (e.g. business units). It may be owned, managed, and operated by the organization, a third-party, or some combination of them, and it may exist on or off premises. Private cloud use cases are diverse, as such, their individual security concerns vary.

Community cloud

NIST defines a community cloud as one whose infrastructure is provisioned for the exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third-party, or some combination of them, and it may exist on or off premises.

Hybrid Cloud

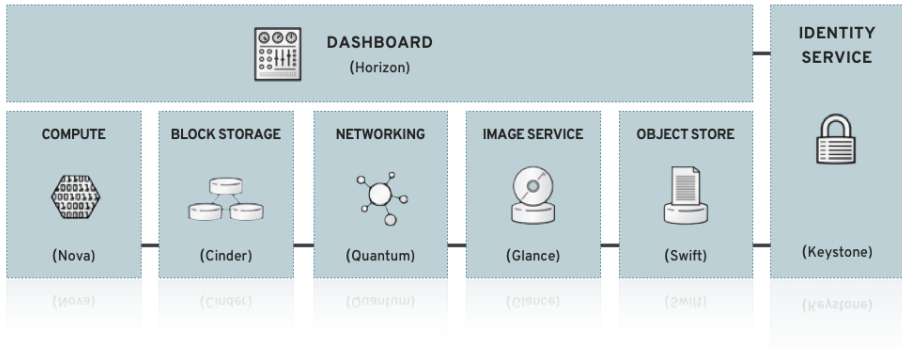
A hybrid cloud is defined by NIST as a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds). For example an online retailer may have their advertising and catalogue presented on a public cloud that allows for elastic provisioning. This would enable them to handle seasonal loads in a flexible, cost-effective fashion. Once a customer begins to process their order, they are transferred to the more secure private cloud backend that is PCI compliant.

For the purposes of this document, we treat Community and Hybrid similarly, dealing explicitly only with the extremes of Public and Private

clouds from a security perspective. Your security measures depend where your deployment falls upon the private public continuum.

OpenStack Service Overview

OpenStack embraces a modular architecture to provide a set of core services that facilitates scalability and elasticity as core design tenets. This chapter briefly reviews OpenStack components, their use cases and security considerations.



OpenStack Compute

OpenStack compute (Nova) provides services to support the management of virtual machine instances at scale, instances that host multi-tiered applications, dev/test environments, "Big Data" crunching Hadoop clusters, and/or high performance computing.

Nova facilitates this management through an abstraction layer that interfaces with supported hypervisors, which we address later on in more detail.

Later in the guide, we focus generically on the virtualization stack as it relates to hypervisors.

For information about the current state of feature support, see OpenStack Hypervisor Support Matrix.

The security of Nova is critical for an OpenStack deployment. Hardening techniques should include support for strong instance isolation, secure communication between Nova sub-components, and resiliency of public-facing *API* endpoints.

OpenStack Object Storage

OpenStack object storage service (Swift) provides support for storing and retrieving arbitrary data in the cloud. Swift provides both a native API and an Amazon Web Services S3 compatible API. The service provides a high degree of resiliency through data replication and can handle petabytes of data.

It is important to understand that object storage differs from traditional file system storage. It is best used for static data such as media files (MP3s, images, videos), virtual machine images, and backup files.

Object security should focus on access control and encryption of data in transit and at rest. Other concerns may relate to system abuse, illegal or malicious content storage, and cross authentication attack vectors.

OpenStack Block Storage

The OpenStack Block Storage service (Cinder) provides persistent block storage for compute instances. Cinder is responsible for managing the life-cycle of block devices, from the creation and attachment of volumes to instances, to their release.

Security considerations for block storage are similar to that of object storage.

OpenStack Networking

The OpenStack networking service (Neutron, previously called Quantum) provides various networking services to cloud users (tenants) such as IP address management, *DNS*, *DHCP*, load balancing, and security groups (network access rules, like firewall policies). It provides a framework for software defined networking (SDN) that allows for pluggable integration with various networking solutions.

OpenStack Networking allows cloud tenants to manage their guest network configurations. Security concerns with the networking service include network traffic isolation, availability, integrity and confidentiality.

OpenStack Dashboard

The OpenStack dashboard service (Horizon) provides a web-based interface for both cloud administrators and cloud tenants. Through this

interface administrators and tenants can provision, manage, and monitor cloud resources. Horizon is commonly deployed in a public facing manner with all the usual security concerns of public web portals.

Identity Management

The identity management service (Keystone) is a **shared service** that provides authentication and authorization services throughout the entire cloud infrastructure. Keystone has pluggable support for multiple forms of authentication.

Security concerns here pertain to trust in authentication, management of authorization tokens, and secure communication.

Image Service

The OpenStack image service (Glance) provides disk image management services. Glance provides image discovery, registration, and delivery services to Nova, the compute service, as needed.

Trusted processes for managing the life cycle of disk images are required, as are all the previously mentioned issues with respect to data security.

Other Supporting Technology

OpenStack relies on messaging for internal communication between several of its services. By default, OpenStack uses message queues based on the Advanced Message Queue Protocol (*AMQP*). Similar to most OpenStack services, it supports pluggable components. Today the implementation backend could be *RabbitMQ*, *Qpid*, or *ZeroMQ*.

As most management commands flow through the message queueing system, it is a primary security concern for any OpenStack deployment. Message queueing security is discussed in detail later in this guide.

Several of the components use databases though it is not explicitly called out. Securing the access to the databases and their contents is yet another security concern, and consequently discussed in more detail later in this guide.

4. Security Boundaries and Threats

A cloud can be abstracted as a collection of logical components by virtue of their function, users, and shared security concerns, which we call security domains. Threat actors and vectors are classified based on their motivation and access to resources. Our goal is to provide you a sense of the security concerns with respect to each domain depending on your risk/vulnerability protection objectives.

Security Domains

A security domain comprises users, applications, servers or networks that share common trust requirements and expectations within a system. Typically they have the same authentication and authorization (AuthN/Z) requirements and users.

Although you may desire to break these domains down further (we later discuss where this may be appropriate), we generally refer to four distinct security domains which form the bare minimum that is required to deploy any OpenStack cloud securely. These security domains are:

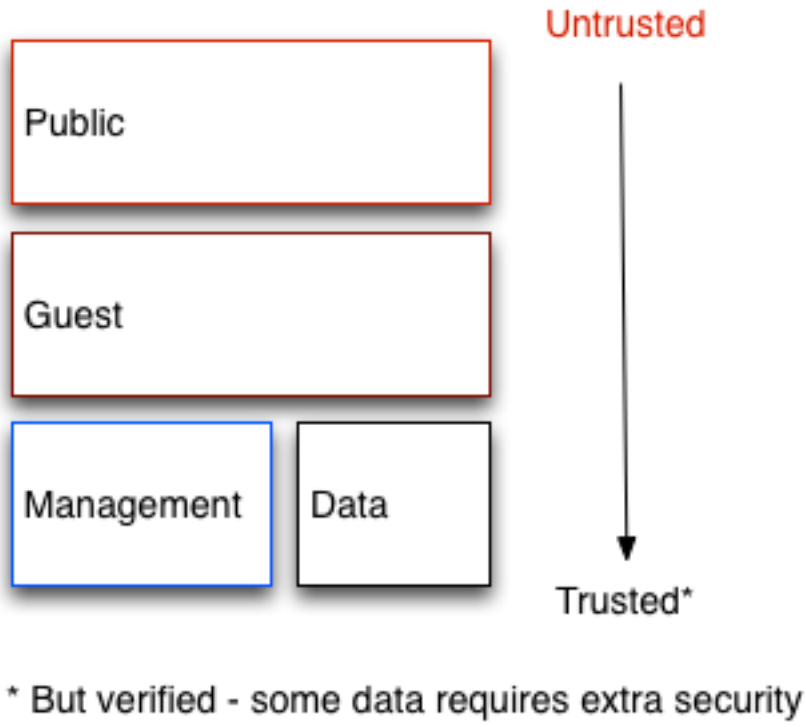
1. Public

2. Guest

3. Management

4. Data

We selected these security domains because they can be mapped independently or combined to represent the majority of the possible areas of trust within a given OpenStack deployment. For example, some deployment topologies combine both guest and data domains onto one physical network versus others, which have these networks physically separated. In each case, the cloud operator should be aware of the appropriate security concerns. Security domains should be mapped out against your specific OpenStack deployment topology. The domains and their trust requirements depend upon whether the cloud instance is public, private, or hybrid.



Public

The public security domain is an entirely untrusted area of the cloud infrastructure. It can refer to the Internet as a whole or simply to networks over which you have no authority. Any data that transits this domain with confidentiality or integrity requirements should be protected using compensating controls.

This domain should always be considered *untrusted*.

Guest

Typically used for compute instance-to-instance traffic, the guest security domain handles compute data generated by instances on the cloud but not services that support the operation of the cloud, such as API calls.

Public cloud providers and private cloud providers who do not have stringent controls on instance use or who allow unrestricted internet

access to VMs should consider this domain to be *untrusted*. Private cloud providers may want to consider this network as internal and therefore *trusted* only if they have controls in place to assert that they trust instances and all their tenants.

Management

The management security domain is where services interact. Sometimes referred to as the "control plane", the networks in this domain transport confidential data such as configuration parameters, usernames, and passwords. Command and Control traffic typically resides in this domain, which necessitates strong integrity requirements. Access to this domain should be highly restricted and monitored. At the same time, this domain should still employ all of the security best practices described in this guide.

In most deployments this domain is considered *trusted*. However, when considering an OpenStack deployment, there are many systems that bridge this domain with others, potentially reducing the level of trust you can place on this domain. See the section called “Bridging Security Domains” [17] for more information.

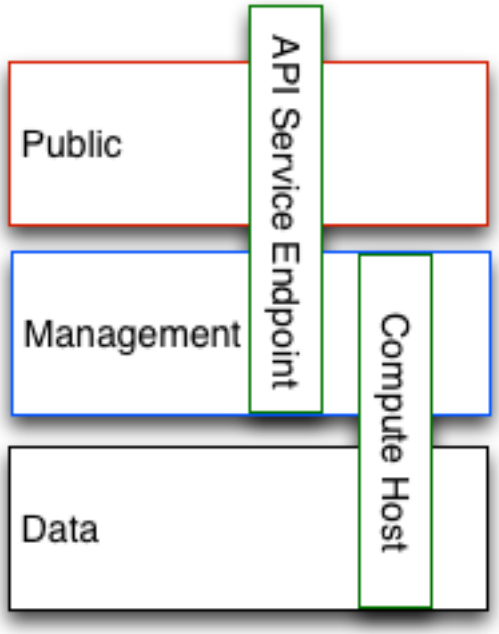
Data

The data security domain is concerned primarily with information pertaining to the storage services within OpenStack. Much of the data that crosses this network has high integrity and confidentiality requirements and depending on the type of deployment there may also be strong availability requirements.

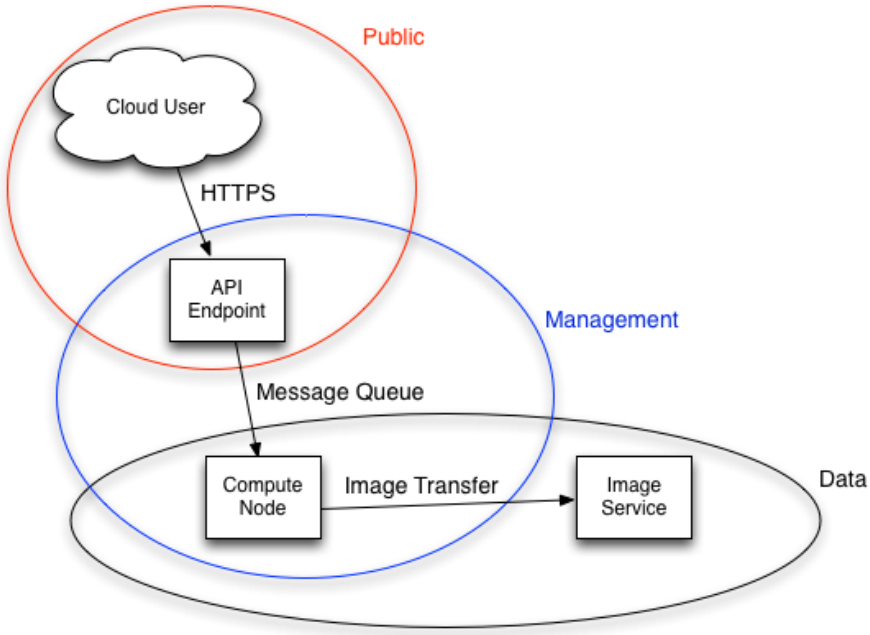
The trust level of this network is heavily dependent on deployment decisions and as such we do not assign this any default level of trust.

Bridging Security Domains

A *bridge* is a component that exists inside more than one security domain. Any component that bridges security domains with different trust levels or authentication requirements must be carefully configured. These bridges are often the weak points in network architecture. A bridge should always be configured to meet the security requirements of the highest trust level of any of the domains it is bridging. In many cases the security controls for bridges should be a primary concern due to the likelihood of attack.



The diagram above shows a compute node bridging the data and management domains, as such the compute node should be configured to meet the security requirements of the management domain. Similarly the API Endpoint in this diagram is bridging the untrusted public domain and the management domain, and should be configured to protect against attacks from the public domain propagating through to the management domain.



In some cases deployers may want to consider securing a bridge to a higher standard than any of the domains in which it resides. Given the above example of an API endpoint, an adversary could potentially target the API endpoint from the public domain, leveraging it in the hopes of compromising or gaining access to the management domain.

The design of OpenStack is such that separation of security domains is difficult - as core services will usually bridge at least two domains, special consideration must be given when applying security controls to them.

Threat Classification, Actors and Attack Vectors

Most types of cloud deployment, public or private, are exposed to some form of attack. In this chapter we categorize attackers and summarize potential types of attacks in each security domain.

Threat Actors

A threat actor is an abstract way to refer to a class of adversary that you may attempt to defend against. The more capable the actor, the

more expensive the security controls that are required for successful attack mitigation and prevention. Security is a tradeoff between cost, usability and defense. In some cases it will not be possible to secure a cloud deployment against all of the threat actors we describe here. Those deploying an OpenStack cloud will have to decide where the balance lies for their deployment / usage.

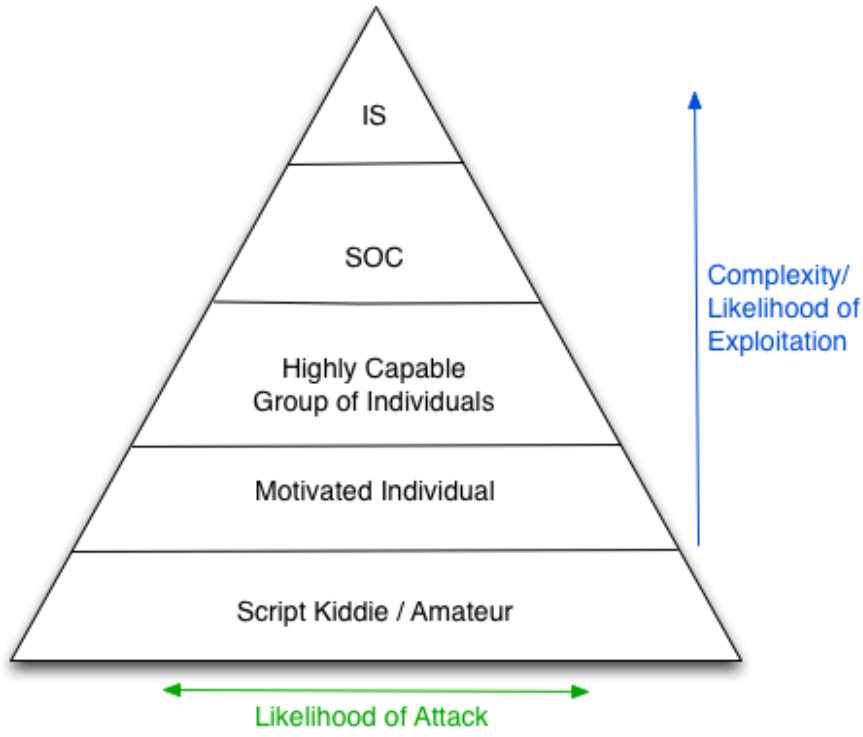
- **Intelligence Services** — Considered by this guide as the most capable adversary. Intelligence Services and other state actors can bring tremendous resources to bear on a target. They have capabilities beyond that of any other actor. It is very difficult to defend against these actors without incredibly stringent controls in place, both human and technical.

- **Serious Organized Crime** — Highly capable and financially driven groups of attackers. Able to fund in-house exploit development and target research. In recent years the rise of organizations such as the Russian Business Network, a massive cyber-criminal enterprise has demonstrated how cyber attacks have become a commodity. Industrial espionage falls within the SOC group.

- **Highly Capable Groups** — This refers to 'Hacktivist' type organizations who are not typically commercially funded but can pose a serious threat to service providers and cloud operators.

- **Motivated Individuals** — Acting alone, these attackers come in many guises, such as rogue or malicious employees, disaffected customers, or small-scale industrial espionage.

- **Script Kiddies** — Automated vulnerability scanning/exploitation. Non-targeted attacks. Often only a nuisance, compromise by one of these actors presents a major risk to an organization's reputation.



Public / Private Considerations

Private clouds are typically deployed by enterprises or institutions inside their networks and behind their firewalls. Enterprises will have strict policies on what data is allowed to exit their network and may even have different clouds for specific purposes. Users of a private cloud are typically employees of the organization that owns the cloud and are able to be held accountable for their actions. Employees often attend training sessions before accessing the cloud and will likely take part in regular scheduled security awareness training. Public clouds by contrast cannot make any assertions about their users, cloud use-cases or user motivations. This immediately pushes the guest security domain into a completely *untrusted* state for public cloud providers.

A notable difference in the attack surface of public clouds is that they must provide internet access to their services. Instance connectivity, access to files over the internet and the ability to interact with the cloud controlling fabric such as the API endpoints and dashboard are must-haves for the public cloud.

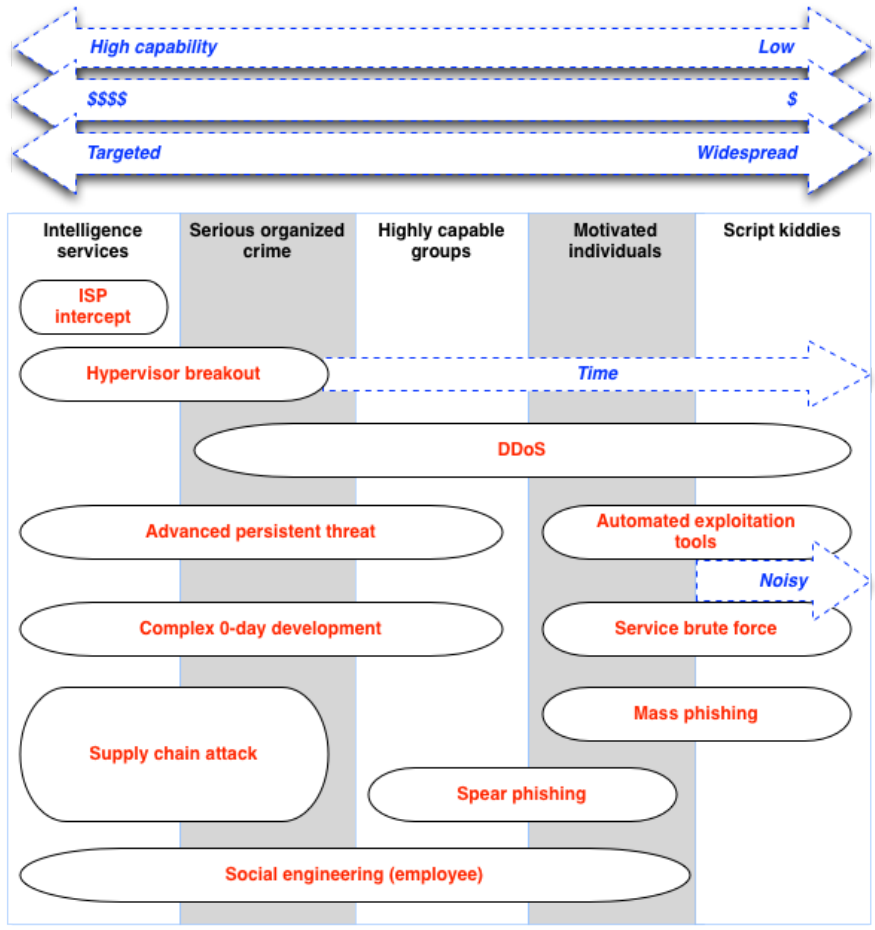
Privacy concerns for public and private cloud users are typically diametrically opposed. The data generated and stored in private clouds is normally owned by the operator of the cloud, who is able to deploy technologies such as data loss prevention (DLP) protection, file inspection, deep packet inspection and prescriptive firewalling. In contrast, privacy is one of the primary barriers to adoption for the public cloud, as many of these controls do not exist.

Outbound attacks and reputational risk

Careful consideration should be given to potential outbound abuse from a cloud deployment. Whether public or private, clouds tend to have lots of resource available. An attacker who has established a point of presence within the cloud, either through hacking in or via entitled access (rogue employee), can bring these resources to bear against the internet at large. Clouds with compute services make for ideal DDoS and brute force engines. This is perhaps a more pressing issue for public clouds as their users are largely unaccountable, and can quickly spin up numerous disposable instances for outbound attacks. Major damage can be inflicted upon a company's reputation if it becomes known for hosting malicious software or launching attacks on other networks. Methods of prevention include egress security groups, outbound traffic inspection, customer education and awareness, and fraud and abuse mitigation strategies.

Attack Types

The diagram shows the types of attacks that may be expected from the actors described in the previous section. Note that there will always be exceptions to this diagram but in general, this describes the sorts of attack that could be typical for each actor.



The prescriptive defense for each form of attack is beyond the scope of this document. The above diagram can assist you in making an informed decision about which types of threats, and threat actors, should be protected against. For commercial public cloud deployments this might include prevention against serious crime. For those deploying private clouds for government use, more stringent protective mechanisms should be in place, including carefully protected facilities and supply chains. In contrast those standing up basic development or test environments will likely require less restrictive controls (middle of the spectrum).

5. Introduction to Case Studies

Throughout this guide we will refer to two running case studies. We introduce them here and will return to them at the end of each chapter.

Case Study : Alice the private cloud builder

Alice is deploying a private cloud for use by a government department in the US. The cloud must comply with relevant standards such as FedRAMP. The security paperwork requirements for this cloud are very high. It will have no direct access to the internet, its API endpoints, compute instances and other resources will be exposed only to systems within the departments network which is entirely air-gapped from all other networks. The cloud can access other network services on the Organization's Intranet e.g. the authentication and logging services.

Case Study : Bob the public cloud provider

Bob is a lead architect for a company deploying a large greenfield public cloud. This cloud will provide IaaS for the masses, allowing any consumer with a valid credit-card access to utility computing and storage but the primary focus is enterprise customers. Data privacy concerns are a big priority for Bob as they are seen as a major barrier to large-scale adoption of the cloud by organizations.

6. System Documentation Requirements

The system documentation for an OpenStack cloud deployment should follow the templates and best practices for the Enterprise Information Technology System in your organization. Organizations often have compliance requirements which may require an overall System Security Plan to inventory and document the architecture of a given system. There are common challenges across the industry related to documenting the dynamic cloud infrastructure and keeping the information up-to-date.

System Roles & Types

It is necessary to describe the two broadly defined types of nodes that generally make up an OpenStack installation.

1. Infrastructure nodes, or the nodes that run the cloud related services such as the OpenStack Identity service, the message queuing service, storage, networking, and other services required to support the operation of the cloud.

2. The other type of nodes are compute, storage, or other resource nodes, those that provide storage capacity or virtual machines for your cloud.

System Inventory

Documentation should provide a general description of the OpenStack environment and cover all systems used (production, development, test, etc.). Documenting system components, networks, services, and software often provides the bird's-eye view needed to thoroughly cover and consider security concerns, attack vectors and possible security domain bridging points. A system inventory may need to capture ephemeral resources such as virtual machines or virtual disk volumes that would otherwise be persistent resources in a traditional IT system.

Hardware Inventory

Clouds without stringent compliance requirements for written documentation may at least benefit from having a Configuration Management Database (*CMDB*). CMDB's are normally used for hardware asset tracking and overall life-cycle management. By leveraging a CMDB, an organization can quickly identify cloud infrastructure hardware (e.g. compute nodes, storage nodes, and network devices) that exists on the network but may not be adequately protected and/or forgotten. OpenStack provisioning system may provide some CMDB-like functions especially if auto-discovery features of hardware attributes are available.

Software Inventory

Just as with hardware, all software components within the OpenStack deployment should be documented. Components here should include system databases; OpenStack software components and supporting sub-components; and, supporting infrastructure software such as load-balancers, reverse proxies, and network address translators. Having an authoritative list like this may be critical toward understanding total system impact due to a compromise or vulnerability of a specific class of software.

Network Topology

A Network Topology should be provided with highlights specifically calling out the data flows and bridging points between the security domains. Network ingress and egress points should be identified along with any OpenStack logical system boundaries. Multiple diagrams may be needed to provide complete visual coverage of the system. A network topology document should include virtual networks created on behalf of tenants by the system along with virtual machine instances and gateways created by OpenStack.

Services, Protocols and Ports

The Service, Protocols and Ports table provides important additional detail of an OpenStack deployment. A table view of all services running within the cloud infrastructure can immediately inform, guide, and help check security procedures. Firewall configuration, service port conflicts, security remediation areas, and compliance requirements become easier

to manage when you have concise information available. E.g. tabular information as shown below.

Service	Protocols	Ports	Purpose	Used By	Security Domain(s)
beam.smp	AMQP	tcp/5672	AMQP message service	RabbitMQ	MGMT
tgtd	iSCSI	tcp/3260	iSCSI initiator service	iSCSI	PRIVATE (data network)
sshd	ssh	tcp/22	allows secure login to nodes and guest VMs	Various	MGMT, GUEST and PUBLIC as configured
mysqld	mysql	tcp/3306	MySQL database service	Various	MGMT
apache2	http	tcp/443	Horizon dashboard service	Tenants	PUBLIC
dnsmasq	dns	tcp/53	DNS services	Guest VMs	GUEST

Referencing a table of services, protocols and ports can help in understanding the relationship between OpenStack components. It is highly recommended that OpenStack deployments have information similar to this on record.

7. Case Studies

In this case study we discuss how Alice and Bob would address their system documentation requirements. The documentation suggested above includes hardware and software records, network diagrams, and system configuration details.

Alice's Private Cloud

Alice needs detailed documentation to satisfy FedRamp requirements. She sets up a configuration management database (CMDB) to store information regarding all of the hardware, firmware, and software versions used throughout the cloud. She also creates a network diagram detailing the cloud architecture, paying careful attention to the security domains and the services that span multiple security domains.

Alice also needs to record each network service running in the cloud, what interfaces and ports it binds to, the security domains for each service, and why the service is needed. Alice decides to build automated tools to log into each system in the cloud over secure shell (SSH) using the Python Fabric library. The tools collect and store the information in the CMDB, which simplifies the audit process.

Bob's Public Cloud

In this case, Bob will approach these steps the same as Alice.

8. Management Introduction

A cloud deployment is a living system. Machines age and fail, software becomes outdated, vulnerabilities are discovered. When errors or omissions are made in configuration, or when software fixes must be applied, these changes must be made in a secure, but convenient, fashion. These changes are typically solved through configuration management.

Likewise, it is important to protect the cloud deployment from being configured or manipulated by malicious entities. With many systems in a cloud employing compute and networking virtualization, there are distinct challenges applicable to OpenStack which must be addressed through integrity lifecycle management.

Finally, administrators must perform command and control over the cloud for various operational functions. It is important these command and control facilities are understood and secured.

9. Continuous Systems Management

A cloud will always have bugs. Some of these will be security problems. For this reason, it is critically important to be prepared to apply security updates and general software updates. This involves smart use of configuration management tools, which are discussed below. This also involves knowing when an upgrade is necessary.

Vulnerability Management

For announcements regarding security relevant changes, subscribe to the OpenStack Announce mailing list. The security notifications are also posted through the downstream packages for example through Linux distributions that you may be subscribed to as part of the package updates.

The OpenStack components are only a small fraction of the software in a cloud. It is important to keep up to date with all of these other components, too. While the specific data sources will be deployment specific, the key idea is to ensure that a cloud administrator subscribes to the necessary mailing lists for receiving notification of any related security updates. Often this is as simple as tracking an upstream Linux distribution.

Note

OpenStack releases security information through two channels.

- OpenStack Security Advisories (OSSA) are created by the OpenStack Vulnerability Management Team (VMT). They pertain to security holes in core OpenStack services. More information on the VMT can be found here: https://wiki.openstack.org/wiki/Vulnerability_Management
- OpenStack Security Notes (OSSN) were created by the OpenStack Security Group (OSSG) to support the work of

the VMT. OSSN address issues in supporting software and common deployment configurations. They're referenced throughout this guide. Security Notes are archived at https://launchpad.net/ossn/

Triage

After receiving notification about a security update, the next step is to determine how critical this update is to a given cloud deployment. In this case, it is useful to have a pre-defined policy. Existing vulnerability rating systems such as the common vulnerability scoring system (CVSS) v2 do not properly account for cloud deployments.

In this example we introduce a scoring matrix that places vulnerabilities in three categories: Privilege Escalation, Denial of Service and Information Disclosure. Understanding the type of vulnerability and where it occurs in your infrastructure will enable you to make reasoned response decisions.

Privilege Escalation describes the ability of a user to act with the privileges of some other user in a system, bypassing appropriate authorization checks. For example a standard Linux user running code or performing an operation that allows them to conduct further operations with the privileges of the root users on the system.

Denial of Service refers to an exploited vulnerability that may cause service or system disruption. This includes both distributed attacks to overwhelm network resources, and single-user attacks that are typically caused through resource allocation bugs or input induced system failure flaws.

Information Disclosure vulnerabilities reveal information about your system or operations. These vulnerabilities range from debugging information disclosure, to exposure of critical security data, such as authentication credentials and passwords.

	Attacker Position / Privilege Level			
	External	**Cloud User**	**Cloud Admin**	**Control Plane**
Privilege Elevation (3 levels)	Critical	n/a	n/a	n/a
Privilege Elevation (2 levels)	Critical	Critical	n/a	n/a
Privilege Elevation (1 level)	Critical	Critical	Critical	n/a

Denial of Service	High	Medium	Low	Low
Information Disclosure	Critical / High	Critical / High	Medium / Low	Low

The above table illustrates a generic approach to measuring the impact of a vulnerability based on where it occurs in your deployment and the effect; for example, a single level privilege escalation on a Nova API node would potentially allow a standard user of the API to escalate to have the same privileges as the root user on the node.

We suggest cloud administrators customize and expand this table to suit their needs. Then work to define how to handle the various severity levels. For example, a critical-level security update might require the cloud to be upgraded on a specified time line, whereas a low-level update might be more relaxed.

Testing the Updates

Any update should be fully tested before deploying in a production environment. Typically this requires having a separate test cloud setup that first receives the update. This cloud should be as close to the production cloud as possible, in terms of software and hardware. Updates should be tested thoroughly in terms of performance impact, stability, application impact, and more. Especially important is to verify that the problem theoretically addressed by the update (e.g., a specific vulnerability) is actually fixed.

Deploying the Updates

Once the updates are fully tested, they can be deployed to the production environment. This deployment should be fully automated using the configuration management tools described below.

Configuration Management

A production quality cloud should always use tools to automate configuration and deployment. This eliminates human error, and allows the cloud to scale much more rapidly. Automation also helps with continuous integration and testing.

When building an OpenStack cloud it is strongly recommended to approach your design and implementation with a configuration management tool or framework in mind. Configuration management

allows you to avoid the many pitfalls inherent in building, managing, and maintaining an infrastructure as complex as OpenStack. By producing the manifests, cookbooks, or templates required for a configuration management utility, you are able to satisfy a number of documentation and regulatory reporting requirements. Further, configuration management can also function as part of your BCP and DR plans wherein you can rebuild a node or service back to a known state in a DR event or given a compromise.

Additionally, when combined with a version control system such as Git or SVN, you can track changes to your environment over time and remediate unauthorized changes that may occur. For example, a nova.conf or other configuration file falls out of compliance with your standard, your configuration management tool will be able to revert or replace the file and bring your configuration back into a known state. Finally a configuration management tool can also be used to deploy updates; simplifying the security patch process. These tools have a broad range of capabilities that are useful in this space. The key point for securing your cloud is to choose a tool for configuration management and use it.

There are many configuration management solutions; at the time of this writing there are two in the marketplace that are robust in their support of OpenStack environments: *Chef* and *Puppet*. A non-exhaustive listing of tools in this space is provided below:

- Chef
- Puppet
- Salt Stack
- Ansible

Policy Changes

Whenever a policy or configuration management is changed, it is good practice to log the activity, and backup a copy of the new set. Often, such policies and configurations are stored in a version controlled repository such as git.

Secure Backup and Recovery

It is important to include Backup procedures and policies in the overall System Security Plan. For a good overview of OpenStack's Backup and

Recovery capabilities and procedures, please refer to the OpenStack Operations Guide.

Security Considerations

- Ensure only authenticated users and backup clients have access to the backup server.
- Use data encryption options for storage and transmission of backups.
- Use a dedicated and hardened backup server(s). The backup server's logs should be monitored daily and should be accessible by only few individuals.
- Test data recovery options regularly. One of the things that can be restored from secured backups is the images. In case of a compromise, the best practice would be to terminate running instances immediately and then relaunch the instances from the images in the secured backup repository.

References

- *OpenStack Operations Guide* on backup and recovery
- http://www.sans.org/reading_room/whitepapers/backup/security-considerations-enterprise-level-backups_515
- OpenStack Security Primer

Security Auditing Tools

Security auditing tools can complement the configuration management tools. Security auditing tools automate the process of verifying that a large number of security controls are satisfied for a given system configuration. These tools help to bridge the gap from security configuration guidance documentation (for example, the STIG and NSA Guides) to a specific system installation. For example, SCAP can compare a running system to a pre-defined profile. SCAP outputs a report detailing which controls in the profile were satisfied, which ones failed, and which ones were not checked.

Combining configuration management and security auditing tools creates a powerful combination. The auditing tools will highlight deployment concerns. And the configuration management tools simplify the process

of changing each system to address the audit concerns. Used together in this fashion, these tools help to maintain a cloud that satisfies security requirements ranging from basic hardening to compliance validation.

Configuration management and security auditing tools will introduce another layer of complexity into the cloud. This complexity brings with it additional security concerns. We view this as an acceptable risk trade-off, given their security benefits. Securing the operational use of these tools is beyond the scope of this guide.

10. Integrity Life-cycle

We define integrity lifecycle as a deliberate process that provides assurance that we are always running the expected software with the expected configurations throughout the cloud. This process begins with secure bootstrapping and is maintained through configuration management and security monitoring. This chapter provides recommendations on how to approach the integrity life-cycle process.

Secure Bootstrapping

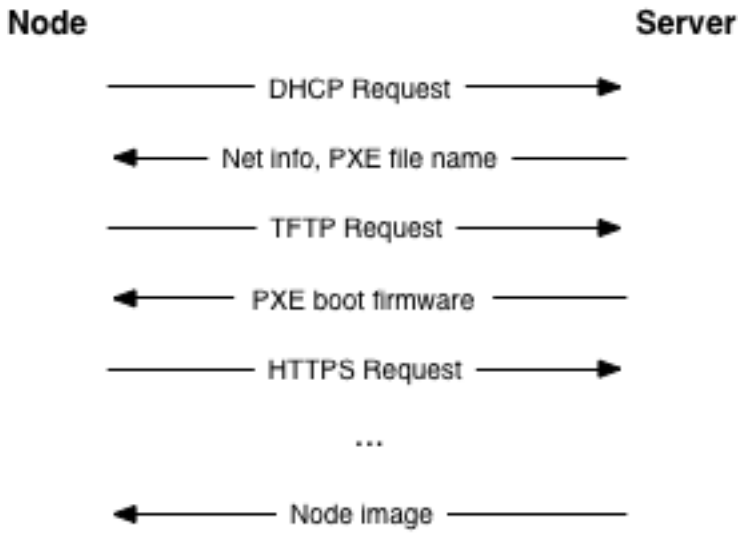
Nodes in the cloud – including compute, storage, network, service, and hybrid nodes – should have an automated provisioning process. This ensures that nodes are provisioned consistently and correctly. This also facilitates security patching, upgrading, bug fixing, and other critical changes. Since this process installs new software that runs at the highest privilege levels in the cloud, it is important to verify that the correct software is installed. This includes the earliest stages of the boot process.

There are a variety of technologies that enable verification of these early boot stages. These typically require hardware support such as the trusted platform module (TPM), Intel Trusted Execution Technology (TXT), dynamic root of trust measurement (DRTM), and Unified Extensible Firmware Interface (UEFI) secure boot. In this book, we will refer to all of these collectively as *secure boot technologies*. We recommend using secure boot, while acknowledging that many of the pieces necessary to deploy this require advanced technical skills in order to customize the tools for each environment. Utilizing secure boot will require deeper integration and customization than many of the other recommendations in this guide. TPM technology, while common in most business class laptops and desktops for several years, and is now becoming available in servers together with supporting BIOS. Proper planning is essential to a successful secure boot deployment.

A complete tutorial on secure boot deployment is beyond the scope of this book. Instead, here we provide a framework for how to integrate secure boot technologies with the typical node provisioning process. For additional details, cloud architects should refer to the related specifications and software configuration manuals.

Node Provisioning

Nodes should use Preboot eXecution Environment (PXE) for provisioning. This significantly reduces the effort required for redeploying nodes. The typical process involves the node receiving various boot stages (i.e., progressively more complex software to execute) from a server.



We recommend using a separate, isolated network within the management security domain for provisioning. This network will handle all PXE traffic, along with the subsequent boot stage downloads depicted above. Note that the node boot process begins with two insecure operations: DHCP and TFTP. Then the boot process downloads over SSL the remaining information required to deploy the node. This information might include an initramfs and a kernel. This concludes by downloading the remaining information needed to deploy the node. This may be an operating system installer, a basic install managed by Chef or Puppet, or even a complete file system image that is written directly to disk.

While utilizing SSL during the PXE boot process is somewhat more challenging, common PXE firmware projects (e.g., iPXE) provide this support. Typically this involves building the PXE firmware with knowledge of the allowed SSL certificate chain(s) so that it can properly validate the server certificate. This raises the bar for an attacker by limiting the number of insecure, plaintext network operations.

Verified Boot

In general, there are two different strategies for verifying the boot process. Traditional *secure boot* will validate the code run at each step in the process, and stop the boot if code is incorrect. *Boot attestation* will record which code is run at each step, and provide this information to another machine as proof that the boot process completed as expected. In both cases, the first step is to measure each piece of code before it is run. In this context, a measurement is effectively a SHA-1 hash of the code, taken before it is executed. The hash is stored in a platform configuration register (PCR) in the TPM.

Note: SHA-1 is used here because this is what the TPM chips support.

Each TPM has at least 24 PCRs. The TCG Generic Server Specification, v1.0, March 2005, defines the PCR assignments for boot-time integrity measurements. The table below shows a typical PCR configuration. The context indicates if the values are determined based on the node hardware (firmware) or the software provisioned onto the node. Some values are influenced by firmware versions, disk sizes, and other low-level information. Therefore, it is important to have good practices in place around configuration management to ensure that each system deployed is configured exactly as desired.

Register	What Is Measured	Context
PCR-00	Core Root of Trust Measurement (CRTM), Bios code, Host platform extensions	Hardware
PCR-01	Host Platform Configuration	Hardware
PCR-02	Option ROM Code	Hardware
PCR-03	Option ROM Configuration and Data	Hardware
PCR-04	Initial Program Loader (IPL) Code (e.g., master boot record)	Software
PCR-05	IPL Code Configuration and Data	Software
PCR-06	State Transition and Wake Events	Software
PCR-07	Host Platform Manufacturer Control	Software
PCR-08	Platform specific, often Kernel, Kernel Extensions, and Drivers	Software

PCR-09	Platform specific, often Initramfs	Software
PCR-10 to PCR-23	Platform specific	Software

At the time of this writing, very few clouds are using secure boot technologies in a production environment. As a result, these technologies are still somewhat immature. We recommend planning carefully in terms of hardware selection (e.g., ensure that you have a TPM and Intel TXT support). Then verify how the node hardware vendor populates the PCR values (e.g., which values will be available for validation). Typically the PCR values listed under the software context in the table above are the ones that a cloud architect has direct control over. But even these may change as the software in the cloud is upgraded. Configuration management should be linked into the PCR policy engine to ensure that the validation is always up to date.

Each manufacturer must provide the BIOS and firmware code for their servers. Different servers, hypervisors, and operating systems will choose to populate different PCRs. In most real world deployments, it will be impossible to validate every PCR against a known good quantity ("golden measurement"). Experience has shown that, even within a single vendor's product line, the measurement process for a given PCR may not be consistent. We recommend establishing a baseline for each server and monitoring the PCR values for unexpected changes. Third-party software may be available to assist in the TPM provisioning and monitoring process, depending upon your chosen hypervisor solution.

The initial program loader (IPL) code will most likely be the PXE firmware, assuming the node deployment strategy outlined above. Therefore, the secure boot or boot attestation process can measure all of the early stage boot code (e.g., bios, firmware, etc), the PXE firmware, and the node kernel. Ensuring that each node has the correct versions of these pieces installed provides a solid foundation on which to build the rest of the node software stack.

Depending on the strategy selected, in the event of a failure the node will either fail to boot or it can report the failure back to another entity in the cloud. For secure boot, the node will fail to boot and a provisioning service within the management security domain must recognize this and log the event. For boot attestation, the node will already be running when the failure is detected. In this case the node should be immediately quarantined by disabling its network access. Then the event should be analyzed for the root cause. In either case, policy should dictate how to proceed after a failure. A cloud may automatically attempt to reprovision a node a certain number of times. Or it may immediately notify a cloud

administrator to investigate the problem. The right policy here will be deployment and failure mode specific.

Node Hardening

At this point we know that the node has booted with the correct kernel and underlying components. There are many paths for hardening a given operating system deployment. The specifics on these steps are outside of the scope of this book. We recommend following the guidance from a hardening guide specific to your operating system. For example, the security technical implementation guides (STIG) and the NSA guides are useful starting places.

The nature of the nodes makes additional hardening possible. We recommend the following additional steps for production nodes:

- Use a read-only file system where possible. Ensure that writeable file systems do not permit execution. This can be handled through the mount options provided in `/etc/fstab`.

- Use a mandatory access control policy to contain the instances, the node services, and any other critical processes and data on the node. See the discussions on sVirt / SELinux and AppArmor below.

- Remove any unnecessary software packages. This should result in a very stripped down installation because a compute node has a relatively small number of dependencies.

Finally, the node kernel should have a mechanism to validate that the rest of the node starts in a known good state. This provides the necessary link from the boot validation process to validating the entire system. The steps for doing this will be deployment specific. As an example, a kernel module could verify a hash over the blocks comprising the file system before mounting it using dm-verity.

Runtime Verification

Once the node is running, we need to ensure that it remains in a good state over time. Broadly speaking, this includes both configuration management and security monitoring. The goals for each of these areas are different. By checking both, we achieve higher assurance that the system is operating as desired. We discuss configuration management in the management section, and security monitoring below.

Intrusion Detection System

Host-based intrusion detection tools are also useful for automated validation of the cloud internals. There are a wide variety of host-based intrusion detection tools available. Some are open source projects that are freely available, while others are commercial. Typically these tools analyze data from a variety of sources and produce security alerts based on rule sets and/or training. Typical capabilities include log analysis, file integrity checking, policy monitoring, and rootkit detection. More advanced – often custom – tools can validate that in-memory process images match the on-disk executable and validate the execution state of a running process.

One critical policy decision for a cloud architect is what to do with the output from a security monitoring tool. There are effectively two options. The first is to alert a human to investigate and/or take corrective action. This could be done by including the security alert in a log or events feed for cloud administrators. The second option is to have the cloud take some form of remedial action automatically, in addition to logging the event. Remedial actions could include anything from re-installing a node to performing a minor service configuration. However, automated remedial action can be challenging due to the possibility of false positives.

False positives occur when the security monitoring tool produces a security alert for a benign event. Due to the nature of security monitoring tools, false positives will most certainly occur from time to time. Typically a cloud administrator can tune security monitoring tools to reduce the false positives, but this may also reduce the overall detection rate at the same time. These classic trade-offs must be understood and accounted for when setting up a security monitoring system in the cloud.

The selection and configuration of a host-based intrusion detection tool is highly deployment specific. We recommend starting by exploring the following open source projects which implement a variety of host-based intrusion detection and file monitoring features.

- OSSEC
- Samhain
- Tripwire
- AIDE

Network intrusion detection tools complement the host-based tools. OpenStack doesn't have a specific network IDS built-in, but OpenStack's

networking component, Neutron, provides a plugin mechanism to enable different technologies via the Neutron API. This plugin architecture will allow tenants to develop API extensions to insert and configure their own advanced networking services like a firewall, an intrusion detection system, or a VPN between the VMs.

Similar to host-based tools, the selection and configuration of a network-based intrusion detection tool is deployment specific. Snort is the leading open source networking intrusion detection tool, and a good starting place to learn more.

There are a few important security considerations for network and host-based intrusion detection systems.

- It is important to consider the placement of the Network IDS on the cloud (e.g., adding it to the network boundary and/or around sensitive networks). The placement depends on your network environment but make sure to monitor the impact the IDS may have on your services depending on where you choose to add it. Encrypted traffic, such as SSL, cannot generally be inspected for content by a Network IDS. However, the Network IDS may still provide some benefit in identifying anomalous unencrypted traffic on the network.

- In some deployments it may be required to add host-based IDS on sensitive components on security domain bridges. A host-based IDS may detect anomalous activity by compromised or unauthorized processes on the component. The IDS should transmit alert and log information on the Management network.

11. Management Interfaces

It is necessary for administrators to perform command and control over the cloud for various operational functions. It is important these command and control facilities are understood and secured.

OpenStack provides several management interfaces for operators and tenants:

- OpenStack Dashboard (Horizon)
- OpenStack API
- Secure Shell (SSH)
- OpenStack Management Utilities (nova-manage, glance-manage, etc.)
- Out-of-Band Management Interfaces (IPMI, etc.)

OpenStack Dashboard (Horizon)

The OpenStack Dashboard (Horizon) provides administrators and tenants a web-based graphical interface to provision and access cloud-based resources. Horizon communicates with the back-end services via calls to the OpenStack API (discussed above).

Capabilities

- As a cloud administrator, the dashboard provides an overall view of the size and state of your cloud. You can create users and tenants/projects, assign users to tenant/projects and set limits on the resources available for them.
- The dashboard provides tenant-users a self-service portal to provision their own resources within the limits set by administrators.
- The dashboard provides GUI support for routers and load-balancers. For example, Horizon now implements all of the main Neutron features.

- It is an extensible *Django* web application that allows easy plug-in of third-party products and services, such as billing, monitoring, and additional management tools.
- The dashboard can also be branded for service providers and other commercial vendors.

Security Considerations

- Horizon requires cookies and JavaScript to be enabled in the web browser.
- The web server that hosts Horizon should be configured for SSL to ensure data is encrypted.
- Both the Horizon web service and the OpenStack API it uses to communicate with the back-end are susceptible to web attack vectors such as denial of service and must be monitored.
- It is now possible (though there are numerous deployment/security implications) to upload an image file directly from a user's hard disk to Glance through Horizon. For multi-GB images it is still strongly recommended that the upload be done using the Glance CLI
- Create and manage security groups through dashboard. The security groups allows L3-L4 packet filtering for security policies to protect virtual machines

References

Grizzly Release Notes

OpenStack API

The OpenStack API is a RESTful web service endpoint to access, provision and automate cloud-based resources. Operators and users typically access the API through command-line utilities (i.e. Nova, Glance, etc.), language-specific libraries, or third-party tools.

Capabilities

- To the cloud administrator the API provides an overall view of the size and state of the cloud deployment and allows the creation of users,

tenants/projects, assigning users to tenants/projects and specifying resource quotas on a per tenant/project basis.

- The API provides a tenant interface for provisioning, managing, and accessing their resources.

Security Considerations

- The API service should be configured for SSL to ensure data is encrypted.
- As a web service, OpenStack API is susceptible to familiar web site attack vectors such as denial of service attacks.

Secure Shell (SSH)

It has become industry practice to use secure shell (SSH) access for the management of Linux and Unix systems. SSH uses secure cryptographic primitives for communication. With the scope and importance of SSH in typical OpenStack deployments, it is important to understand best practices for deploying SSH.

Host Key Fingerprints

Often overlooked is the need for key management for SSH hosts. As most or all hosts in an OpenStack deployment will provide an SSH service, it is important to have confidence in connections to these hosts. It cannot be understated that failing to provide a reasonably secure and accessible method to verify SSH host key fingerprints is ripe for abuse and exploitation.

All SSH daemons have private host keys and, upon connection, offer a host key fingerprint. This host key fingerprint is the hash of an unsigned public key. It is important these host key fingerprints are known in advance of making SSH connections to those hosts. Verification of host key fingerprints is instrumental in detecting man-in-the-middle attacks.

Typically, when an SSH daemon is installed, host keys will be generated. It is necessary that the hosts have sufficient entropy during host key generation. Insufficient entropy during host key generation can result in the possibility to eavesdrop on SSH sessions.

Once the SSH host key is generated, the host key fingerprint should be stored in a secure and queriable location. One particularly convenient

solution is DNS using SSHFP resource records as defined in RFC-4255. For this to be secure, it is necessary that DNSSEC be deployed.

OpenStack Management Utilities

The OpenStack Management Utilities are open-source Python command-line clients that make API calls. There is a client for each OpenStack service (nova, glance, etc.). In addition to the standard CLI client, most of the services have a management command line which makes direct calls to the database. These dedicated management utilities are slowly being deprecated.

Security Considerations

- The dedicated management utilities (*-manage) in some cases use the direct database connection.
- Ensure that the .rc file which has your credential information is secured.

References

OpenStack End User Guide section command line clients overview

OpenStack End User Guide section Download and source the OpenStack RC file

Out-of-Band Management Interface

OpenStack management relies on out-of-band management interfaces such as the IPMI protocol to access into nodes running OpenStack components. IPMI is a very popular specification to remotely manage, diagnose and reboot servers whether the operating system is running or the system has crashed.

Security Considerations

- Use strong passwords and safeguard them, or use client-side SSL authentication.
- Ensure that the network interfaces are on their own private(management or a separate) network. Segregate management domains with firewalls or other network gear.

- If you use a web interface to interact with the *BMC*/IPMI, always use the SSL interface (e.g. https or port 443). This SSL interface should **NOT** use self-signed certificates, as is often default, but should have trusted certificates using the correctly defined fully qualified domain names (FQDNs).
- Monitor the traffic on the management network. The anomalies may be easier to track than on the busier compute nodes

Out of band management interfaces also often include graphical machine console access. It is often possible, although not necessarily default, that these interfaces are encrypted. Consult with your system software documentation for encrypting these interfaces.

References

Hacking servers that are turned off

12. Case Studies

Earlier in this chapter we discussed typical OpenStack management interfaces and associated backplane issues. We will now approach these issues by returning to our Alice and Bob case study. Specifically, we will look into how both Alice and Bob will address:

- Cloud Administration
- Self Service
- Data Replication & Recovery
- SLA & Security Monitoring.

Alice's Private Cloud

When building her private cloud, while air-gapped, Alice still needs to consider her service management interfaces. Before deploying her private cloud, Alice has completed her system documentation. Specifically she has identified which OpenStack services will exist in each security domain. From there Alice has further restricted access to management interfaces by deploying a combination of IDS, SSL encryption, and physical network isolation. Additionally, Alice requires high availability and redundant services. Thus, Alice sets up redundant infrastructure for various OpenStack API services.

Alice also needs to provide assurances that the physical servers and hypervisors have been built from a known secure state into a well-defined configuration. To enable this, Alice uses a combination of a Configuration Management platform to configure each machine according to the standards and regulations she must comply with. It will also enable Alice to report periodically on the state of her cloud and perform remediation to a known state should anything be out of the ordinary. Additionally, Alice provides hardware assurances by using a PXE system to build her nodes from a known set of base images. During the boot process, Alice provides further assurances by enabling Intel TXT and related trusted boot technologies provided by the hardware.

Bob's Public Cloud

As a public cloud provider, Bob is concerned with both the continuous availability of management interfaces and the security of transactions to the management interfaces. To that end Bob implements multiple redundant OpenStack API endpoints for the services his cloud will run. Additionally on the public network Bob uses SSL to encrypt all transactions between his customers and his cloud interfaces. To isolate his cloud operations Bob has physically isolated his management, instance migration, and storage networks.

To ease scaling and reduce management overhead Bob implements a configuration management system. For customer data assurances, Bob offers a backup as a service product as requirements will vary between customers. Finally, Bob does not provide a "baremetal" or the ability to schedule an entire node, so to reduce management overhead and increase operational efficiency Bob does not implement any node boot time security.

13. Introduction to SSL/TLS

OpenStack services receive requests on behalf of users on public networks as well as from other internal services over management networks. Inter-service communications can also occur over public networks depending on deployment and architecture choices.

While it is commonly accepted that data over public networks should be secured using cryptographic measures, such as Secure Sockets Layer or Transport Layer Security (SSL/TLS) protocols, it is insufficient to rely on security domain separation to protect internal traffic. Using a security-in-depth approach, we recommend securing all domains with SSL/TLS, including the management domain services. It is important that should a tenant escape their VM isolation and gain access to the hypervisor or host resources, compromise an API endpoint, or any other service, they must not be able to easily inject or capture messages, commands, or otherwise affect or control management capabilities of the cloud. SSL/TLS provides the mechanisms to ensure authentication, non-repudiation, confidentiality, and integrity of user communications to the OpenStack services and between the OpenStack services themselves.

Public Key Infrastructure (PKI) is the set of hardware, software, and policies to operate a secure system which provides authentication, non-repudiation, confidentiality, and integrity. The core components of PKI are:

- End Entity - user, process, or system that is the subject of a certificate
- Certification Authority (*CA*) - defines certificate policies, management, and issuance of certificates
- Registration Authority (RA) - an optional system to which a CA delegates certain management functions
- Repository - Where the end entity certificates and certificate revocation lists are stored and looked up - sometimes referred to as the "certificate bundle"

- Relying Party - The end point that is trusting that the CA is valid.

PKI builds the framework on which to provide encryption algorithms, cipher modes, and protocols for securing data and authentication. We strongly recommend securing all services with Public Key Infrastructure (PKI), including the use of SSL/TLS for API endpoints. It is impossible for the encryption or signing of transports or messages alone to solve all these problems. Hosts themselves must be secure and implement policy, namespaces, and other controls to protect their private credentials and keys. However, the challenges of key management and protection do not reduce the necessity of these controls, or lessen their importance.

Certification Authorities

Many organizations have an established Public Key Infrastructure with their own certification authority (CA), certificate policies, and management for which they should use to issue certificates for internal OpenStack users or services. Organizations in which the public security domain is Internet facing will additionally need certificates signed by a widely recognized public CA. For cryptographic communications over the management network, it is recommended one not use a public CA. Instead, we expect and recommend most deployments deploy their own internal CA.

It is recommended that the OpenStack cloud architect consider using separate PKI deployments for internal systems and customer facing services. This allows the cloud deployer to maintain control of their PKI infrastructure and among other things makes requesting, signing and deploying certificates for internal systems easier. Advanced configurations may use separate PKI deployments for different security domains. This allows deployers to maintain cryptographic separation of environments, ensuring that certificates issued to one are not recognised by another.

Certificates used to support SSL/TLS on internet facing cloud endpoints (or customer interfaces where the customer is not expected to have installed anything other than standard operating system provided certificate bundles) should be provisioned using Certificate Authorities that are installed in the operating system certificate bundle. Typical well known vendors include Verisign and Thawte but many others exist.

There are many management, policy, and technical challenges around creating and signing certificates as such is an area where cloud architects or operators may wish to seek the advice of industry leaders and vendors in addition to the guidance recommended here.

SSL/TLS Libraries

Various components, services, and applications within the OpenStack ecosystem or dependencies of OpenStack are implemented and can be configured to use SSL/TLS libraries. The SSL/TLS and HTTP services within OpenStack are typically implemented using OpenSSL which has been proven to be fairly secure and has a module that has been validated for FIPS 140-2. However, keep in mind that each application or service can still introduce weaknesses in how they use the OpenSSL libraries.

Cryptographic Algorithms, Cipher Modes, and Protocols

We recommend only using TLS v1.1 or v1.2. SSLv3 and TLSv1.0 may be used for compatibility but we recommend using caution and only enabling these protocols if you have a strong requirement to do so. Other SSL/TLS versions, explicitly older versions, should not be used. These older versions include SSLv1 and SSLv2. As this book does not intend to be a thorough reference on cryptography we do not wish to be prescriptive about what specific algorithms or cipher modes you should enable or disable in your OpenStack services. However, there are some authoritative references we would like to recommend for further information:

- National Security Agency, Suite B Cryptography
- OWASP Guide to Cryptography
- OWASP Transport Layer Protection Cheat Sheet
- SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements
- The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software
- OpenSSL and FIPS 140-2

Summary

Given the complexity of the OpenStack components and the number of deployment possibilities, you must take care to ensure that each component gets the appropriate configuration of SSL certificates, keys,

and CAs. The following services will be discussed in later sections of this book where SSL and PKI is available (either natively or possible via SSL proxy):

- Compute API endpoints
- Identity API endpoints
- Networking API endpoints
- Storage API endpoints
- Messaging server
- Database server
- Dashboard

Throughout this book we will use SSL as shorthand to refer to these recommendations for SSL/TLS protocols.

14. Case Studies

In this case study we discuss how Alice and Bob would address deployment of PKI certification authorities (CA) and certificate management.

Alice's Private Cloud

Alice as a cloud architect within a government agency knows that her agency operates its own certification authority. Alice contacts the PKI office in her agency that manages her PKI and certificate issuance. Alice obtains certificates issued by this CA and configures the services within both the public and management security domains to use these certificates. Since Alice's OpenStack deployment exists entirely on a disconnected from the Internet network, she makes sure to remove all default CA bundles that contain external public CA providers to ensure the OpenStack services only accept client certificates issued by her agency's CA.

Bob's Public Cloud

Bob is architecting a public cloud and needs to ensure that the publicly facing OpenStack services are using certificates issued by a major public CA. Bob acquires certificates for his public OpenStack services and configures the services to use PKI and SSL and includes the public CAs in his trust bundle for the services. Additionally, Bob also wants to further isolate the internal communications amongst the services within the management security domain. Bob contacts the team within his organization that is responsible for managing his organizations PKI and issuance of certificates using their own internal CA. Bob obtains certificates issued by this internal CA and configures the services that communicate within the management security domain to use these certificates and configures the services to only accept client certificates issued by his internal CA.

15. SSL Proxies and HTTP Services

OpenStack endpoints are HTTP services providing APIs to both end-users on public networks and to other OpenStack services within the same deployment operating over the management network. It is highly recommended these requests, both those internal and external, operate over SSL.

In order for API requests to be encrypted by SSL it's necessary to position the API services behind a proxy that will establish and terminate SSL sessions. The following table offers a non-exhaustive list of software services that can proxy SSL traffic for API requests:

- Pound
- Stud
- nginx
- Apache httpd
- Hardware appliance SSL acceleration proxies

It is important to be mindful of the size of requests that will be processed by any chosen SSL proxy.

Examples

Below we provide some sample configuration setting for enabling SSL in some of the most popular web servers/SSL terminators with recommended configurations. Note that we have SSL v3 enabled in some of these examples as this will be required in many deployments for client compatibility.

Pound - with AES-NI acceleration

```
## see pound(8) for details
daemon      1
######################################################################
## global options:
User        "swift"
```

```
Group       "swift"
#RootJail   "/chroot/pound"
## Logging: (goes to syslog by default)
##  0   no logging
##  1   normal
##  2   extended
##  3   Apache-style (common log format)
LogLevel    0
## turn on dynamic scaling (off by default)
# Dyn Scale 1
## check backend every X secs:
Alive       30
## client timeout
#Client     10
## allow 10 second proxy connect time
ConnTO      10
## use hardware-accelleration card supported by openssl(1):
SSLEngine   "aesni"
# poundctl control socket
Control "/var/run/pound/poundctl.socket"
######################################################################
## listen, redirect and ... to:
## redirect all swift requests on port 443 to local swift proxy
ListenHTTPS
    Address 0.0.0.0
    Port    443
    Cert    "/etc/pound/cert.pem"
    ## Certs to accept from clients
    ##  CAlist      "CA_file"
    ## Certs to use for client verification
    ##  VerifyList  "Verify_file"
    ## Request client cert - don't verify
    ##  Ciphers     "AES256-SHA"
    ## allow PUT and DELETE also (by default only GET, POST and
 HEAD)?:
    NoHTTPS11   0
    ## allow PUT and DELETE also (by default only GET, POST and
 HEAD)?:
    xHTTP       1
    Service
        BackEnd
            Address 127.0.0.1
            Port    80
        End
    End
End
```

Stud

This stud example enables SSL v3 for client compatibility. The ciphers line can be tweaked based on your needs, however this is a reasonable starting place.

```
# SSL x509 certificate file.
pem-file = "
# SSL protocol.
ssl = on
# List of allowed SSL ciphers.
# OpenSSL's high-strength ciphers which require authentication
# NOTE: This list does not include any RC4 ciphers.
ciphers = "HIGH:!aNULL:!eNULL:!DES:!3DES"
# Enforce server cipher list order
prefer-server-ciphers = on
# Number of worker processes
workers = 4
# Listen backlog size
backlog = 1000
# TCP socket keepalive interval in seconds
keepalive = 3600
# Chroot directory
chroot = ""
# Set uid after binding a socket
user = "www-data"
# Set gid after binding a socket
group = "www-data"
# Quiet execution, report only error messages
quiet = off
# Use syslog for logging
syslog = on
# Syslog facility to use
syslog-facility = "daemon"
# Run as daemon
daemon = off
# Report client address using SENDPROXY protocol for haproxy
# Disabling this until we upgrade to HAProxy 1.5
write-proxy = off
```

nginx

This nginx example requires TLS v1.1 or v1.2 for maximum security. The ssl_ciphers line can be tweaked based on your needs, however this is a reasonable starting place.

```
server {
```

```
    listen : ssl;
    ssl_certificate ;
    ssl_certificate_key ;
    ssl_protocols TLSv1.1 TLSv1.2;
    ssl_ciphers HIGH:!aNULL:!eNULL:!DES:!3DES;

    server_name _;
    keepalive_timeout 5;

    location / {

    }
}
```

Apache

```
<VirtualHost <ip address>:80>
  ServerName <site FQDN>
  RedirectPermanent / https://<site FQDN>/
</VirtualHost>
<VirtualHost <ip address>:443>
  ServerName <site FQDN>
  SSLEngine On
  SSLProtocol +SSLv3 +TLSv1 +TLSv1.1 +TLSv1.2,
  SSLCipherSuite HIGH:!aNULL:!eNULL:!DES:!3DES;
  SSLCertificateFile    /path/<site FQDN>.crt
  SSLCACertificateFile  /path/<site FQDN>.crt
  SSLCertificateKeyFile /path/<site FQDN>.key
  WSGIScriptAlias / <WSGI script location>
  WSGIDaemonProcess horizon user=<user> group=<group> processes=
3 threads=10
  Alias /static <static files location>
  <Directory <WSGI dir>>
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

Compute API SSL endpoint in Apache2, which needs to be paired with a short WSGI script.

```
<VirtualHost <ip address>:8447>
  ServerName <site FQDN>
  SSLEngine On
  SSLProtocol +SSLv3 +TLSv1 +TLSv1.1 +TLSv1.2,
  SSLCipherSuite HIGH:!aNULL:!eNULL:!DES:!3DES;
  SSLCertificateFile    /path/<site FQDN>.crt
  SSLCACertificateFile  /path/<site FQDN>.crt
  SSLCertificateKeyFile /path/<site FQDN>.key
```

```
  WSGIScriptAlias / <WSGI script location>
  WSGIDaemonProcess osapi user=<user> group=<group> processes=3
 threads=10
  <Directory <WSGI dir>>
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

HTTP Strict Transport Security

We recommend that all production deployments use HSTS. This header prevents browsers from making insecure connections after they have made a single secure one. If you have deployed your HTTP services on a public or an untrusted domain, HSTS is especially important. To enable HSTS, configure your web server to send a header like this with all requests:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

Start with a short timeout of 1 day during testing, and raise it to one year after testing has shown that you haven't introduced problems for users. Note that once this header is set to a large timeout, it is (by design) very difficult to disable.

16. API Endpoint Configuration Recommendations

This chapter provides recommendations for improving the security of both public and internal endpoints.

Internal API Communications

OpenStack provides both public facing and private API endpoints. By default, OpenStack components use the publicly defined endpoints. The recommendation is to configure these components to use the API endpoint within the proper security domain.

Services select their respective API endpoints based on the OpenStack service catalog. The issue here is these services may not obey the listed public or internal API end point values. This can lead to internal management traffic being routed to external API endpoints.

Configure Internal URLs in Keystone Catalog

The keystone catalog should be aware of your internal URLs. While this feature is not utilized by default, it may be leveraged through configuration. Additionally, it should be forward-compatible with expectant changes once this behavior becomes the default.

To register an internal URL for an endpoint:

```
$ keystone endpoint-create \
 --region RegionOne \
 --service-id=1ff4ece13c3e48d8a6461faebd9cd38f \
 --publicurl='https://public-ip:8776/v1/%(tenant_id)s' \
 --internalurl='https://management-ip:8776/v1/%(tenant_id)s' \
 --adminurl='https://management-ip:8776/v1/%(tenant_id)s'
```

Configure Applications for Internal URLs

Some services can be forced to use specific API endpoints. Therefore, it is recommended that each OpenStack service communicating to the API of

another service must be explicitly configured to access the proper internal API endpoint.

Each project may present an inconsistent way of defining target API endpoints. Future releases of OpenStack seek to resolve these inconsistencies through consistent use of the Keystone catalog.

Configuration Example #1: Nova

```
[DEFAULT]
cinder_catalog_info='volume:cinder:internalURL'
glance_protocol='https'
neutron_url='https://neutron-host:9696'
neutron_admin_auth_url='https://neutron-host:9696'
s3_host='s3-host'
s3_use_ssl=True
```

Configuration Example #2: Cinder

```
glance_host='https://glance-server'
```

Paste and Middleware

Most API endpoints and other HTTP services in OpenStack utilize the Python Paste Deploy library. This is important to understand from a security perspective as it allows for manipulation of the request filter pipeline through the application's configuration. Each element in this chain is referred to as *middleware*. Changing the order of filters in the pipeline or adding additional middleware may have unpredictable security impact.

It is not uncommon that implementors will choose to add additional middleware to extend OpenStack's base functionality. We recommend implementors make careful consideration of the potential exposure introduced by the addition of non-standard software components to their HTTP request pipeline.

Additional information on Paste Deploy may be found at http://pythonpaste.org/deploy/.

API Endpoint Process Isolation & Policy

API endpoint processes, especially those that reside within the public security domain should be isolated as much as possible. Where

deployments allow, API endpoints should be deployed on separate hosts for increased isolation.

Namespaces

Many operating systems now provide compartmentalization support. Linux supports namespaces to assign processes into independent domains. System compartmentalization is covered in more detail in other parts of the guide.

Network Policy

API endpoints typically bridge multiple security domains, as such particular attention should be paid to the compartmentalization of the API processes. See the *Security Domain Bridging* section for additional information in this area.

With careful modeling, network ACLs and IDS technologies can be use to enforce explicit point to point communication between network services. As critical cross domain service, this type of explicit enforcement works well for OpenStack's message queue service.

Policy enforcement can be implemented through the configuration of services, host-based firewalls (such as IPTables), local policy (SELinux or AppArmor), and optionally enforced through global network policy.

Mandatory Access Controls

API endpoint processes should be isolated from each other and other processes on a machine. The configuration for those processes should be restricted to those processes not only by Discretionary Access Controls, but through Mandatory Access Controls. The goal of these enhanced access control is to aid in the containment and escalation of API endpoint security breaches. With mandatory access controls, such breaches will severely limit access to resources and provide earlier alerting on such events.

17. Case Studies

In this case study we discuss how Alice and Bob would address endpoint configuration to secure their private and public clouds. Alice's cloud is not publicly accessible, but she is still concerned about securing the endpoints against improper use. Bob's cloud, being public, must take measures to reduce the risk of attacks by external adversaries.

Alice's Private Cloud

Alice's organization requires that the security architecture protect the access to the public and private endpoints, so she elects to use the Apache SSL proxy on both public and internal services. Alice's organization has implemented its own certificate authority. Alice contacts the PKI office in her agency that manages her PKI and certificate issuance. Alice obtains certificates issued by this CA and configures the services within both the public and management security domains to use these certificates. Since Alice's OpenStack deployment exists entirely on a disconnected from the Internet network, she makes sure to remove all default CA bundles that contain external public CA providers to ensure the OpenStack services only accept client certificates issued by her agency's CA. Alice has registered all of the services in the Keystone Services Catalog, using the internal URLs for access by internal services. She has installed host-based intrusion detection on all of the API endpoints.

Bob's Public Cloud

Bob must also protect the access to the public and private endpoints, so he elects to use the Apache SSL proxy on both public and internal services. On the public services, he has configured the certificate key files with certificates signed by a well-known Certificate Authority. He has used his organization's self-signed CA to sign certificates in the internal services on the Management network. Bob has registered his services in the Keystone Services Catalog, using the internal URLs for access by internal services. Bob's public cloud runs services on SELinux, which he has configured with a mandatory access control policy to reduce the impact of any publicly accessible services that may be compromised. He has also configured the endpoints with a host-based IDS.

18. Identity

The OpenStack Identity Service (Keystone) supports multiple methods of authentication, including username & password, LDAP, and external authentication methods. Upon successful authentication, Keystone provides the user with an authorization token used for subsequent service requests.

Transport Layer Security TLS/SSL provides authentication between services and persons using X.509 certificates. Although the default mode for SSL is server-side only authentication, certificates may also be used for client authentication.

Authentication

Invalid Login Attempts

Keystone does not provide a method to limit access to accounts after repeated unsuccessful login attempts. Repeated failed login attempts are likely brute-force attacks (Refer figure Attack-types). This is a more significant issue in Public clouds.

Prevention is possible by using an external authentication system that blocks out an account after some configured number of failed login attempts. The account then may only be unlocked with further side-channel intervention.

If prevention is not an option, detection can be used to mitigate damage.Detection involves frequent review of access control logs to identify unauthorized attempts to access accounts. Possible remediation would include reviewing the strength of the user password, or blocking the network source of the attack via firewall rules. Firewall rules on the keystone server that restrict the number of connections could be used to reduce the attack effectiveness, and thus dissuade the attacker.

In addition, it is useful to examine account activity for unusual login times and suspicious actions, with possibly disable the account. Often times this approach is taken by credit card providers for fraud detection and alert.

Multi-factor Authentication

Employ multi-factor authentication for network access to privileged user accounts. Keystone supports external authentication services through the Apache web server that can provide this functionality. Servers may also enforce client-side authentication using certificates.

This recommendation provides insulation from brute force, social engineering, and both spear and mass phishing attacks that may compromise administrator passwords.

Authentication Methods

Internally Implemented Authentication Methods

Keystone can store user credentials in an SQL Database, or may use an LDAP-compliant directory server. The Keystone database may be separate from databases used by other OpenStack services to reduce the risk of a compromise of the stored credentials.

When authentication is provided via username and password, Keystone does not enforce policies on password strength, expiration, or failed authentication attempts as recommended by NIST Special Publication 800-118 (draft). Organizations that desire to enforce stronger password policies should consider using Keystone Identity Service Extensions or external authentication services.

LDAP simplifies integration of Keystone authentication into an organization's existing directory service and user account management processes.

Authentication and authorization policy in OpenStack may be delegated to an external LDAP server. A typical use case is an organization that seeks to deploy a private cloud and already has a database of employees, the users. This may be in an LDAP system. Using LDAP as a source of authority authentication, requests to Keystone are delegated to the LDAP service, which will authorize or deny requests based on locally set policies. A token is generated on successful authentication.

Note that if the LDAP system has attributes defined for the user such as admin, finance, HR etc, these must be mapped into roles and groups within Keystone for use by the various OpenStack services. The *etc/keystone.conf* file provides the mapping from the LDAP attributes to Keystone attributes.

Keystone **MUST NOT** be allowed to write to LDAP services used for authentication outside of the OpenStack deployment as this would allow a sufficiently privileged keystone user to make changes to the LDAP directory. This would allow privilege escalation within the wider organization or facilitate unauthorized access to other information and resources. In such a deployment, user provisioning would be out of the realm of the OpenStack deployment.

Note

There is an OpenStack Security Note (OSSN) regarding keystone.conf permissions.

There is an OpenStack Security Note (OSSN) regarding potential DoS attacks.

External Authentication Methods

Organizations may desire to implement external authentication for compatibility with existing authentication services or to enforce stronger authentication policy requirements. Although passwords are the most common form of authentication, they can be compromised through numerous methods, including keystroke logging and password compromise. External authentication services can provide alternative forms of authentication that minimize the risk from weak passwords.

These include:

- Password Policy Enforcement: Requires user passwords to conform to minimum standards for length, diversity of characters, expiration, or failed login attempts.
- Multi-factor authentication: The authentication service requires the user to provide information based on something they have (e.g., a one-time password token or X.509 certificate) and something they know (e.g., a password).
- Kerberos

Authorization

Keystone supports the notion of groups and roles. Users belong to groups. A group has a list of roles. OpenStack services reference the roles of the user attempting to access the service. The OpenStack policy enforcer middleware takes into consideration the policy rule associated with each resource and the user's group/roles and tenant association to determine if he/she has access to the requested resource.

The Policy enforcement middleware enables fine-grained access control to OpenStack resources. Only admin users can provision new users and have access to various management functionality. The cloud tenant would be able to only spin up instances, attach volumes, etc.

Establish Formal Access Control Policies

Prior to configuring roles, groups, and users, document your required access control policies for the OpenStack installation. The policies should be consistent with any regulatory or legal requirements for the organization. Future modifications to access control configuration should be done consistently with the formal policies. The policies should include the conditions and processes for creating, deleting, disabling, and enabling accounts, and for assigning privileges to the accounts. Periodically review the policies and ensure that configuration is in compliance with approved policies.

Service Authorization

As described in the *OpenStack Cloud Administrator Guide*, cloud administrators must define a user for each service, with a role of Admin. This service user account provides the service with the authorization to authenticate users.

The Nova and Swift services can be configured to use either the "tempAuth" file or Keystone to store authentication information. The "tempAuth" solution MUST NOT be deployed in a production environment since it stores passwords in plain text.

Keystone supports client authentication for SSL which may be enabled. SSL client authentication provides an additional authentication factor, in addition to the username / password, that provides greater reliability on user identification. It reduces the risk of unauthorized access when usernames and passwords may be compromised. However, there is

additional administrative overhead and cost to issue certificates to users that may not be feasible in every deployment.

NOTE: We recommend using client authentication using SSL for the authentication of services to Keystone.

The cloud administrator should protect sensitive configuration files for unauthorized modification. This can be achieved with mandatory access control frameworks such as SELinux, including `/etc/keystone.conf` and X.509 certificates.

Administrative Users

We recommend that admin users authenticate using Keystone and an external authentication service that supports 2-factor authentication, such as a certificate. This reduces the risk from passwords that may be compromised. This recommendation is in compliance with NIST 800-53 IA-2(1) guidance in the use of multifactor authentication for network access to privileged accounts.

End Users

Keystone can directly provide end-user authentication, or can be configured to use external authentication methods to conform to an organization's security policies and requirements.

Policies

Each OpenStack service has a policy file in json format, called **policy.json**. The policy file specifies rules, and the rule that governs each resource. A resource could be API access, the ability to attach to a volume, or to fire up instances.

The policies can be updated by the cloud administrator to further control access to the various resources. The middleware could also be further customized. Note that your users must be assigned to groups/roles that you refer to in your policies.

Below is a snippet of the Cinder service policy.json file.

```
{
    "context_is_admin": [["role:admin"]],
    "admin_or_owner":  [["is_admin:True"], ["project_id:
%(project_id)s"]],
```

```
    "default": [["rule:admin_or_owner"]],

    "admin_api": [["is_admin:True"]],

    "volume:create": [],
    "volume:get_all": [],
    "volume:get_volume_metadata": [],
    "volume:get_snapshot": [],
    "volume:get_all_snapshots": [],

    "volume_extension:types_manage": [["rule:admin_api"]],
    "volume_extension:types_extra_specs": [["rule:admin_api"]],
  ...
}
```

Note the **default** rule specifies that the user must be either an admin or the owner of the volume. It essentially says only the owner of a volume or the admin may create/delete/update volumes. Certain other operations such as managing volume types are accessible only to admin users.

Tokens

Once a user is authenticated, a token is generated and used internally in OpenStack for authorization and access. The default token **lifespan** is **24 hours**. It is recommended that this value be set lower but caution needs to be taken as some internal services will need sufficient time to complete their work. The cloud may not provide services if tokens expire too early. An example of this would be the time needed by Nova to transfer a disk image onto the hypervisor for local caching.

The Identity service could alternatively be configured to provide UUID tokens which are significantly shorter but may be less secure depending on your specific deployment model. Decisions about token implementation should take into consideration the level of trust needed within a given security domain.

Below is an example of a PKI token. Note that, in practice, the token id value is very long (e.g., around 3500 bytes), but for brevity we shorten it in this example.

```
       "token": {
           "expires": "2013-06-26T16:52:50Z",
           "id": "MIIKXAY...",
           "issued_at": "2013-06-25T16:52:50.622502",
           "tenant": {
               "description": null,
```

```
                "enabled": true,
                "id": "912426c8f4c04fb0a07d2547b0704185",
                "name": "demo"
            }
        }
```

Note that the token is often passed within the structure of a larger context of a Keystone response. These responses also provide a catalog of the various OpenStack services. Each service is listed with its name, access endpoints for internal, admin, and public access.

Keystone supports token revocation. This manifests as an API to revoke a token, to list revoked tokens and individual OpenStack services that cache tokens to query for the revoked tokens and remove them from their cache and append the same to their list of cached revoked tokens.

Future

Domains are high-level containers for projects, users and groups. As such, they can be used to centrally manage all Keystone-based identity components. With the introduction of account Domains, server, storage and other resources can now be logically grouped into multiple Projects (previously called Tenants) which can themselves be grouped under a master account-like container. In addition, multiple users can be managed within an account Domain and assigned roles that vary for each Project.

Keystone's V3 API supports multiple domains. Users of different domains may be represented in different authentication backends and even have different attributes that must be mapped to a single set of roles and privileges, that are used in the policy definitions to access the various service resources.

Where a rule may specify access to only admin users and users belonging to the tenant, the mapping may be trivial. In other scenarios the cloud administrator may need to approve the mapping routines per tenant.

19. Dashboard

Horizon is the OpenStack dashboard, providing access to a majority of the capabilities available in OpenStack. These include provisioning users, defining instance flavors, uploading VM images, managing networks, setting up security groups, starting instances, and accessing the instances via a console.

Horizon is based on the Django web framework, therefore secure deployment practices for Django apply directly to Horizon. This guide provides a popular set of Django security recommendations, further information can be found by reading the Django deployment and security documentation.

Horizon ships with reasonable default security settings, and has good deployment and configuration documentation.

Basic Web Server Configuration

Horizon should be deployed as a Web Services Gateway Interface (WSGI) application behind an HTTPS proxy such as Apache or nginx. If Apache is not already in use, we recommend nginx since it is lighter weight and easier to configure correctly.

When using nginx, we recommend gunicorn as the wsgi host with an appropriate number of synchronous workers. We strongly advise against deployments using fastcgi, scgi, or uWSGI. We strongly advise against the use of synthetic performance benchmarks when choosing a wsgi server.

When using Apache, we recommend mod_wsgi to host Horizon.

HTTPS

Horizon should be deployed behind a secure HTTPS server using a valid, trusted certificate from a recognized certificate authority (CA). Private organization-issued certificates are only appropriate when the root of trust is pre-installed in all user browsers.

HTTP requests to the Horizon domain should be configured to redirect to the fully qualified HTTPS URL.

HTTP Strict Transport Security (HSTS)

It is highly recommended to use HTTP Strict Transport Security (HSTS).

NOTE: If you are using an HTTPS proxy in front of your web server, rather than using an HTTP server with HTTPS functionality, follow the Django documentation on modifying the SECURE_PROXY_SSL_HEADER variable.

See the chapter on PKI/SSL Everywhere for more specific recommendations and server configurations for HTTPS configurations, including the configuration of HSTS.

Frontend Caching

Since Horizon is rendering dynamic content passed directly from OpenStack API requests, we do not recommend frontend caching layers such as varnish. In Django, static media is directly served from Apache or nginx and already benefits from web host caching.

Domain Names

Many organizations typically deploy web applications at subdomains of an overarching organization domain. It is natural for users to expect a domain of the form openstack.example.org. In this context, there are often many other applications deployed in the same second-level namespace, often

serving user-controlled content. This name structure is convenient and simplifies nameserver maintenance.

We strongly recommend deploying horizon to a *second-level domain*, for example `https://example.com`, and advise against deploying horizon on a *shared subdomain* of any level, for example `https://openstack.example.org` or `https://horizon.openstack.example.org`. We also advise against deploying to bare internal domains like `https://horizon/`.

This recommendation is based on the limitations browser same-origin-policy. The recommendations in this guide cannot effectively protect users against known attacks if Horizon is deployed on a domain which also hosts user-generated content (e.g. scripts, images, uploads of any kind) even if the user-generated content is on a different subdomain. This approach is used by most major web presences (e.g. googleusercontent.com, fbcdn.com, github.io, twimg.com) to ensure that user generated content stays separate from cookies and security tokens.

Additionally, if you decline to follow this recommendation above about second-level domains, it is vital that you avoid the cookie backed session store and employ HTTP Strict Transport Security (HSTS). When deployed on a subdomain, Horizon's security is only as strong as the weakest application deployed on the same second-level domain.

Static Media

Horizon's static media should be deployed to a subdomain of the Horizon domain and served by the web server. The use of an external content delivery network (CDN) is also acceptable. This subdomain should not set cookies or serve user-provided content. The media should also be served with HTTPS.

Django media settings are documented at https://docs.djangoproject.com/en/1.5/ref/settings/#static-root.

Horizon's default configuration uses django_compressor to compress and minify css and JavaScript content before serving it. This process should be statically done before deploying Horizon, rather than using the default in-request dynamic compression and copying the resulting files along with deployed code or to the CDN server. Compression should be done in a non-production build environment. If this is not practical, we recommend disabling resource minification entirely. Online compression dependencies (less, nodejs) should not be installed on production machines.

Secret Key

Horizon depends on a shared SECRET_KEY setting for some security functions. It should be a randomly generated string at least 64 characters long. It must be shared across all active Horizon instances. Compromise of this key may allow a remote attacker to execute arbitrary code. Rotating this key invalidates existing user sessions and caching. Do not commit this key to public repositories.

Session Backend

Horizon's default session backend (*django.contrib.sessions.backends.signed_cookies*) stores user data in *signed* but *unencrypted* cookies stored in the browser. This approach allows the most simple session backend scaling since each Horizon instance is stateless, but it comes at the cost of *storing sensitive access tokens in the client browser* and transmitting them with every request. This backend ensures that session data has not been tampered with, but the data itself is not encrypted other than the encryption provided by HTTPS.

If your architecture allows it, we recommend using *django.contrib.sessions.backends.cache* as your session backend with memcache as the cache. Memcache must not be exposed publicly, and should communicate over a secured private channel. If you choose to use the signed cookies backend, refer to the Django documentation understand the security tradeoffs.

For further details, consult the Django session backend documentation.

Allowed Hosts

Configure the ALLOWED_HOSTS setting with the domain or domains where Horizon is available. Failure to configure this setting (especially if not following the recommendation above regarding second level domains) opens Horizon to a number of serious attacks. Wildcard domains should be avoided.

For further details, see the Django documentation on settings.

Cookies

Session Cookies should be set to HTTPONLY:

```
SESSION_COOKIE_HTTPONLY = True
```

Never configure CSRF or session cookies to have a wildcard domain with a leading dot. Horizon's session and CSRF cookie should be secured when deployed with HTTPS:

```
Code CSRF_COOKIE_SECURE = True
SESSION_COOKIE_SECURE = True
```

Password Auto Complete

We recommend that implementers do not change the default password autocomplete behavior. Users choose stronger passwords in environments that allow them to use the secure browser password manager. Organizations which forbid the browser password manager should enforce this policy at the desktop level.

Cross Site Request Forgery (CSRF)

Django has a dedicated middleware for cross-site request forgery (CSRF).

Horizon is designed to discourage developers from introducing cross-site scripting vulnerabilities with custom dashboards. However, it is important to audit custom dashboards, especially ones that are javascript-heavy for inappropriate use of the @csrf_exempt decorator. Dashboards which do not follow these recommended security settings should be carefully evaluated before restrictions are relaxed.

Cross Site Scripting (XSS)

Unlike many similar systems, OpenStack Horizon allows the entire unicode character set in most fields. This means developers have less latitude to make escaping mistakes that open attack vectors for cross-site scripting (XSS).

Horizon provides tools for developers to avoid creating XSS vulnerabilities, but they only work if developers use them correctly. Audit any custom dashboards, paying particular attention to use of the mark_safe function, use of is_safe with custom template tags, the safe template tag, anywhere autoescape is turned off, and any javascript which might evaluate improperly escaped data.

Cross Origin Resource Sharing (CORS)

Configure your web server to send a restrictive CORS header with each response, allowing only the Horizon domain and protocol:

```
Access-Control-Allow-Origin: https://example.com/
```

Never allow the wildcard origin.

Horizon Image Upload

We recommend that implementers disable HORIZON_IMAGES_ALLOW_UPLOAD unless they have implemented a plan to prevent resource exhaustion and denial of service.

Upgrading

Django security releases are generally well tested and aggressively backwards compatible. In almost all cases, new major releases of Django are also fully backwards compatible with previous releases. Horizon implementers are strongly encouraged to run the latest stable release of Django with up-to-date security releases.

Debug

Make sure DEBUG is set to False in production. In Django, DEBUG displays stack traces and sensitive web server state information on any exception.

20. Compute

The compute service (Nova) is one of the more complex OpenStack services. It runs in many locations throughout the cloud and interacts with a variety of internal services. For this reason, most of our recommendations regarding best practices for Nova configuration are distributed throughout this book. We provide specific details in the sections on Management, API Endpoints, Messaging, and Database.

Virtual Console Selection

One decision a cloud architect will need to make regarding Nova configuration is whether to use VNC or SPICE. Below we provide some details on the differences between these options.

Virtual Network Computer (VNC)

OpenStack can be configured to provide remote desktop console access to instances for tenants and/or administrators using the Virtual Network Computer (VNC) protocol.

Capabilities

- The OpenStack Dashboard (Horizon) can provide a VNC console for instances directly on the web page using the HTML5 noVNC client. This requires the `nova-novncproxy` service to bridge from the public network to the management network.
- The nova command line utility can return a URL for the VNC console for access by the nova Java VNC client. This requires the nova-xvpvncproxy service to bridge from the public network to the management network.

Security Considerations

- The `nova-novncproxy`and nova-xvpvncproxy services by default open public-facing ports that are token authenticated.
- By default, the remote desktop traffic is not encrypted. Havana is expected to have VNC connections secured by Kerberos.

References

Secure Connections to VNC ports

Simple Protocol for Independent Computing Environments (SPICE)

As an alternative to VNC, OpenStack provides remote desktop access to guest virtual machines using the Simple Protocol for Independent Computing Environments (SPICE) protocol.

Capabilities

- SPICE is supported by the OpenStack Dashboard (Horizon) directly on the instance web page. This requires the nova-spicehtml5proxy service.
- The nova command line utility can return a URL for SPICE console for access by a SPICE-html client.

Limitations

- Although SPICE has many advantages over VNC, the spice-html5 browser integration currently doesn't really allow admins to take advantage of any of the benefits. To take advantage of SPICE features like multi-monitor, USB pass through, etc. admins are recommended to use a standalone SPICE client within the Management Network.

Security Considerations

- The nova-spicehtml5proxy service by default opens public-facing ports that are token authenticated.
- The functionality and integration are still evolving. We will access the features in the next release and make recommendations.
- As is the case for VNC, at this time we recommend using SPICE from the management network in addition to limiting use to few individuals.

References

SPICE Console

Red Hat bug 913607

SPICE support in RDO Grizzly

21. Storage

In the Protection of Tenant Data section we discuss a variety of issues related to storage security. However, due to the time constraints with the book sprint that created the original version of this book, we do not have specific guidance related to configuration of the storage projects (e.g., Swift, Cinder, and Glance).

We encourage community contributions to help improve this chapter for a future version.

22. Case Studies

In this case study we discuss how Alice and Bob would address configuration of OpenStack core services. These include the Keystone Identity service, Dashboard, and Compute services. Alice will be concerned with integration into the existing government directory services, while Bob will need to provide access to the public.

Alice's Private Cloud

Alice's enterprise has a well-established directory service with two-factor authentication for all users. She configures Keystone to support an external authentication service supporting authentication with government-issued access cards. She also uses an external LDAP server to provide role information for the users that is integrated with the access control policy. Due to FedRAMP compliance requirements, Alice implements two-factor authentication on the Management network for all administrator access.

Alice also deploys the Dashboard to manage many aspects of the cloud. She deploys the Dashboard with HSTS to ensure that only HTTPS is used. The Dashboard resides within an internal subdomain of the private network domain name system.

Alice decides to use SPICE instead of VNC for the virtual console. She wants to take advantage of the emerging capabilities in SPICE.

Bob's Public Cloud

Bob must support authentication by the general public, so he elects to use provide for username / password authentication. He has concerns about brute force attacks attempting to crack user passwords, so he also uses an external authentication extension that throttles the number of failed login attempts. Bob's Management network is separate from the other networks within his cloud, but can be reached from his corporate network via ssh. As recommended earlier, Bob requires administrators to use two-factor authentication on the Management network to reduce the risk from compromised administrator passwords.

Bob also deploys the Dashboard to manage many aspects of the cloud. He deploys the Dashboard with HSTS to ensure that only HTTPS is used. He has ensured that the Dashboard is deployed on a second-level domain due to the limitations of the same-origin policy. He also disables HORIZON_IMAGES_ALLOW_UPLOAD to prevent resource exhaustion.

Bob decides to use VNC for his virtual console for its maturity and security features.

23. State of Networking

OpenStack Networking in the Grizzly release enables the end-user or tenant to define, utilize, and consume networking resources in new ways that had not been possible in previous OpenStack Networking releases. OpenStack Networking provides a tenant-facing API for defining network connectivity and IP addressing for instances in the cloud in addition to orchestrating the network configuration. With the transition to an API-centric networking service, cloud architects and administrators should take into consideration best practices to secure physical and virtual network infrastructure and services.

OpenStack Networking was designed with a plug-in architecture that provides extensibility of the API via open source community or third-party services. As you evaluate your architectural design requirements, it is important to determine what features are available in OpenStack Networking core services, any additional services that are provided by third-party products, and what supplemental services are required to be implemented in the physical infrastructure.

This section is a high-level overview of what processes and best practices should be considered when implementing OpenStack Networking. We will talk about the current state of services that are available, what future services will be implemented, and the current limitations in this project.

24. Networking Architecture

OpenStack Networking is a standalone service that often involves deploying several processes across a number of nodes. These processes interact with each other and with other OpenStack services. The main process of the OpenStack Networking service is neutron-server, a Python daemon that exposes the OpenStack Networking API and passes tenant requests to a suite of plugins for additional processing.

OpenStack Networking components encompasses the following elements:

- **neutron server** (`neutron-server` and `neutron-*-plugin`): This service runs on the network node to service the Networking API and its extensions. It also enforces the network model and IP addressing of each port. The neutron-server and plugin agents require access to a database for persistent storage and access to a message queue for inter-communication.

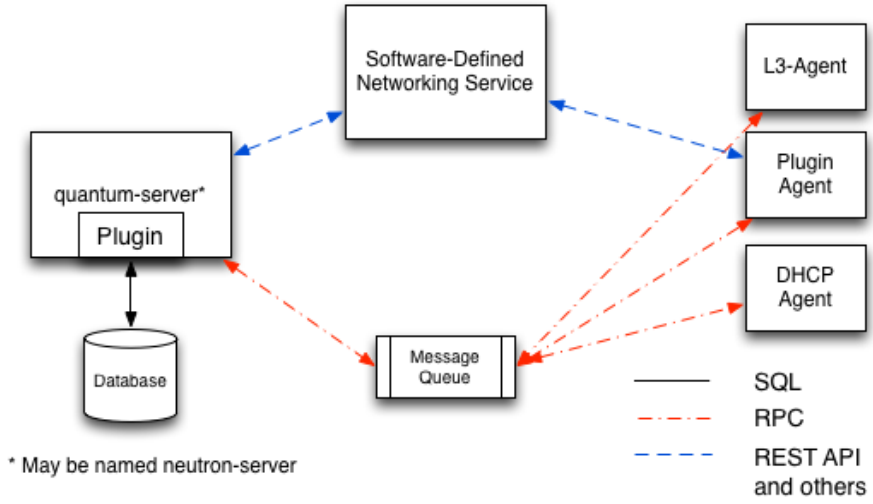
- **plugin agent** (`neutron-*-agent`): Runs on each Nova compute node to manage local virtual switch (vswitch) configuration. The agents to be run will depend on which plugin you are using. This service requires message queue access. *Optional depending on plugin.*

- **DHCP agent** (`neutron-dhcp-agent`): Provides DHCP services to tenant networks. This agent is the same across all plugins and is responsible for maintaining DHCP configuration. The neutron-dhcp-agent requires message queue access.

- **l3 agent** (`neutron-l3-agent`): Provides L3/NAT forwarding for external network access of VMs on tenant networks. Requires message queue access. *Optional depending on plugin.*

- **network provider services** (SDN server/services). Provide additional networking services that are provided to tenant networks. These SDN services may interact with the neutron-server, neutron-plugin, and/or plugin-agents via REST APIs or other communication channels.

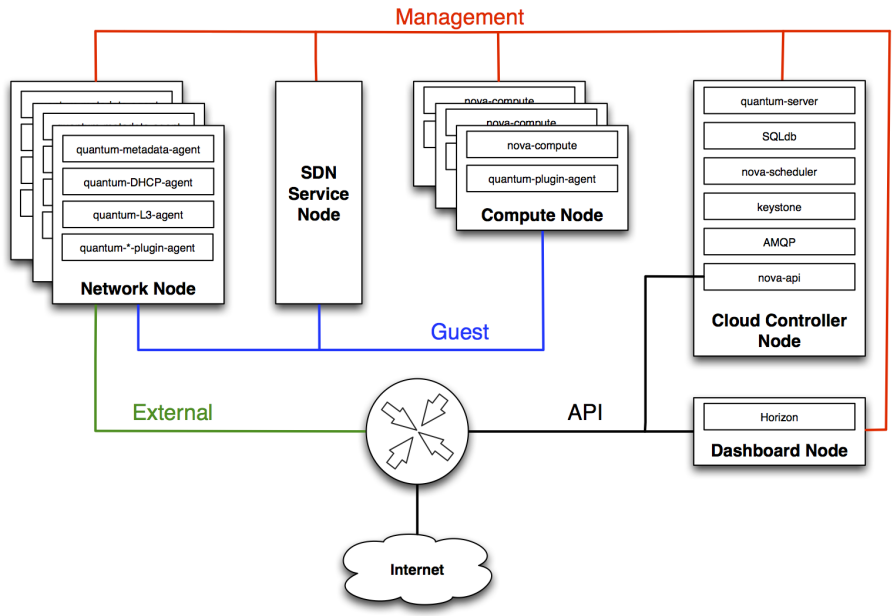
The figure that follows provides an architectural and networking flow diagram of the OpenStack Networking components:



OS Networking Service placement on Physical Servers

In this guide, we focus primarily on a standard architecture that includes a *cloud controller* host, a *network* host, and a set of *compute* hypervisors for running VMs.

Network Connectivity of Physical Servers



A standard OpenStack Networking setup has up to four distinct physical data center networks:

- **Management network** Used for internal communication between OpenStack Components. The IP addresses on this network should be reachable only within the data center and is considered the Management Security Domain.

- **Guest network** Used for VM data communication within the cloud deployment. The IP addressing requirements of this network depend on the OpenStack Networking plugin in use and the network configuration choices of the virtual networks made by the tenant. This network is considered the Guest Security Domain.

- **External network** Used to provide VMs with Internet access in some deployment scenarios. The IP addresses on this network should be reachable by anyone on the Internet and is considered to be in the Public Security Domain.

- **API network** Exposes all OpenStack APIs, including the OpenStack Networking API, to tenants. The IP addresses on this network should be reachable by anyone on the Internet. This may be the same network as

the external network, as it is possible to create a subnet for the external network that uses IP allocation ranges to use only less than the full range of IP addresses in an IP block. This network is considered the Public Security Domain.

For additional information see the Networking chapter in the *OpenStack Cloud Administrator Guide*.

25. Networking Services

In the initial architectural phases of designing your OpenStack Network infrastructure it is important to ensure appropriate expertise is available to assist with the design of the physical networking infrastructure, to identify proper security controls and auditing mechanisms.

OpenStack Networking adds a layer of virtualized network services - giving tenants the capability to architect their own, virtual networks. These virtualized services are not as currently as mature as their traditional networking counterparts. It is important to be aware of the current state of these virtualized services and what controls may need to be implemented at the virtualized and traditional network boundary.

L2 Isolation using VLANs and Tunneling

OpenStack networking can employ two different mechanisms for traffic segregation on a per tenant/network combination: VLANs (IEEE 802.1Q tagging) or L2 tunnels using GRE encapsulation. Which method you choose for traffic segregation and isolation is determined by the scope and scale of your OpenStack deployment.

VLANs

VLANs are realized as packets on a specific physical network containing IEEE 802.1Q headers with a specific VLAN ID (VID) field value. VLAN networks sharing the same physical network are isolated from each other at L2, and can even have overlapping IP address spaces. Each distinct physical network supporting VLAN networks is treated as a separate VLAN trunk, with a distinct space of VID values. Valid VID values are 1 through 4094.

VLAN configuration complexity depends on your OpenStack design requirements. In order to allow OpenStack Networking to efficiently use VLANs, you must allocate a VLAN range (one for each tenant) and turn each compute node physical switch port into a VLAN trunk port.

Note

NOTE: If you intend for your network to support more than 4094 tenants VLAN is probably not the correct option for you as multiple 'hacks' are required to extend the VLAN tags to more than 4094 tenants.

L2 Tunneling

Network tunneling encapsulates each tenant/network combination with a unique "tunnel-id" that is used to identify the network traffic belonging to that combination. The tenant's L2 network connectivity is independent of physical locality or underlying network design. By encapsulating traffic inside IP packets, that traffic can cross Layer-3 boundaries, removing the need for preconfigured VLANs and VLAN trunking. Tunneling adds a layer of obfuscation to network data traffic, reducing the visibility of individual tenant traffic from a monitoring point of view.

OpenStack Networking currently only supports GRE encapsulation with planned future support of VXLAN due in the Havana release.

The choice of technology to provide L2 isolation is dependent upon the scope and size of tenant networks that will be created in your deployment. If your environment has limited VLAN ID availability or will have a large number of L2 networks, it is our recommendation that you utilize tunneling.

Network Services

The choice of tenant network isolation affects how the network security and control boundary is implemented for tenant services. The following additional network services are either available or currently under development to enhance the security posture of the OpenStack network architecture.

Access Control Lists

OpenStack Compute supports tenant network traffic access controls directly when deployed with the legacy nova-network service, or may defer access control to the OpenStack Networking service.

Note, legacy nova-network security groups are applied to all virtual interface ports on an instance using IPTables.

Security groups allow administrators and tenants the ability to specify the type of traffic, and direction (ingress/egress) that is allowed to pass through a virtual interface port. Security groups rules are stateful L2-L4 traffic filters.

It is our recommendation that you enable security groups via OpenStack Networking.

L3 Routing and NAT

OpenStack Networking routers can connect multiple L2 networks, and can also provide a *gateway* that connects one or more private L2 networks to a shared *external* network, such as a public network for access to the Internet.

The L3 router provides basic Network Address Translation (NAT) capabilities on *gateway* ports that uplink the router to external networks. This router SNATs (Static NAT) all traffic by default, and supports floating IPs, which creates a static one-to-one mapping from a public IP on the external network to a private IP on one of the other subnets attached to the router.

It is our recommendation to leverage per tenant L3 routing and Floating IPs for more granular connectivity of tenant VMs.

Quality of Service (QoS)

The ability to set QoS on the virtual interface ports of tenant instances is a current deficiency for OpenStack Networking. The application of QoS for traffic shaping and rate-limiting at the physical network edge device is insufficient due to the dynamic nature of workloads in an OpenStack deployment and can not be leveraged in the traditional way. QoS-as-a-Service (QoSaaS) is currently in development for the OpenStack Networking Havana release as an experimental feature. QoSaaS is planning to provide the following services:

- Traffic shaping via DSCP markings
- Rate-limiting on a per port/network/tenant basis.
- Port mirroring (via open source or third-party plugins)
- Flow analysis (via open source or third-party plugins)

Tenant traffic port mirroring or Network Flow monitoring is currently not an exposed feature in OpenStack Networking. There are third-party plugin

extensions that do provide Port Mirroring on a per port/network/tenant basis. If Open vSwitch is used on the networking hypervisor, it is possible to enable sFlow and port mirroring, however it will require some operational effort to implement.

Load Balancing

An experimental feature in the Grizzly release of OpenStack Networking is Load-Balancer-as-a-service (LBaaS). The LBaaS API gives early adopters and vendors a chance to build implementations of the technology. The reference implementation however, is still experimental and should likely not be run in a production environment. The current reference implementation is based on HA-Proxy. There are third-party plugins in development for extensions in OpenStack Networking to provide extensive L4-L7 functionality for virtual interface ports.

Firewalls

FW-as-a-Service (FWaaS) is currently in development for the OpenStack Networking Havana release as an experimental feature. FWaaS will address the need to manage and leverage the rich set of security features provided by typical firewall products which are typically far more comprehensive than what is currently provided by security groups. There are third-party plugins in development for extensions in OpenStack Networking to support this.

It is critical during the design of an OpenStack Networking infrastructure to understand the current features and limitations of network services that are available. Understanding where the boundaries of your virtual and physical networks will help you add the required security controls in your environment.

Network Services Extensions

Here is a list of known plugins provided by the open source community or by SDN companies that work with OpenStack Networking:

Big Switch Controller Plugin, Brocade Neutron Plugin Brocade Neutron Plugin, Cisco UCS/Nexus Plugin, Cloudbase Hyper-V Plugin, Extreme Networks Plugin, Juniper Networks Neutron Plugin, Linux Bridge Plugin, Mellanox Neutron Plugin, MidoNet Plugin, NEC OpenFlow Plugin, Nicira Network Virtualization Platform (NVP) Plugin, Open vSwitch Plugin, PLUMgrid Plugin, Ruijie Networks Plugin, Ryu OpenFlow Controller Plugin

For a more detailed comparison of all features provided by plugins as of the Folsom release, see Sebastien Han's comparison.

Networking Services Limitations

OpenStack Networking has the following known limitations:

- **Overlapping IP addresses** — If nodes that run either `neutron-l3-agent` or `neutron-dhcp-agent` use overlapping IP addresses, those nodes must use Linux network namespaces. By default, the DHCP and L3 agents use Linux network namespaces. However, if the host does not support these namespaces, run the DHCP and L3 agents on different hosts.

  If network namespace support is not present, a further limitation of the L3 Agent is that only a single logical router is supported.

- **Multi-Host DHCP-agent** — OpenStack Networking supports multiple l3-agent and dhcp-agents with load balancing. However, tight coupling of the location of the virtual machine is not supported.

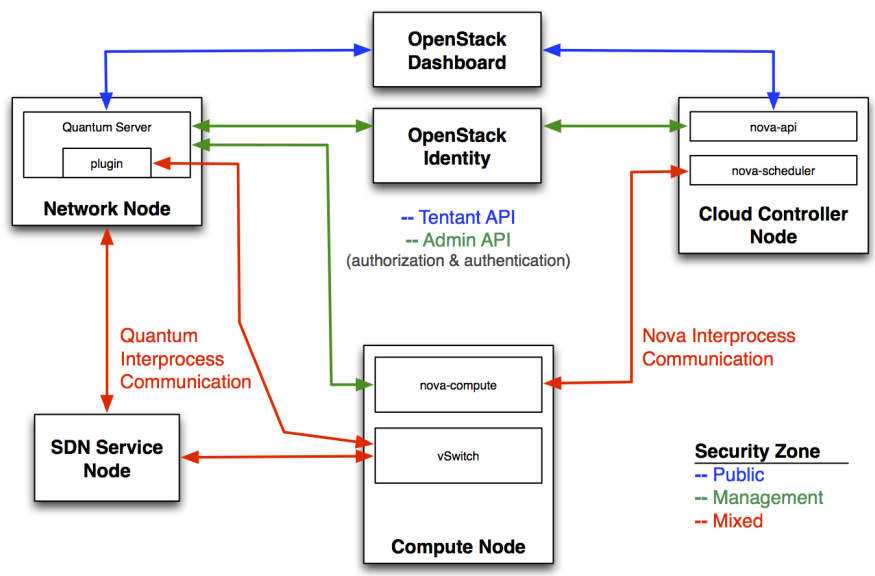
- **No IPv6 Support for L3 agents** — The neutron-l3-agent, used by many plugins to implement L3 forwarding, supports only IPv4 forwarding.

26. Securing OpenStack Networking Services

In order to secure OpenStack Networking, an understanding of the workflow process for tenant instance creation needs to be mapped to security domains.

There are four main services that interact with OpenStack Networking. In a typical OpenStack deployment these services map to the following security domains:

- OpenStack Dashboard: Public and Management
- OpenStack Identity: Management
- OpenStack Compute Node: Management and Guest
- OpenStack Network Node: Management, Guest, and possibly Public depending upon neutron-plugin in use.
- SDN Services Node: Management, Guest and possibly Public depending upon product used.



In order to isolate sensitive data communication between the OpenStack Networking services and other OpenStack core services, we strongly recommend that these communication channels be configured to only allow communications over an isolated management network.

OpenStack Networking Service Configuration

Restrict Bind Address of the API server: neutron-server

To restrict the interface or IP address on which the OpenStack Networking API service binds a network socket for incoming client connections, specify the bind_host and bind_port in the neutron.conf file as shown:

```
# Address to bind the API server
bind_host = <ip address of server>

# Port the bind the API server to
bind_port = 9696
```

Restrict DB and RPC communication of the OpenStack Networking services:

Various components of the OpenStack Networking services use either the messaging queue or database connections to communicate with other components in OpenStack Networking.

It is recommended that you follow the guidelines provided in the Database Authentication and Access Control chapter in the Database section for all components that require direct DB connections.

It is recommended that you follow the guidelines provided in the Queue Authentication and Access Control chapter in the Messaging section for all components that require RPC communication.

27. Networking Services Security Best Practices

This section discusses OpenStack Networking configuration best practices as they apply to tenant network security within your OpenStack deployment.

Tenant Network Services Workflow

OpenStack Networking provides users real self services of network resources and configurations. It is important that Cloud Architects and Operators evaluate the their design use cases in providing users the ability to create, update, and destroy available network resources.

Networking Resource Policy Engine

A policy engine and its configuration file, `policy.json`, within OpenStack Networking provides a method to provide finer grained authorization of users on tenant networking methods and objects. It is important that cloud architects and operators evaluate their design and use cases in providing users and tenants the ability to create, update, and destroy available network resources as it has a tangible effect on tenant network availability, network security, and overall OpenStack security. For a more detailed explanation of OpenStack Networking policy definition, please refer to the Authentication and authorization section in the *OpenStack Cloud Administrator Guide*.

It is important to review the default networking resource policy and modify the policy a

If your deployment of OpenStack provides multiple external access points into different security domains it is important that you limit the tenant's ability to attach multiple vNICs to multiple external access points – this would bridge these security domains and could lead to unforseen security compromise. It is possible mitigate this risk by utilizing the host aggregates functionality provided by OpenStack Compute or through splitting the

tenant VMs into multiple tenant projects with different virtual network configurations.

Security Groups

The OpenStack Networking Service provides security group functionality using a mechanism that is more flexible and powerful than the security group capabilities built into OpenStack Compute. Thus, when using OpenStack Networking, *nova.conf* should always disable built-in security groups and proxy all security group calls to the OpenStack Networking API. Failure to do so will result in conflicting security policies being simultaneously applied by both services. To proxy security groups to OpenStack Networking, use the following configuration values:

- firewall_driver : must be set to 'nova.virt.firewall.NoopFirewallDriver' so that `nova-compute` does not perform iptables-based filtering itself.
- security_group_api : must be set to 'neutron' so that all security group requests are proxied to the OpenStack Network Service.

Security groups and security group rules allow administrators and tenants the ability to specify the type of traffic and direction (ingress/egress) that is allowed to pass through a virtual interface port. A security group is a container for security group rules. When a virtual interface port is created in OpenStack Networking it is associated with a security group. If a security group is not specified, the port will be associated with a 'default' security group. By default this group will drop all ingress traffic and allow all egress. Rules can be added to this group in order to change the behaviour.

When using the security group API through OpenStack Compute, security groups are applied to all virtual interface ports on an instance. The reason for this is that OpenStack Compute security group APIs are instance based and not virtual interface port based as OpenStack Networking.

Quotas

Quotas provide the ability to limit the number of network resources available to tenants. You can enforce default quotas for all tenants.

```
/etc/neutron/neutron.conf
[QUOTAS]
# resource name(s) that are supported in quota features
quota_items = network,subnet,port
```

```
# default number of resource allowed per tenant, minus for
 unlimited
#default_quota = -1

# number of networks allowed per tenant, and minus means
 unlimited
quota_network = 10

# number of subnets allowed per tenant, and minus means
 unlimited
quota_subnet = 10

# number of ports allowed per tenant, and minus means unlimited
quota_port = 50

# number of security groups allowed per tenant, and minus means
 unlimited
quota_security_group = 10

# number of security group rules allowed per tenant, and minus
 means unlimited
quota_security_group_rule = 100

# default driver to use for quota checks
quota_driver = neutron.quota.ConfDriver
```

OpenStack Networking also supports per-tenant quotas limit via a quota extension API. To enable per-tenant quotas, you need to set `quota_driver` in `neutron.conf`.

```
quota_driver = neutron.db.quota_db.DbQuotaDriver
```

28. Case Studies

In this case study we discuss how Alice and Bob would address providing networking services to the user.

Alice's Private Cloud

A key objective of Alice's cloud is to integrate with the existing auth services and security resources. The key design parameters for this private cloud are a limited scope of tenants, networks and workload type. This environment can be designed to limit what available network resources are available to the tenant and what are the various default quotas and security policies are available. The network policy engine can be modified to restrict creation and changes to network resources. In this environment, Alice might want to leverage nova-network in the application of security group polices on a per instance basis vs. neutron's application of security group polices on a per port basis. L2 isolation in this environment would leverage VLAN tagging. The use of VLAN tags will allow great visibility of tenant traffic by leveraging existing features and tools of the physical infrastructure.

Bob's Public Cloud

A major business driver for Bob is to provide an advanced networking services to his customers. Bob's customers would like to deploy multi-tiered application stacks. This multi-tiered application are either existing enterprise application or newly deployed applications. Since Bob's Public cloud is a multi-tenancy enterprise service, the choice to use for L2 isolation in this environment is to use Overlay networking. Another aspect of Bob's cloud is the self-service aspect where the customer can provision available networking services as needed. These networking services encompass L2 networks, L3 Routing, Network *ACL* and NAT. It is important that per-tenant quota's be implemented in this environment.

An added benefit with utilizing OpenStack Networking is when new advanced networking services become available, these new features can be easily provided to the end customers.

29. Message Queuing Architecture

Inter-process communication within OpenStack is facilitated via message queueing services. Today, three messaging service backends are supported:

- RabbitMQ
- Qpid
- ZeroMQ

Both RabbitMQ and Qpid are Advanced Message Queuing Protocol (AMQP) frameworks which provide message queues for peer-to-peer communication. Queue implementations are typically deployed as centralized or decentralized pool of queue servers. ZeroMQ differs by communicating directly using TCP sockets between peers.

Message queues effectively facilitate command and control functions across OpenStack deployments. Once access to the queue is permitted no further authorization checks are performed. Services accessible via the queue do validate the contexts and tokens within the actual message payload. However, awareness of the token's expiration value should be noted as these tokens are potentially replayable and may provide authorization for other services within the infrastructure.

OpenStack does not support message-level confidence (i.e., message signing). Because of this, the message transport itself must be secured and authentication to the queue server must be performed. For HA configurations, queue to queue authentication and encryption should to be performed as well.

With ZeroMQ messaging, IPC sockets are used on individual machines. These sockets may be vulnerable to attack for local message injection and snooping unless secured by an operator.

30. Messaging Security

This chapter discusses security hardening approaches for the three most common message queuing solutions use in OpenStack: RabbitMQ, Qpid, and ZeroMQ.

Messaging Transport Security

AMQP based solutions (Qpid and RabbitMQ) support transport-level security using SSL. ZeroMQ messaging does not natively support SSL, but transport-level security is possible using labelled IPSec or CIPSO network labels.

We highly recommend enabling transport-level cryptography for your message queue. Using SSL for the messaging client connections provides protection of the communications from tampering and eavesdropping in-transit to the messaging server. Below is guidance on how SSL is typically configured for the two popular messaging servers Qpid and RabbitMQ. When configuring the trusted certificate authority (CA) bundle that your messaging server uses to verify client connections, it is recommended that this be limited to only the CA used for your nodes, preferably an internally managed CA. The bundle of trusted CAs will determine which client certificates will be authorized and pass the client-server verification step of the setting up the SSL connection. Note, when installing the certificate and key files, ensure that the file permissions are restricted, for example chmod 0600, and the ownership is restricted to the messaging server daemon user to prevent unauthorized access by other processes and users on the messaging server.

RabbitMQ Server SSL Configuration

The following lines should be added to the system-wide RabbitMQ configuration file, typically /etc/rabbitmq/rabbitmq.config:

```
[
  {rabbit, [
     {tcp_listeners, [] },
     {ssl_listeners, [{"<ip address or hostname of management
 network interface", 5671}] },
```

```
     {ssl_options, [{cacertfile,"/etc/ssl/cacert.pem"},
                    {certfile,"/etc/ssl/rabbit-server-cert.
pem"},
                    {keyfile,"/etc/ssl/rabbit-server-key.pem"},
                    {verify,verify_peer},
                    {fail_if_no_peer_cert,true}]}
   ]}
].
```

Note, the 'tcp_listeners' option is set to '[]' to prevent it from listening an on non-SSL port. 'ssl_listeners' option should be restricted to only listen on the management network for the services.

For more information on RabbitMQ SSL configuration see:

- RabbitMQ Configuration
- RabbitMQ SSL

Qpid Server SSL Configuration

The Apache Foundation has a messaging security guide for Qpid. See:

- Apache Qpid SSL

Queue Authentication and Access Control

RabbitMQ and Qpid offer authentication and access control mechanisms for controlling access to queues. ZeroMQ offers no such mechanisms.

Simple Authentication and Security Layer (SASL) is a framework for authentication and data security in Internet protocols. Both RabbitMQ and Qpid offer SASL and other pluggable authentication mechanisms beyond simple usernames and passwords that allow for increased authentication security. While RabbitMQ supports SASL, support in OpenStack does not currently allow for requesting a specific SASL authentication mechanism. RabbitMQ support in OpenStack allows for either username and password authentication over an unencrypted connection or username and password in conjunction with X.509 client certificates to establish the secure SSL connection.

We recommend configuring X.509 client certificates on all the OpenStack service nodes for client connections to the messaging queue and where possible (currently only Qpid) perform authentication with X.509 client certificates. When using usernames and passwords, accounts should be

created per-service and node for finer grained auditability of access to the queue.

The SSL libraries in use by these queuing servers should also be considered prior to deployment. Qpid uses Mozilla's NSS library, whereas RabbitMQ uses Erlang's SSL module which uses OpenSSL.

Authentication Configuration Example - RabbitMQ

On the RabbitMQ server, delete the default 'guest' user:

```
rabbitmqctl delete_user quest
```

On the RabbitMQ server, for each OpenStack service or node that communicates with the message queue set up user accounts and privileges:

```
rabbitmqctl add_user compute01 password
rabbitmqctl set_permissions compute01 ".*"".*"".*"
```

For additional configuration information see:

- RabbitMQ Access Control
- RabbitMQ Authentication
- RabbitMQ Plugins
- RabbitMQ SASL External Auth

OpenStack Service Configuration - RabbitMQ

```
[DEFAULT]
rpc_backend=nova.openstack.common.rpc.impl_kombu
rabbit_use_ssl=True
rabbit_host=
rabbit_port=5671
rabbit_user=compute01
rabbit_password=password
kombu_ssl_keyfile=/etc/ssl/node-key.pem
kombu_ssl_certfile=/etc/ssl/node-cert.pem
kombu_ssl_ca_certs=/etc/ssl/cacert.pem
```

NOTE: A bug exists in the current version of OpenStack Grizzly where if 'kombu_ssl_version' is currently specified in the configuration file for any

of the OpenStack services it will cause the following python traceback error: 'TypeError: an integer is required'. The current workaround is to remove 'kombu_ssl_version' from the configuration file. Refer to bug report 1195431 for current status.

Authentication Configuration Example - Qpid

For configuration information see:

- Apache Qpid Authentication
- Apache Qpid Authorization

OpenStack Service Configuration - Qpid

```
[DEFAULT]
rpc_backend=nova.openstack.common.rpc.impl_qpid
qpid_protocol=ssl
qpid_hostname=<ip or hostname of management network interface of
 messaging server>
qpid_port=5671qpid_username=compute01

qpid_password=password
```

Optionally, if using SASL with Qpid specify the SASL mechanisms in use by adding:

```
qpid_sasl_mechanisms=<space separated list of SASL mechanisms to
 use for auth>
```

Message Queue Process Isolation & Policy

Each project provides a number of services which send and consume messages. Each binary which sends a message is expected to consume messages, if only replies, from the queue.

Message queue service processes should be isolated from each other and other processes on a machine.

Namespaces

Network namespaces are highly recommended for all services running on OpenStack Compute Hypervisors. This will help prevent against the

bridging of network traffic between VM guests and the management network.

When using ZeroMQ messaging, each host must run at least one ZeroMQ message receiver to receive messages from the network and forward messages to local processes via IPC. It is possible and advisable to run an independent message receiver per project within an IPC namespace, along with other services within the same project.

Network Policy

Queue servers should only accept connections from the management network. This applies to all implementations. This should be implemented through configuration of services and optionally enforced through global network policy.

When using ZeroMQ messaging, each project should run a separate ZeroMQ receiver process on a port dedicated to services belonging to that project. This is equivalent to the AMQP concept of control exchanges.

Mandatory Access Controls

The configuration for these processes should be restricted to those processes, not only by Directory Access Controls, but through Mandatory Access Controls. The goal of such restrictions is to prevent isolation from other processes running on the same machine(s).

31. Case Studies

The message queue is a critical piece of infrastructure that supports a number of OpenStack services but is most strongly associated with the Compute service. Due to the nature of the message queue service, Alice and Bob have similar security concerns. One of the larger concerns that remains is that many systems have access to this queue and there is no way for a consumer of the queue messages to verify which host or service placed the messages on the queue. An attacker who is able to successfully place messages on the queue is able to create and delete VM instances, attach the block storage of any tenant and a myriad of other malicious actions. There are a number of solutions on the horizon to fix this, with several proposals for message signing and encryption making their way through the OpenStack development process.

Alice's Private Cloud

In this case Alice's controls mimic those Bob has deployed for the public cloud.

Bob's Public Cloud

Bob assumes that at some point infrastructure or networks underpinning the Compute service may become compromised. Due to this, he recognizes the importance of locking down access to the message queue. To do this Bob deploys his RabbitMQ servers with SSL and X.509 client auth for access control. This in turn limits the capabilities of an attacker who has compromised a system that does not have queue access.

Additionally, Bob adds strong network ACL rulesets to enforce which endpoints can communicate with the message servers. This second control provides some additional assurance should the other protections fail.

32. Database Backend Considerations

The choice of database server is an important consideration in the security of an OpenStack deployment. While security considerations are not the only basis on which a database server must be chosen, security considerations are the only ones within the scope of this book. In practice, OpenStack only supports two database types: PostgreSQL and MySQL.

PostgreSQL has a number of desirable security features such as Kerberos authentication, object-level security, and encryption support. The PostgreSQL community has done well to provide solid guidance, documentation, and tooling to promote positive security practices.

MySQL has a large community, wide-spread adoption, and provides high availability options. MySQL also has the ability to provide enhanced client authentication by way of plug-in authentication mechanisms. Forked distributions in the MySQL community provide many options for consideration. It is important to choose a specific implementation of MySQL based on a thorough evaluation of the security posture and the level of support provided for the given distribution.

Security References for Database Backends

Those deploying MySQL or PostgreSQL are advised to refer to existing security guidance. Some references are listed below:

MySQL:

- OWASP MySQL Hardening
- MySQL Pluggable Authentication
- Security in MySQL

PostgreSQL:

- OWASP PostgreSQL Hardening

- Total security in a PostgreSQL database

33. Database Access Control

Each of the core OpenStack services (Compute, Identity, Networking, Block Storage) store state and configuration information in databases. In this chapter, we discuss how databases are used currently in OpenStack. We also explore security concerns, and the security ramifications of database backend choices.

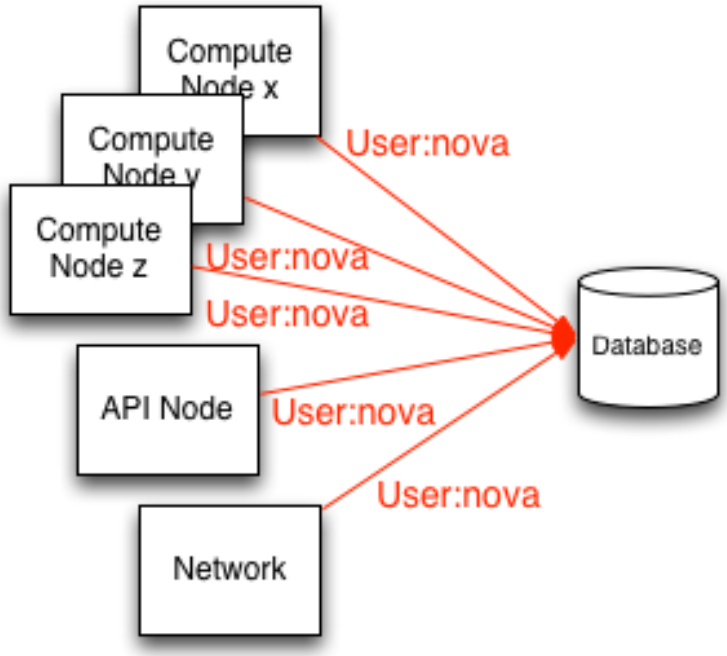
OpenStack Database Access Model

All of the services within an OpenStack project access a single database. There are presently no reference policies for creating table or row based access restrictions to the database.

There are no general provisions for granular control of database operations in OpenStack. Access and privileges are granted simply based on whether a node has access to the database or not. In this scenario, nodes with access to the database may have full privileges to DROP, INSERT, or UPDATE functions.

Granular Access Control

By default, each of the OpenStack services and their processes access the database using a shared set of credentials. This makes auditing database operations and revoking access privileges from a service and its processes to the database particularly difficult.

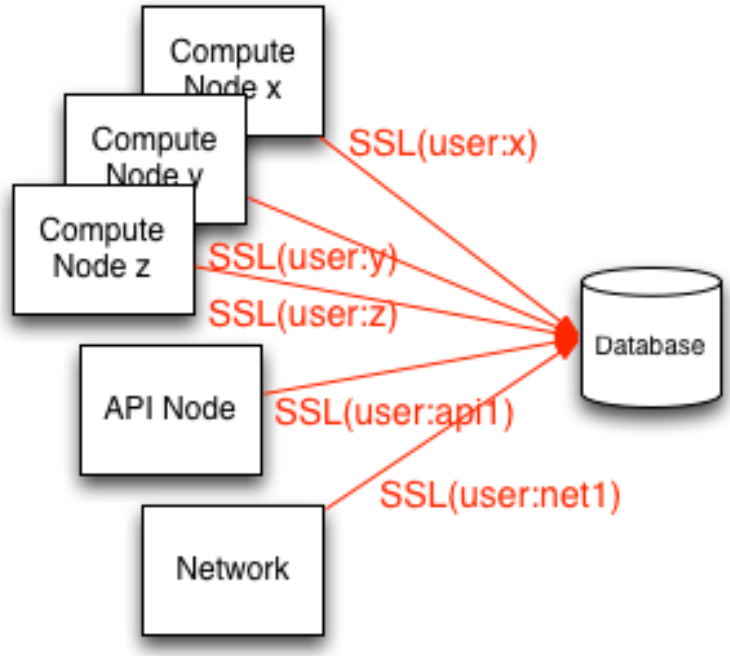


Nova Conductor

The compute nodes are the least trusted of the services in OpenStack because they host tenant instances. The `nova-conductor` service has been introduced to serve as a database proxy, acting as an intermediary between the compute nodes and the database. We discuss its ramifications later in this chapter.

We strongly recommend:

- All database communications be isolated to a management network
- Securing communications using SSL
- Creating unique database user accounts per OpenStack service endpoint (illustrated below)



Database Authentication and Access Control

Given the risks around access to the database, we strongly recommend that unique database user accounts be created per node needing access to the database. Doing this facilitates better analysis and auditing for ensuring compliance or in the event of a compromise of a node allows you to isolate the compromised host by removing access for that node to the database upon detection. When creating these per service endpoint database user accounts, care should be taken to ensure that they are configured to require SSL. Alternatively, for increased security it is recommended that the database accounts be configured using X.509 certificate authentication in addition to usernames and passwords.

Privileges

A separate database administrator (DBA) account should be created and protected that has full privileges to create/drop databases, create user accounts, and update user privileges. This simple means of separation of

responsibility helps prevent accidental misconfiguration, lowers risk and lowers scope of compromise.

The database user accounts created for the OpenStack services and for each node should have privileges limited to just the database relevant to the service where the node is a member.

Require User Accounts to Require SSL Transport

Configuration Example #1: (MySQL)

```
GRANT ALL ON dbname.* to 'compute01'@'hostname' IDENTIFIED BY
 'password' REQUIRE SSL;
```

Configuration Example #2: (PostgreSQL)

In file pg_hba.conf:

```
hostssl dbname compute01 hostname md5
```

Note this command only adds the ability to communicate over SSL and is non-exclusive. Other access methods that may allow unencrypted transport should be disabled so that SSL is the sole access method.

The 'md5' parameter defines the authentication method as a hashed password. We provide a secure authentication example in the section below.

Authentication with X.509 Certificates

Security may be enhanced by requiring X.509 client certificates for authentication. Authenticating to the database in this manner provides greater identity assurance of the client making the connection to the database and ensures that the communications are encrypted.

Configuration Example #1: (MySQL)

```
GRANT ALL on dbname.* to 'compute01'@'hostname' IDENTIFIED BY
 'password' REQUIRE SUBJECT
```

```
'/C=XX/ST=YYY/L=ZZZZ/O=cloudycloud/CN=compute01' AND ISSUER
'/C=XX/ST=YYY/L=ZZZZ/O=cloudycloud/CN=cloud-ca';
```

Configuration Example #2: (PostgreSQL)

```
hostssl dbname compute01 hostname cert
```

OpenStack Service Database Configuration

If your database server is configured to require X.509 certificates for authentication you will need to specify the appropriate SQLAlchemy query parameters for the database backend. These parameters specify the certificate, private key, and certificate authority information for use with the initial connection string.

Example of an `:sql_connection` string for X.509 certificate authentication to MySQL:

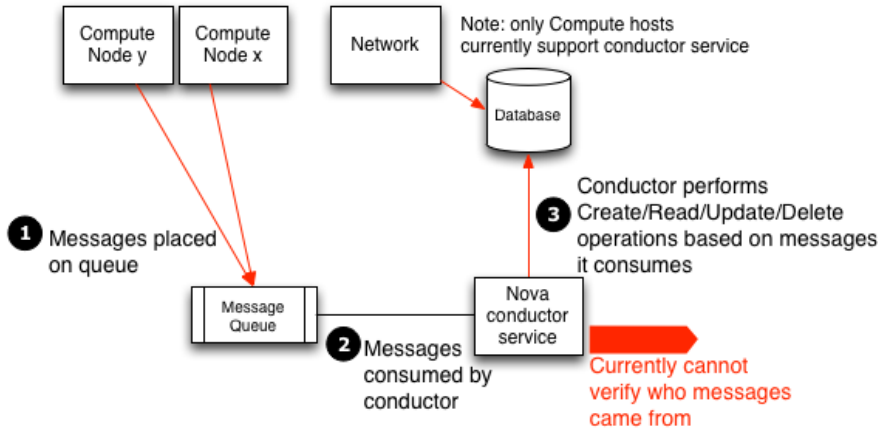
```
sql_connection = mysql://compute01:password@localhost/nova?
charset=utf8&ssl_ca=/etc/mysql/cacert.pem&ssl_cert=/etc/mysql/
server-cert.pem&ssl_key=/etc/mysql/server-key.pem
```

Nova Conductor

OpenStack Compute offers a sub-service called `nova-conductor` which proxies database connections, with the primary purpose of having the nova compute nodes interfacing with `nova-conductor` to meet data persistence needs as opposed to directly communicating with the database.

Nova-conductor receives requests over RPC and performs actions on behalf of the calling service without granting granular access to the database, its tables, or data within. Nova-conductor essentially abstracts direct database access away from Nova compute nodes.

This abstraction offers the advantage of restricting services to executing methods with parameters, similar to stored procedures, preventing a large number of systems from directly accessing or modifying database data. This is accomplished without having these procedures stored or executed within the context or scope of the database itself, a frequent criticism of typical stored procedures.



Unfortunately, this solution complicates the task of more fine-grained access control and the ability to audit data access. Because the `nova-conductor` service receives requests over RPC, it highlights the importance of improving the security of messaging. Any node with access to the message queue may execute these methods provided by the `nova-conductor` and effectively modifying the database.

Finally, it should be noted that as of the Grizzly release, gaps exist where `nova-conductor` is not used throughout OpenStack Compute. Depending on one's configuration, the use of `nova-conductor` may not allow deployers to avoid the necessity of providing database GRANTs to individual compute host systems.

Note, as `nova-conductor` only applies to OpenStack Compute, direct database access from compute hosts may still be necessary for the operation of other OpenStack components such as Metering (Ceilometer), Networking, and Block Storage.

Implementors should weigh the benefits and risks of both configurations before enabling or disabling the `nova-conductor` service. We are not yet prepared to recommend the use of `nova-conductor` in the Grizzly release. However, we do believe that this recommendation will change as additional features are added into OpenStack.

To disable the `nova-conductor`, place the following into your nova.conf file:

```
[conductor]
use_local = true
```

34. Database Transport Security

This chapter covers issues related to network communications to and from the database server. This includes IP address bindings and encrypting network traffic with SSL.

Database Server IP Address Binding

To isolate sensitive database communications between the services and the database, we strongly recommend that the database server(s) be configured to only allow communications to and from the database over an isolated management network. This is achieved by restricting the interface or IP address on which the database server binds a network socket for incoming client connections.

Restricting Bind Address for MySQL

In my.cnf:

```
[mysqld]
...
bind-address <ip address or hostname of management network
 interface>
```

Restricting Listen Address for PostgreSQL

In postgresql.conf:

```
listen_addresses = <ip address or hostname of management network
 interface>
```

Database Transport

In addition to restricting database communications to the management network, we also strongly recommend that the cloud administrator

configure their database backend to require SSL. Using SSL for the database client connections protects the communications from tampering and eavesdropping. As will be discussed in the next section, using SSL also provides the framework for doing database user authentication via X.509 certificates (commonly referred to as PKI). Below is guidance on how SSL is typically configured for the two popular database backends MySQL and PostgreSQL.

Note

NOTE: When installing the certificate and key files, ensure that the file permissions are restricted, for example `chmod 0600`, and the ownership is restricted to the database daemon user to prevent unauthorized access by other processes and users on the database server.

MySQL SSL Configuration

The following lines should be added in the system-wide MySQL configuration file:

In my.cnf:

```
[[mysqld]]
...
ssl-ca=/path/to/ssl/cacert.pem
ssl-cert=/path/to/ssl/server-cert.pem
ssl-key=/path/to/ssl/server-key.pem
```

Optionally, if you wish to restrict the set of SSL ciphers used for the encrypted connection. See http://www.openssl.org/docs/apps/ciphers.html for a list of ciphers and the syntax for specifying the cipher string:

```
ssl-cipher='cipher:list'
```

PostgreSQL SSL Configuration

The following lines should be added in the system-wide PostgreSQL configuration file, `postgresql.conf`.

```
ssl = true
```

Optionally, if you wish to restrict the set of SSL ciphers used for the encrypted connection. See http://www.openssl.org/docs/apps/ciphers.html for a list of ciphers and the syntax for specifying the cipher string:

```
ssl-ciphers = 'cipher:list'
```

The server certificate, key, and certificate authority (CA) files should be placed in the $PGDATA directory in the following files:

- $PGDATA/server.crt - Server certificate
- $PGDATA/server.key - Private key corresponding to server.crt
- $PGDATA/root.crt - Trusted certificate authorities
- $PGDATA/root.crl - Certificate revocation list

35. Case Studies

In this case study we discuss how Alice and Bob would address database selection and configuration for their respective private and public clouds.

Alice's Private Cloud

Alice's organization has high availability concerns, so she has elected to use MySQL for the database. She further places the database on the Management network and uses SSL with mutual authentication among the services to ensure secure access. Given there will be no external access of the database, she uses certificates signed with the organization's self-signed root certificate on the database and its access endpoints. Alice creates separate user accounts for each database user, and configures the database to use both passwords and X.509 certificates for authentication. She elects not to use the `nova-conductor` sub-service due to the desire for fine-grained access control policies and audit support.

Bob's Public Cloud

Bob is concerned about strong separation of his tenants' data, so he has elected to use the Postgres database , known for its stronger security features. The database resides on the Management network and uses SSL with mutual authentication with the services. Since the database is on the Management network, the database uses certificates signed with the company's self-signed root certificate. Bob creates separate user accounts for each database user, and configures the database to use both passwords and X.509 certificates for authentication. He elects not to use the `nova-conductor` sub-service due to a desire for fine-grained access control.

36. Data Privacy Concerns

OpenStack is designed to support multitenancy and those tenants will most probably have different data requirements. As a cloud builder and operator you need to ensure your OpenStack environment can address various data privacy concerns and regulations. In this chapter we will address the following topics around Data Privacy as it pertains to OpenStack implementations:

- Data Residency
- Data Disposal

Data Residency

The privacy and isolation of data has consistently been cited as the primary barrier to cloud adoption over the past few years. Concerns over who owns data in the cloud and whether the cloud operator can be ultimately trusted as a custodian of this data have been significant issues in the past.

Numerous OpenStack services maintain data and metadata belonging to tenants or reference tenant information.

Tenant data stored in an OpenStack cloud may include the following items:

- Swift objects
- Compute instance ephemeral filesystem storage
- Compute instance memory
- Cinder volume data
- Public keys for Compute Access
- Virtual Machine Images in Glance
- Machine snapshots
- Data passed to OpenStack Compute's configuration-drive extension

Metadata stored by an OpenStack cloud includes the following non-exhaustive items:

- Organization name
- User's "Real Name"
- Number or size of running instances, buckets, objects, volumes, and other quota-related items
- Number of hours running instances or storing data
- IP addresses of users
- Internally generated private keys for compute image bundling

Data Disposal

OpenStack operators should strive to provide a certain level of tenant data disposal assurance. Best practices suggest that the operator sanitize cloud system media (digital and non-digital) prior to disposal, release out of organization control or release for reuse. Sanitization methods should implement an appropriate level of strength and integrity given the specific security domain and sensitivity of the information.

> "Sanitization is the process used to remove information from system media such that there is reasonable assurance that the information cannot be retrieved or reconstructed. Sanitization techniques, including clearing, purging, and destroying media information, prevent the disclosure of organizational information to unauthorized individuals when such media is reused or released for disposal." [NIST Special Publication 800-53 Revision 3]

General data disposal and sanitization guidelines as adopted from NIST recommended security controls. Cloud Operators should:

1. Track, document and verify media sanitization and disposal actions.
2. Test sanitation equipment and procedures to verify proper performance.
3. Sanitize portable, removable storage devices prior to connecting such devices to the cloud infrastructure.
4. Destroy cloud system media that cannot be sanitized.

In an OpenStack deployment you will need to address the following:

- Secure data erasure
- Instance memory scrubbing
- Cinder volume data
- Compute instance ephemeral storage
- Bare metal server sanitization

Data not securely erased

Within OpenStack some data may be deleted, but not securely erased in the context of the NIST standards outlined above. This is generally applicable to most or all of the above-defined metadata and information stored in the database. This may be remediated with database and/or system configuration for auto vacuuming and periodic free-space wiping.

Instance memory scrubbing

Specific to various hypervisors is the treatment of instance memory. This behavior is not defined in OpenStack Compute, although it is generally expected of hypervisors that they will make a best effort to scrub memory either upon deletion of an instance, upon creation of an instance, or both.

Xen explicitly assigns dedicated memory regions to instances and scrubs data upon the destruction of instances (or domains in Xen parlance). KVM depends more greatly on Linux page management; A complex set of rules related to KVM paging is defined in the KVM documentation.

It is important to note that use of the Xen memory balloon feature is likely to result in information disclosure. We strongly recommended to avoid use of this feature.

For these and other hypervisors, we recommend referring to hypervisor-specific documentation.

Cinder volume data

Plugins to OpenStack Block Storage will store data in a variety of ways. Many plugins are specific to a vendor or technology, whereas others are more DIY solutions around filesystems such as LVM or ZFS. Methods to

securely destroy data will vary from one plugin to another, from one vendor's solution to another, and from one filesystem to another.

Some backends such as ZFS will support copy-on-write to prevent data exposure. In these cases, reads from unwritten blocks will always return zero. Other backends such as LVM may not natively support this, thus the Cinder plugin takes the responsibility to override previously written blocks before handing them to users. It is important to review what assurances your chosen volume backend provides and to see what mediations may be available for those assurances not provided.

Finally, while not a feature of OpenStack, vendors and implementors may choose to add or support encryption of volumes. In this case, destruction of data is as simple as throwing away the key.

Compute instance ephemeral storage

The creation and destruction of ephemeral storage will be somewhat dependent on the chosen hypervisor and the OpenStack Compute plugin.

The libvirt plugin for compute may maintain ephemeral storage directly on a filesystem, or in LVM. Filesystem storage generally will not overwrite data when it is removed, although there is a guarantee that dirty extents are not provisioned to users.

When using LVM backed ephemeral storage, which is block-based, it is necessary that the OpenStack Compute software securely erases blocks to prevent information disclosure. There have in the past been information disclosure vulnerabilities related to improperly erased ephemeral block storage devices.

Filesystem storage is a more secure solution for ephemeral block storage devices than LVM as dirty extents cannot be provisioned to users. However, it is important to be mindful that user data is not destroyed, so it is suggested to encrypt the backing filesystem.

Bare metal server sanitization

A bare metal server driver for Nova was under development and has since moved into a separate project called Ironic. At the time of this writing, Ironic does not appear to address sanitization of tenant data resident the physical hardware.

Additionally, it is possible for tenants of a bare metal system to modify system firmware. TPM technology, described in ##link:Management/Node

Bootstrapping##, provides a solution for detecting unauthorized firmware changes.

37. Data Encryption

The option exists for implementors to encrypt tenant data wherever it is stored on disk or transported over a network. This is above and beyond the general recommendation that users encrypt their own data before sending it to their provider.

The importance of encrypting data on behalf of tenants is largely related to the risk assumed by a provider that an attacker could access tenant data. There may be requirements here in government, as well as requirements per-policy, in private contract, or even in case law in regard to private contracts for public cloud providers. It is recommended that a risk assessment and legal consul advised before choosing tenant encryption policies.

Per-instance or per-object encryption is preferable over, in descending order, over per-project, per-tenant, per-host, and per-cloud aggregations. This recommendation is inverse to the complexity and difficulty of implementation. Presently, in some projects it is difficult or impossible to implement encryption as loosely granular as even per-tenant. We recommend implementors make a best-effort in encrypting tenant data.

Often, data encryption relates positively to the ability to reliably destroy tenant and per-instance data, simply by throwing away the keys. It should be noted that in doing so, it becomes of great importance to destroy those keys in a reliable and secure manner.

Opportunities to encrypt data for users are present:

- Swift objects
- Cinder volumes & Instance Ephemeral Filesystems
- Network data

Swift Objects

The ability to encrypt objects in Swift is presently limited to disk-level encryption per node. However, there does exist third-party extensions and

modules for per-object encryption. These modules have been proposed upstream, but have not per this writing been formally accepted. Below are some pointers:

https://github.com/Mirantis/swift-encrypt

http://www.mirantis.com/blog/on-disk-encryption-prototype-for-openstack-swift/

Cinder Volumes & Instance Ephemeral Filesystems

The ability to encrypt volumes depends on the service backends chosen. Some backends may not support this at all.

As both Cinder block storage and Nova compute support LVM backed storage, we can easily provide an example applicable to both systems. In deployments using LVM, encryption may be performed against the backing physical volumes. An encrypted block device would be created using the standard Linux tools, with the LVM physical volume (PV) created on top of the decrypted block device using pvcreate. Then, the vgcreate or vgmodify tool may be used to add the encrypted physical volume to an LVM volume group (VG).

A feature aimed for the Havana release provides encryption of the VM's data before it is written to disk. This allows the privacy of data to be maintained while residing on the storage device. The idea is similar to how self-encrypting drives work. This feature presents a normal block storage device to the VM but encrypts the bytes in the virtualization host before writing them to the disk. The block server operates exactly as it does when reading and writing unencrypted blocks, except special handling will be required for Cinder features such as snapshots and live migration. Note that this feature uses an independent key manager.

Network Data

Tenant data for compute could be encrypted over IPSec or other tunnels. This is not functionality common or standard in OpenStack, but is an option available to motivated and interested implementors.

Cinder block storage supports a variety of mechanisms for supplying mountable volumes. It is outside the scope of this guide to specify

recommendations for each Cinder backend driver. For the purpose of performance, many storage protocols are unencrypted. Some protocols such as iSCSI can provide authentication and encrypted sessions, it is our recommendation to enable these features.

38. Key Management

To address the often mentioned concern of tenant data privacy and limiting cloud provider liability, there is greater interest within the OpenStack community to make data encryption more ubiquitous. It is relatively easy for an end-user to encrypt their data prior to saving it to the cloud, and a this is a viable path for tenant objects such as media files, database archives among others. However, when client side encryption is used for virtual machine images, block storage etc, client intervention is necessary in the form of presenting keys to unlock the data for further use. To seamlessly secure the data and yet have it accessible without burdening the client with having to manage their keys and interactively provide them calls for a key management service within OpenStack. Providing encryption and key management services as part of OpenStack eases data-at-rest security adoption, addresses customer concerns about the privacy and misuse of their data with the added advantage of limiting cloud provider liability. Provider liability is of concern in multi-tenant public clouds with respect to handing over tenant data during a misuse investigation.

A key management service is in the early stages of being developed and has a way to go before becoming an official component of OpenStack. Refer to https://github.com/cloudkeep/barbican/wiki/_pages for details.

It shall support the creation of keys, and their secure saving (with a service master-key). Some of the design questions still being debated are how much of the Key Management Interchange Protocol (KMIP) to support, key formats, and certificate management. The key manager will be pluggable to facilitate deployments that need a third-party Hardware Security Module (HSM).

OpenStack Block Storage, Cinder, is the first service looking to integrate with the key manager to provide volume encryption.

References:

- Barbican
- KMIP

39. Case Studies

Returning to Alice and Bob, we will use this section to dive into their particular tenant data privacy requirements. Specifically, we will look into how Alice and Bob both handle tenant data, data destruction, and data encryption.

Alice's Private Cloud

As stated during the introduction to Alice's case study, data protection is of an extremely high priority. She needs to ensure that a compromise of one tenant's data does not cause loss of other tenant data. She also has strong regulator requirements that require documentation of data destruction activities. Alice does this using the following:

- Establishing procedures to sanitize tenant data when a program or project ends
- Track the destruction of both the tenant data and metadata via ticketing in a CMDB
- For Volume storage:
- Physical Server Issues
- To provide secure ephemeral instance storage, Alice implements qcow2 files on an encrypted filesystem.

Bob's Public Cloud

As stated during the introduction to Bob's case study, tenant privacy is of an extremely high priority. In addition to the requirements and actions Bob will take to isolate tenants from one another at the infrastructure layer, Bob also needs to provide assurances for tenant data privacy. Bob does this using the following:

- Establishing procedures to sanitize customer data when a customer churns
- Track the destruction of both the customer data and metadata via ticketing in a CMDB

- For Volume storage:
- Physical Server Issues
- To provide secure ephemeral instance storage, Bob implements qcow2 files on an encrypted filesystems.

40. Hypervisor Selection

Virtualization provides flexibility and other key benefits that enable cloud building. However, a virtualization stack also needs to be secured appropriately to reduce the risks associated with hypervisor breakout attacks. That is, while a virtualization stack can provide isolation between instances, or guest virtual machines, there are situations where that isolation can be less than perfect. Making intelligent selections for virtualization stack as well as following the best practices outlined in this chapter can be included in a layered approach to cloud security. Finally, securing your virtualization stack is critical in order to deliver on the promise of multitennancy, either between customers in a public cloud, between business units in a private cloud, or some mixture of the two in a hybrid cloud.

In this chapter, we discuss the hypervisor selection process. In the chapters that follow, we provide the foundational information needed for securing a virtualization stack.

Hypervisors in OpenStack

Whether OpenStack is deployed within private data centers or as a public cloud service, the underlying virtualization technology provides enterprise-level capabilities in the realms of scalability, resource efficiency, and uptime. While such high-level benefits are generally available across many OpenStack-supported hypervisor technologies, there are significant differences in each hypervisor's security architecture and features, particularly when considering the security threat vectors which are unique to elastic OpenStack environments. As applications consolidate into single Infrastructure as a Service (IaaS) platforms, instance isolation at the hypervisor level becomes paramount. The requirement for secure isolation holds true across commercial, government, and military communities.

Within the framework of OpenStack you can choose from any number of hypervisor platforms and corresponding OpenStack plugins to optimize your cloud environment. In the context of the OpenStack Security guide, we will be highlighting hypervisor selection considerations as they pertains to feature sets that are critical to security. However, these considerations are not meant to be an exhaustive investigation into the pros and cons

of particular hypervisors. NIST provides additional guidance in Special Publication 800-125, "*Guide to Security for Full Virtualization Technologies*".

Selection Criteria

As part of your hypervisor selection process, you will need to consider a number of important factors to help increase your security posture. Specifically, we will be looking into the following areas:

- Team Expertise
- Product or Project maturity
- Certifications, Attestations
- Additional Security Features
- Hypervisor vs. Baremetal
- Hardware Concerns
- Common Criteria

Additionally, the following security-related criteria are highly encouraged to be evaluated when selecting a hypervisor for OpenStack deployments:

- Has the hypervisor undergone Common Criteria certification? If so, to what levels?
- Is the underlying cryptography certified by a third-party?

Team Expertise

Most likely, the most important aspect in hypervisor selection is the expertise of your staff in managing and maintaining a particular hypervisor platform. The more familiar your team is with a given product, its configuration, and its eccentricities, the less likely will there be configuration mistakes. Additionally, having staff expertise spread across an organization on a given hypervisor will increase availability of your systems, allow for developing a segregation of duties, and mitigate problems in the event that a team member is unavailable.

Product or Project Maturity

The maturity of a given hypervisor product or project is critical to your security posture as well. Product maturity will have a number of effects

once you have deployed your cloud, in the context of this security guide we are interested in the following:

- Availability of expertise
- Active developer and user communities
- Timeliness and Availability of updates
- Incidence response

One of the biggest indicators of a hypervisor's maturity is the size and vibrancy of the community that surrounds it. As this concerns security, the quality of the community will affect the availability of expertise should you need additional cloud operators. It is also a sign of how widely deployed the hypervisor is, in turn leading to the battle readiness of any reference architectures and best practices.

Further, the quality of community, as it surrounds an open source hypervisor like KVM or Xen, will have a direct impact on the timeliness of bug fixes and security updates. When investigating both commercial and open source hypervisors, you will want to look into their release and support cycles as well as the time delta between the announcement of a bug or security issue and a patch or response. Lastly, the supported capabilities of OpenStack compute vary depending on the hypervisor chosen. Refer to the OpenStack Hypervisor Support Matrix for OpenStack compute feature support by hypervisor.

Certifications and Attestations

One additional consideration when selecting a hypervisor is the availability of various formal certifications and attestations. While they may not be requirements for your specific organization, these certifications and attestations speak to the maturity, production readiness, and thoroughness of the testing a particular hypervisor platform has been subjected to.

Common Criteria

Common Criteria is an internationally standardized software evaluation process, used by governments and commercial companies to validate software technologies perform as advertised. In the government sector, NSTISSP No. 11 mandates that U.S. Government agencies only procure software which has been Common Criteria certified, a policy which has been in place since July 2002. It should be specifically noted that

OpenStack has not undergone Common Criteria certification, however many of the available hypervisors have.

In addition to validating a technologies capabilities, the Common Criteria process evaluates *how* technologies are developed.

- How is source code management performed?
- How are users granted access to build systems?
- Is the technology cryptographically signed before distribution?

The KVM hypervisor has been Common Criteria certified through the U.S. Government and commercial distributions, which have been validated to separate the runtime environment of virtual machines from each other, providing foundational technology to enforce instance isolation. In addition to virtual machine isolation, KVM has been Common Criteria certified to

> "*provide system-inherent separation mechanisms to the resources of virtual machines. This separation ensures that large software component used for virtualizing and simulating devices executing for each virtual machine cannot interfere with each other. Using the SELinux multi-category mechanism, the virtualization and simulation software instances are isolated. The virtual machine management framework configures SELinux multi-category settings transparently to the administrator*"

While many hypervisor vendors, such as Red Hat, Microsoft, and VMWare have achieved Common Criteria Certification their underlying certified feature set differs. It is recommended to evaluate vendor claims to ensure they minimally satisfy the following requirements:

Identification and Authentication	Identification and authentication using pluggable authentication modules (PAM) based upon user passwords. The quality of the passwords used can be enforced through configuration options.
Audit	The system provides the capability to audit a large number of events including individual system calls as well as events generated by trusted processes. Audit data is collected in regular files in ASCII format. The system provides a program for the purpose of searching the audit records.

	The system administrator can define a rule base to restrict auditing to the events they are interested in. This includes the ability to restrict auditing to specific events, specific users, specific objects or a combination of all of this. Audit records can be transferred to a remote audit daemon.
Discretionary Access Control	Discretionary Access Control (DAC) restricts access to file system objects based on Access Control Lists (ACLs) that include the standard UNIX permissions for user, group and others. Access control mechanisms also protect IPC objects from unauthorized access. The system includes the ext4 file system, which supports POSIX ACLs. This allows defining access rights to files within this type of file system down to the granularity of a single user.
Mandatory Access Control	Mandatory Access Control (MAC) restricts access to objects based on labels assigned to subjects and objects. Sensitivity labels are automatically attached to processes and objects. The access control policy enforced using these labels is derived from the BellLaPadula access control model. SELinux categories are attached to virtual machines and its resources. The access control policy enforced using these categories grant virtual machines access to resources if the category of the virtual machine is identical to the category of the accessed resource. The TOE implements non-hierarchical categories to control access to virtual machines.
Role-Based Access Control	Role-based access control (RBAC) allows separation of roles to eliminate the need for an all-powerful system administrator.
Object Reuse	File system objects as well as memory and IPC objects will be cleared before they can be reused by a process belonging to a different user.

Security Management	The management of the security critical parameters of the system is performed by administrative users. A set of commands that require root privileges (or specific roles when RBAC is used) are used for system management. Security parameters are stored in specific files that are protected by the access control mechanisms of the system against unauthorized access by users that are not administrative users.
Secure Communication	The system supports the definition of trusted channels using SSH. Password based authentication is supported. Only a restricted number of cipher suites are supported for those protocols in the evaluated configuration.
Storage Encryption	The system supports encrypted block devices to provide storage confidentiality via dm_crypt.
TSF Protection	While in operation, the kernel software and data are protected by the hardware memory protection mechanisms. The memory and process management components of the kernel ensure a user process cannot access kernel storage or storage belonging to other processes. Non-kernel TSF software and data are protected by DAC and process isolation mechanisms. In the evaluated configuration, the reserved user ID root owns the directories and files that define the TSF configuration. In general, files and directories containing internal TSF data (e.g., configuration files, batch job queues) are also protected from reading by DAC permissions. The system and the hardware and firmware components are required to be physically protected from unauthorized access. The system kernel mediates all access to the hardware mechanisms themselves, other than program visible CPU instruction functions. In addition, mechanisms for protection against stack overflow attacks are provided.

Cryptography Standards

Several cryptography algorithms are available within OpenStack for identification and authorization, data transfer and protection of data at rest. When selecting a hypervisor, the following are recommended algorithms and implementation standards to ensure the virtualization layer supports:

Algorithm	Key Length	Intended Purpose	Security Function	Implementation Standard
AES	128 bits,192 bits, 256 bits	Encryption / Decryption	Protected Data Transfer, Protection for Data at Rest	RFC 4253
TDES	168 bits	Encryption / Decryption	Protected Data Transfer	RFC 4253
RSA	1024 bits,2048 bits, 3072 bits	Authentication, Key Exchange	Identification and Authentication, Protected Data Transfer	U.S. NIST FIPS PUB 186-3
DSA	L=1024,N=160 bits	Authentication, Key Exchange	Identification and Authentication, Protected Data Transfer	U.S. NIST FIPS PUB 186-3
Serpent	128, 196, or256 bit	Encryption / Decryption	Protection of Data at Rest	http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf
Twofish	128, 196, or256 bit	Encryption / Decryption	Protection of Data at Rest	http://www.schneier.com/paper-twofish-paper.html
SHA-1	-	MessageDigest	Protection of Data at Rest,Protected Data Transfer	U.S. NIST FIPS 180-3
SHA-2(224-, 256-, 384-, 512 bit)	-	MessageDigest	Protection for Data at Rest,Identification and Authentication	U.S. NIST FIPS 180-3

FIPS 140-2

In the United States the National Institute of Science and Technology (NIST) certifies cryptographic algorithms through a process known the

Cryptographic Module Validation Program. NIST certifies algorithms for conformance against Federal Information Processing Standard 140-2 (FIPS 140-2), which ensures:

> *Products validated as conforming to FIPS 140-2 are accepted by the Federal agencies of both countries [United States and Canada] for the protection of sensitive information (United States) or Designated Information (Canada). The goal of the CMVP is to promote the use of validated cryptographic modules and provide Federal agencies with a security metric to use in procuring equipment containing validated cryptographic modules.*

When evaluating base hypervisor technologies, consider if the hypervisor has been certified against FIPS 140-2. Not only is conformance against FIPS 140-2 mandated per U.S. Government policy, formal certification indicates that a given implementation of a cryptographic algorithm has been reviewed for conformance against module specification, cryptographic module ports and interfaces; roles, services, and authentication; finite state model; physical security; operational environment; cryptographic key management; electromagnetic interference/electromagnetic compatibility (EMI/EMC); self-tests; design assurance; and mitigation of other attacks.

Hardware Concerns

Further, when evaluating a hypervisor platform the supportability of the hardware the hypervisor will run on should be considered. Additionally, consider the additional features available in the hardware and how those features are supported by the hypervisor you chose as part of the OpenStack deployment. To that end, hypervisors will each have their own hardware compatibility lists (HCLs). When selecting compatible hardware it is important to know in advance which hardware-based virtualization technologies are important from a security perspective.

Description	Technology	Explanation
I/O MMU	VT-d / AMD-Vi	Required for protecting PCI-passthrough
Intel Trusted Execution Technology	Intel TXT / SEM	Required for dynamic attestation services
PCI-SIG I/O virtualization	SR-IOV, MR-IOV, ATS	Required to allow secure sharing of PCI Express devices
Network virtualization	VT-c	Improves performance of network I/O on hypervisors

Hypervisor vs. Baremetal

To wrap up our discussion around hypervisor selection, it is important to call out the differences between using LXC (Linux Containers) or Baremetal systems vs using a hypervisor like KVM. Specifically, the focus of this security guide will be largely based on having a hypervisor and virtualization platform. However, should your implementation require the use of a baremetal or LXC environment, you will want to pay attention to the particular differences in regard to deployment of that environment. In particular, you will need to provide your end users with assurances that the node has been properly sanitized of their data prior to re-provisioning. Additionally, prior to reusing a node, you will need to provide assurances that the hardware has not been tampered or otherwise compromised.

It should be noted that while OpenStack has a baremetal project, a discussion of the particular security implications of running baremetal is beyond the scope of this book.

Finally, due to the time constraints around a book sprint, the team chose to use KVM as the hypervisor in our example implementations and architectures.

Note

There is an OpenStack Security Note pertaining to the use of LXC in Nova.

Additional Security Features

Another thing to look into when selecting a hypervisor platform is the availability of specific security features. In particular, we are referring to features like Xen Server's XSM or Xen Security Modules, sVirt, Intel TXT, and AppArmor. The presence of these features will help increase your security profile as well as provide a good foundation.

The following table calls out these features by common hypervisor platforms.

	KSM	XSM	sVirt	TXT	AppArmor	cGroups	MAC Policy
KVM	X		X	X	x	x	x
Xen		X		X			x
ESXi				X			

Hyper-V							

KSM: Kernel Samepage Merging

XSM: Xen Security Modules

xVirt: Mandatory Access Control for Linux-based virtualization

TXT: Intel Trusted Execution Technology

AppArmor: Linux security module implementing MAC

cgroups: Linux kernel feature to control resource usage

MAC Policy: Mandatory Access Control; may be implemented with SELinux or other operating systems

* Features in this table may not be applicable to all hypervisors or directly mappable between hypervisors.

41. Hardening the Virtualization Layers

In the beginning of this chapter we discuss the use of both physical and virtual hardware by instances, the associated security risks, and some recommendations for mitigating those risks. We conclude the chapter with a discussion of sVirt, an open source project for integrating SELinux mandatory access controls with the virtualization components.

Physical Hardware (PCI Passthrough)

Many hypervisors offer a functionality known as PCI passthrough. This allows an instance to have direct access to a piece of hardware on the node. For example, this could be used to allow instances to access video cards offering the compute unified device architecture (CUDA) for high performance computation. This feature carries two types of security risks: direct memory access and hardware infection.

Direct memory access (DMA) is a feature that permits certain hardware devices to access arbitrary physical memory addresses in the host computer. Often video cards have this capability. However, an instance should not be given arbitrary physical memory access because this would give it full view of both the host system and other instances running on the same node. Hardware vendors use an input/output memory management unit (IOMMU) to manage DMA access in these situations. Therefore, cloud architects should ensure that the hypervisor is configured to utilize this hardware feature.

- KVM: How to assign devices with VT-d in KVM
- Xen: VTd Howto

Note

The IOMMU feature is marketed as VT-d by Intel and AMD-Vi by AMD.

A hardware infection occurs when an instance makes a malicious modification to the firmware or some other part of a device. As this device

is used by other instances, or even the host OS, the malicious code can spread into these systems. The end result is that one instance can run code outside of its security domain. This is a potential problem in any hardware sharing scenario. The problem is specific to this scenario because it is harder to reset the state of physical hardware than virtual hardware.

Solutions to the hardware infection problem are domain specific. The strategy is to identify how an instance can modify hardware state then determine how to reset any modifications when the instance is done using the hardware. For example, one option could be to re-flash the firmware after use. Clearly there is a need to balance hardware longevity with security as some firmwares will fail after a large number of writes. TPM technology, described in `link:Management/Node Bootstrapping`, provides a solution for detecting unauthorized firmware changes. Regardless of the strategy selected, it is important to understand the risks associated with this kind of hardware sharing so that they can be properly mitigated for a given deployment scenario.

Additionally, due to the risk and complexities associated with PCI passthrough, it should be disabled by default. If enabled for a specific need, you will need to have appropriate processes in place to ensure the hardware is clean before re-issue.

Virtual Hardware (QEMU)

When running a virtual machine, virtual hardware is a software layer that provides the hardware interface for the virtual machine. Instances use this functionality to provide network, storage, video, and other devices that may be needed. With this in mind, most instances in your environment will exclusively use virtual hardware, with a minority that will require direct hardware access. The major open source hypervisors use QEMU for this functionality. While QEMU fills an important need for virtualization platforms, it has proven to be a very challenging software project to write and maintain. Much of the functionality in QEMU is implemented with low-level code that is difficult for most developers to comprehend. Furthermore, the hardware virtualized by QEMU includes many legacy devices that have their own set of quirks. Putting all of this together, QEMU has been the source of many security problems, including hypervisor breakout attacks.

For the reasons stated above, it is important to take proactive steps to harden QEMU. We recommend three specific steps: minimizing the codebase, using compiler hardening, and using mandatory access controls, for example: sVirt, SELinux, or AppArmor.

Minimizing the Qemu Codebase

One classic security principle is to remove any unused components from your system. QEMU provides support for many different virtual hardware devices. However, only a small number of devices are needed for a given instance. Most instances will use the virtio devices. However, some legacy instances will need access to specific hardware, which can be specified using glance metadata:

```
glance image-update \
    --property hw_disk_bus=ide \
    --property hw_cdrom_bus=ide \
    --property hw_vif_model=e1000 \
    f16-x86_64-openstack-sda
```

A cloud architect should decide what devices to make available to cloud users. Anything that is not needed should be removed from QEMU. This step requires recompiling QEMU after modifying the options passed to the QEMU configure script. For a complete list of up-to-date options simply run `./configure --help` from within the QEMU source directory. Decide what is needed for your deployment, and disable the remaining options.

Compiler Hardening

The next step is to harden QEMU using compiler hardening options. Modern compilers provide a variety of compile time options to improve the security of the resulting binaries. These features, which we will describe in more detail below, include relocation read-only (RELRO), stack canaries, never execute (NX), position independent executable (PIE), and address space layout randomization (ASLR).

Many modern linux distributions already build QEMU with compiler hardening enabled, so you may want to verify your existing executable before proceeding with the information below. One tool that can assist you with this verification is called `checksec.sh`.

- *RELocation Read-Only (RELRO)*: Hardens the data sections of an executable. Both full and partial RELRO modes are supported by gcc. For QEMU full RELRO is your best choice. This will make the global offset table read-only and place various internal data sections before the program data section in the resulting executable.

- *Stack Canaries*: Places values on the stack and verifies their presence to help prevent buffer overflow attacks.

- *Never eXecute (NX)*: Also known as Data Execution Prevention (DEP), ensures that data sections of the executable can not be executed.

- *Position Independent Executable (PIE)*: Produces a position independent executable, which is necessary for ASLR.

- *Address Space Layout Randomization (ASLR)* : This ensures that both code and data regions will be randomized. Enabled by the kernel (all modern linux kernels support ASLR), when the executable is built with PIE.

Putting this all together, and adding in some additional useful protections, we recommend the following compiler options for gcc when compiling QEMU:

```
CFLAGS="-arch x86_64 -fstack-protector-all -Wstack-protector --
param ssp-buffer-size=4 -pie -fPIE -ftrapv -D_FORTIFY_SOURCE=2
O2 -Wl,-z,relro,-z,now"
```

We recommend testing your QEMU executable file after it is compiled to ensure that the compiler hardening worked properly.

Most cloud deployments will not want to build software such as QEMU by hand. It is better to use packaging to ensure that the process is repeatable and to ensure that the end result can be easily deployed throughout the cloud. The references below provide some additional details on applying compiler hardening options to existing packages.

- DEB packages: Hardening Walkthrough

- RPM packages: How to create an RPM package

Mandatory Access Controls

Compiler hardening makes it more difficult to attack the QEMU process. However, if an attacker does succeed, we would like to limit the impact of the attack. Mandatory access controls accomplish this by restricting the privileges on QEMU process to only what is needed. This can be accomplished using sVirt / SELinux or AppArmor. When using sVirt, SELinux is configured to run every QEMU process under a different security context. AppArmor can be configured to provide similar functionality. We provide more details on sVirt in the instance isolation section below.

sVirt: SELinux + Virtualization

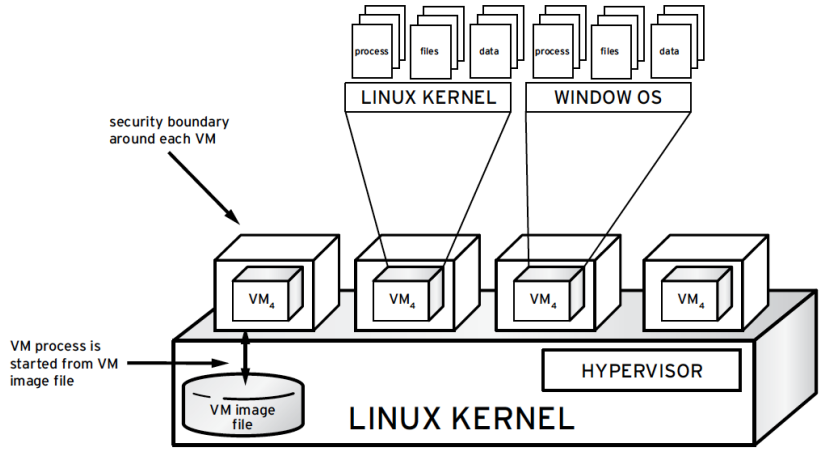
With unique kernel-level architecture and National Security Agency (NSA) developed security mechanisms, KVM provides foundational isolation technologies for multitenancy. With developmental origins dating back to 2002, the Secure Virtualization (sVirt) technology is the application of SELinux against modern day virtualization. SELinux, which was designed to apply separation control based upon labels, has been extended to provide isolation between virtual machine processes, devices, data files and system processes acting upon their behalf.

OpenStack's sVirt implementation aspires to protect hypervisor hosts and virtual machines against two primary threat vectors:

- **Hypervisor threats** A compromised application running within a virtual machine attacks the hypervisor to access underlying resources (e.g. the host OS, applications, or devices within the physical machine). This is a threat vector unique to virtualization and represents considerable risk as the underlying real machine can be compromised due to vulnerability in a single virtual application.

- **Virtual Machine (multi-tenant) threats** A compromised application running within a VM attacks the hypervisor to access/control another virtual machine and its resources. This is a threat vector unique to virtualization and represents considerable risk as a multitude of virtual machine file images could be compromised due to vulnerability in a single application. This virtual network attack is a major concern as the administrative techniques for protecting real networks do not directly apply to the virtual environment.

Each KVM-based virtual machine is a process which is labeled by SELinux, effectively establishing a security boundary around each virtual machine. This security boundary is monitored and enforced by the Linux kernel, restricting the virtual machine's access to resources outside of its boundary such as host machine data files or other VMs.



As shown above, sVirt isolation is provided regardless of the guest Operating System running inside the virtual machine – Linux or Windows VMs can be used. Additionally, many Linux distributions provide SELinux within the operating system, allowing the virtual machine to protect internal virtual resources from threats.

Labels and Categories

KVM-based virtual machine instances are labelled with their own SELinux data type, known as svirt_image_t. Kernel level protections prevent unauthorized system processes, such as malware, from manipulating the virtual machine image files on disk. When virtual machines are powered off, images are stored as svirt_image_t as shown below:

```
system_u:object_r:svirt_image_t:SystemLow image1
system_u:object_r:svirt_image_t:SystemLow image2
system_u:object_r:svirt_image_t:SystemLow image3
system_u:object_r:svirt_image_t:SystemLow image4
```

The svirt_image_t label uniquely identifies image files on disk, allowing for the SELinux policy to restrict access. When a KVM-based Nova image is powered on, sVirt appends a random numerical identifier to the image. sVirt is technically capable of assigning numerical identifiers to 524,288 virtual machines per hypervisor node, however OpenStack deployments are highly unlikely to encounter this limitation.

An example of the sVirt category identifier is shown below:

```
system_u:object_r:svirt_image_t:s0:c87,c520 image1
system_u:object_r:svirt_image_t:s0:419,c172 image2
```

Booleans

To ease the administrative burden of managing SELinux, many enterprise Linux platforms utilize SELinux Booleans to quickly change the security posture of sVirt.

Red Hat Enterprise Linux-based KVM deployments utilize the following sVirt booleans:

sVirt SELinux Boolean	Description
virt_use_common	Allow virt to use serial/parallel communication ports.
virt_use_fusefs	Allow virt to read FUSE mounted files.
virt_use_nfs	Allow virt to manage NFS mounted files.
virt_use_samba	Allow virt to manage CIFS mounted files.
virt_use_sanlock	Allow confined virtual guests to interact with the sanlock.
virt_use_sysfs	Allow virt to manage device configuration (PCI).
virt_use_usb	Allow virt to use USB devices.
virt_use_xserver	Allow virtual machine to interact with the X Window System.

42. Case Studies

In this case study we discuss how Alice and Bob would ensure that their instances are properly isolated. First we consider hypervisor selection, and then techniques for hardening QEMU and applying mandatory access controls.

Alice's Private Cloud

Alice chooses Xen for the hypervisor in her cloud due to a strong internal knowledge base and a desire to use the Xen security modules (XSM) for fine-grained policy enforcement.

Alice is willing to apply a relatively large amount of resources to software packaging and maintenance. She will use these resources to build a highly customized version of QEMU that has many components removed, thereby reducing the attack surface. She will also ensure that all compiler hardening options are enabled for QEMU. Alice accepts that these decisions will increase long-term maintenance costs.

Alice writes XSM policies (for Xen) and SELinux policies (for Linux domain 0, and device domains) to provide stronger isolation between the instances. Alice also uses the Intel TXT support in Xen to measure the hypervisor launch in the TPM.

Bob's Public Cloud

Bob is very concerned about instance isolation since the users in a public cloud represent anyone with a credit card, meaning they are inherently untrusted. Bob has just started hiring the team that will deploy the cloud, so he can tailor his candidate search for specific areas of expertise. With this in mind, Bob chooses a hypervisor based on its technical features, certifications, and community support. KVM has an EAL 4+ common criteria rating, with a layered security protection profile (LSPP) to provide added assurance for instance isolation. This, combined with the strong support for KVM within the OpenStack community drives Bob's decision to use KVM.

Bob weighs the added cost of repackaging QEMU and decides that he cannot commit those resources to the project. Fortunately, his Linux

distribution has already enabled the compiler hardening options. So he decides to use this QEMU package. Finally, Bob leverages sVirt to manage the SELinux polices associated with the virtualization stack.

43. Security Services for Instances

One of the virtues of running instances in a virtualized environments is that it opens up new opportunities for security controls that are not typically available when deploying onto bare metal. There are several technologies that can be applied to the virtualization stack that bring improved information assurance for cloud tenants.

Deployers or users of OpenStack with strong security requirements may want to consider deploying these technologies. Not all are applicable in every situation, indeed in some cases technologies may be ruled out for use in a cloud because of prescriptive business requirements. Similarly some technologies inspect instance data such as run state which may be undesirable to the users of the system.

In this chapter we explore these technologies and describe the situations where they can be used to enhance security for instances or underlying instances. We also seek to highlight where privacy concerns may exist. These include data pass through, introspection, or providing a source of entropy. In this section we highlight the following additional security services:

- Entropy to Instances
- Scheduling Instances to Nodes
- Trusted Images
- Instance Migrations

Entropy To Instances

We consider entropy to refer to the quality and source of random data that is available to an instance. Cryptographic technologies typically rely heavily on randomness, requiring a high quality pool of entropy to draw from. It is typically hard for a virtual machine to get enough entropy to support these operations. Entropy starvation can manifest in instances

as something seemingly unrelated for example, slow boot times because the instance is waiting for ssh key generation. Entropy starvation may also motivate users to employ poor quality entropy sources from within the instance, making applications running in the cloud less secure overall.

Fortunately, a cloud architect may address these issues by providing a high quality source of entropy to the cloud instances. This can be done by having enough hardware random number generators (HRNG) in the cloud to support the instances. In this case, "enough" is somewhat domain specific. For everyday operations, a modern HRNG is likely to produce enough entropy to support 50-100 compute nodes. High bandwidth HRNGs, such as the RdRand instruction available with Intel Ivy Bridge and newer processors could potentially handle more nodes. For a given cloud, an architect needs to understand the application requirements to ensure that sufficient entropy is available.

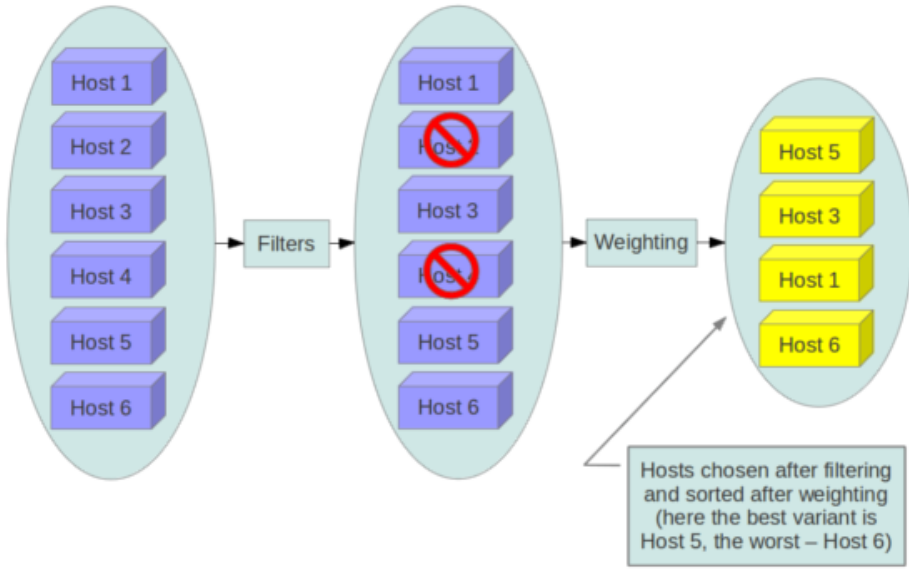
Once the entropy is available in the cloud, the next step is getting that entropy into the instances. Tools such as the entropy gathering daemon (EGD) provide a way to fairly and securely distribute entropy through a distributed system. Support exists for using the EGD as an entropy source for LibVirt.

Nova support for these features is not generally available, but it would only require a moderate amount of work for implementors to integrate this functionality.

Scheduling Instances to Nodes

Before an instance is created, a host for the image instantiation must be selected. This selection is performed by the `nova-scheduler` which determines how to dispatch compute and volume requests.

The default nova scheduler in Grizzly is the Filter Scheduler, although other schedulers exist (see the section Scheduling in the *OpenStack Configuration Reference*). The filter scheduler works in collaboration with 'filters' to decide where an instance should be started. This process of host selection allows administrators to fulfil many different security requirements. Depending on the cloud deployment type for example, one could choose to have tenant instances reside on the same hosts whenever possible if data isolation was a primary concern, conversely one could attempt to have instances for a tenant reside on as many different hosts as possible for availability or fault tolerance reasons. The following diagram demonstrates how the filter scheduler works:



The use of scheduler filters may be used to segregate customers, data, or even discard machines of the cloud that cannot be attested as secure. This generally applies to all OpenStack projects offering a scheduler. When building a cloud, you may choose to implement scheduling filters for a variety of security-related purposes.

Below we highlight a few of the filters that may be useful in a security context, depending on your requirements, the full set of filter documentation is documented in the Filter Scheduler section of the *OpenStack Configuration Reference*.

Tenant Driven Whole Host Reservation

There currently exists a blueprint for whole host reservation - This would allow a tenant to exclusively reserve hosts for only it's instances, incurring extra costs.

Host Aggregates

While not a filter in themselves, host aggregates allow administrators to assign key-value pairs to groups of machines. This allows cloud administrators, not users, to partition up their compute host resources. Each node can have multiple aggregates (see the Host Aggregates section

of the *OpenStack Configuration Reference* for more information on creating and managing aggregates).

AggregateMultiTenancyIsolation

Isolates tenants to specific host aggregates. If a host is in an aggregate that has the metadata key `filter_tenant_id` it will only create instances from that tenant (or list of tenants). A host can be in multiple aggregates. If a host does not belong to an aggregate with the metadata key, it can create instances from all tenants.

DifferentHostFilter

Schedule the instance on a different host from a set of instances. To take advantage of this filter, the requester must pass a scheduler hint, using `different_host` as the key and a list of instance uuids as the value. This filter is the opposite of the `SameHostFilter`.

GroupAntiAffinityFilter

The GroupAntiAffinityFilter ensures that each instance in a group is on a different host. To take advantage of this filter, the requester must pass a scheduler hint, using `group` as the key and a list of instance uuids as the value.

Trusted Compute Pools

There exists a scheduler filter which integrates with the Open Attestation Project (OATS) to define scheduler behavior according to the attestation of PCRs received from a system using Intel TXT.

It is unclear if this feature is compatible with AMD's similar SEM, although the OpenAttestation agent relies on the vendor-agnostic TrouSerS library.

Trusted Images

With regards to images, users will be working with pre-installed images or images that they upload themselves. In both cases, users will want to ensure that the image they are ultimately running has not been tampered with. This requires some source of truth such as a checksum for the known good version of an image as well as verification of the running image. This section describes the current best practices around image handling, while also calling out some of the existing gaps in this space.

Image Creation Process

The OpenStack Documentation provides guidance on how to create and upload an image to Glance. Additionally it is assumed that you have a process by which you install and harden operating systems. Thus, the following items will provide additional guidance on how to ensure your images are built securely prior to upload. There are a variety of options for obtaining images. Each has specific steps that help validate the image's provenance.

The first option is to obtain boot media from a trusted source.

```
mkdir -p /tmp/download_directorycd /tmp/download_directory

wget http://mirror.anl.gov/pub/ubuntu-iso/CDs/precise/ubuntu-12.
04.2-server-amd64.iso
wget http://mirror.anl.gov/pub/ubuntu-iso/CDs/precise/SHA256SUMS
wget http://mirror.anl.gov/pub/ubuntu-iso/CDs/precise/
SHA256SUMS.gpg
gpg --keyserver hkp://keyserver.ubuntu.com --recv-
keys 0xFBB75451
gpg --verify SHA256SUMS.gpg SHA256SUMSsha256sum -c SHA256SUMS
 2>&1 | grep OK
```

The second option is to use the *OpenStack Virtual Maschine Image Guide*. In this case, you will want to follow your organizations OS hardening guidelines or those provided by a trusted third-party such as the RHEL6 STIG.

The final option is to use an automated image builder. The following example uses the Oz image builder. The OpenStack community has recently created a newer tool worth investigating: disk-image-builder. We have not evaluated this tool from a security perspective.

Example of RHEL 6 CCE-26976-1 which will help implement NIST 800-53 Section *AC-19(d) in* Oz.

```
<template>
<name>centos64</name>
<os>
  <name>RHEL-6</name>
  <version>4</version>
  <arch>x86_64</arch>
  <install type='iso'>
  <iso>http://trusted_local_iso_mirror/isos/x86_64/RHEL-6.4-
x86_64-bin-DVD1.iso</iso>
```

```
  </install>
  <rootpw>CHANGE THIS TO YOUR ROOT PASSWORD</rootpw>
</os>
<description>RHEL 6.4 x86_64</description>
<repositories>
  <repository name='epel-6'>
    <url>http://download.fedoraproject.org/pub/epel/6/
$basearch</url>
    <signed>no</signed>
  </repository>
</repositories>
<packages>
  <package name='epel-release'/>
  <package name='cloud-utils'/>
  <package name='cloud-init'/>
</packages>
<commands>
  <command name='update'>
  yum update
  yum clean all
  sed -i '/^HWADDR/d' /etc/sysconfig/network-scripts/ifcfg-eth0
  echo -n > /etc/udev/rules.d/70-persistent-net.rules
  echo -n > /lib/udev/rules.d/75-persistent-net-generator.rules
  chkconfig --level 0123456 autofs off
  service autofs stop
  </command>
</commands>
</template>
```

Note, it is the recommendation of this guide to shy away from the manual image building process as it is complex and prone to error. Further, using an automated system like Oz or disk-image-builder for image building, or a configuration management utility like Chef or Puppet for post boot image hardening gives you the ability to produce a consistent image as well as track compliance of your base image to its respective hardening guidelines over time.

If subscribing to a public cloud service, you should check with the cloud provider for an outline of the process used to produce their default images. If the provider allows you to upload your own images, you will want to ensure that you are able to verify that your image was not modified before you spin it up. To do this, refer to the following section on Image Provenance.

Image Provenance and Validation

Unfortunately, it is not currently possible to force Nova to validate an image hash immediately prior to starting an instance. To understand the

situation, we begin with a brief overview of how images are handled around the time of image launch.

Images come from the glance service to the nova service on a node. This transfer should be protected by running over SSL. Once the image is on the node, it is verified with a basic checksum and then it's disk is expanded based on the size of the instance being launched. If, at a later time, the same image is launched with the same instance size on this node, it will be launched from the same expanded image. Since this expanded image is not re-verified before launching, it could be tampered with and the user would not have any way of knowing, beyond a manual inspection of the files in the resulting image.

We hope that future versions of Nova and/or Glance will offer support for validating the image hash before each instance launch. An alternative option that would be even more powerful would be allow users to sign an image and then have the signature validated when the instance is launched.

Instance Migrations

OpenStack and the underlying virtualization layers provide for the Live Migration of images between OpenStack nodes allowing you to seamlessly perform rolling upgrades of your OpenStack Compute nodes without instance downtime. However, Live Migrations also come with their fair share of risk. To understand the risks involved, it is important to first understand how a live migration works. The following are the high level steps preformed during a live migration.

1. Start instance on destination host
2. Transfer memory
3. Stop the guest & sync disks
4. Transfer state
5. Start the guest

Live Migration Risks

At various stages of the live migration process the contents of an instances run time memory and disk are transmitted over the network in plain text.

Thus there are several risks that need to be addressed when using live migration. The following in-exhaustive list details some of these risks:

- *Denial of Service (DoS)* : If something fails during the migration process, the instance could be lost.
- *Data Exposure* : Memory or disk transfers must be handled securely.
- *Data Manipulation* : If memory or disk transfers are not handled securely, then an attacker could manipulate user data during the migration.
- *Code Injection* : If memory or disk transfers are not handled securely, then an attacker could manipulate executables, either on disk or in memory, during the migration.

Live Migration Mitigations

There are several methods to mitigate some of the risk associated with live migrations, the following list details some of these:

- Disable Live Migration
- Isolated Migration Network
- Encrypted Live Migration

Disable Live Migration

At this time, live migration is enabled in OpenStack by default. Live migrations can be disabled by adding the following lines to the nova policy.json file:

```
"compute_extension:admin_actions:migrate": "!",
"compute_extension:admin_actions:migrateLive": "!",
```

Migration Network

As a general practice, live migration traffic should be restricted to the management security domain. Indeed live migration traffic, due to its plain text nature and the fact that you are transferring the contents of disk and memory of a running instance, it is recommended you further separate live migration traffic onto a dedicated network. Isolating the traffic to a dedicated network can reduce the risk of exposure.

Encrypted Live Migration

If your use case involves keeping live migration enabled, then libvirtd can provide tunneled, encrypted live migrations. That said, this feature is not currently exposed in OpenStack Dashboard, nor the nova-client commands and can only be accessed through manual configuration of libvritd. Encrypted live migration modifies the live migration process by first copying the instance data from the running hypervisor to libvirtd. From there an encrypted tunnel is created between the libvirtd processes on both hosts. Finally, the destination libvirtd process copies the instance back to the underlying hypervisor.

44. Case Studies

In this case study we discuss how Alice and Bob would architect their clouds with respect to instance entropy, scheduling instances, trusted images, and instance migrations.

Alice's Private Cloud

Alice has a need for lots of high quality entropy in the instances. For this reason, she decides to purchase hardware with Intel Ivy Bridge chip sets that support the RdRand instruction on each compute node. Using the entropy gathering daemon (EGD) and LibVirt's EGD support, Alice ensures that this entropy pool is distributed to the instances on each compute node.

For instance scheduling, Alice uses the trusted compute pools to ensure that all cloud workloads are deployed to nodes that presented a proper boot time attestation. Alice decides to disable user permissions for image uploading to help ensure that the images used in the cloud are generated in a known and trusted manner by the cloud administrators.

Finally, Alice disables instance migrations as this feature is less critical for the high performance application workloads expected to run in this cloud. This helps avoid the various security concerns related to instance migrations.

Bob's Public Cloud

Bob is aware that entropy will be a concern for some of his customers, such as those in the financial industry. However, due to the added cost and complexity, Bob has decided to forgo integrating hardware entropy into the first iteration of his cloud. He adds hardware entropy as a fast-follow to do for a later improvement for the second generation of his cloud architecture.

Bob is interested in ensuring that customers receive a high quality of service. He is concerned that providing too much explicit user control over instance scheduling could negatively impact the quality of service. So he disables this feature. Bob provides images in the cloud from a known

trusted source for users to use. Additionally, he also allows users to upload their own images. However, users cannot generally share their images. This helps prevent a user from sharing a malicious image, which could negatively impact the security of other users in the cloud.

For migrations, Bob wants to enable secure instance migrations in order to support rolling upgrades with minimal user downtime. Bob ensures that all migrations occur on an isolated VLAN. He plans to defer implementing encrypted migrations until this is better supported in Nova client tools. However, he makes a note to track this carefully and switch to encryption migrations as soon as possible.

45. Forensics and Incident Response

A lot of activity goes on within a cloud environment. It is a mix of hardware, operating systems, virtual machine managers, the OpenStack services, cloud-user activity such as creating instances and attaching storage, the network underlying the whole, and finally end-users using the applications running on the various instances.

The generation and collection of logs is an important component of securely monitoring an OpenStack infrastructure. Logs provide visibility into the day-to-day actions of administrators, tenants, and guests, in addition to the activity in the compute, networking, and storage and other components that comprise your OpenStack deployment.

The basics of logging: configuration, setting log level, location of the log files, and how to use and customize logs, as well as how to do centralized collections of logs is well covered in the *OpenStack Operations Guide*.

Logs are not only valuable for proactive security and continuous compliance activities, but they are also a valuable information source for investigating and responding to incidents.

For instance, analyzing the access logs of Keystone or its replacement authentication system would alert us to failed logins, their frequency, origin IP, whether the events are restricted to select accounts etc. Log analysis supports detection.

On detection, further action may be to black list an IP, or recommend strengthening user passwords, or even de-activating a user account if it is deemed dormant.

Monitoring Use Cases

Monitoring events is more pro-active and provides real-time detection and response. There are several tools to aid in monitoring.

In the case of a OpenStack cloud instance, we need to monitor the hardware, the OpenStack services, and the cloud resource usage. The last

stems from wanting to be elastic, to scale to the dynamic needs of the users.

Here are a few important use cases to consider when implementing log aggregation, analysis and monitoring. These use cases can be implemented and monitored through various commercial and open source tools, homegrown scripts, etc. These tools and scripts can generate events that can then be sent to the administrators through email or integrated dashboard. It is important to consider additional use cases that may apply to your specific network and what you may consider anomalous behavior.

- Detecting the absence of log generation is an event of high value. Such an event would indicate a service failure or even an intruder who has temporarily switched off logging or modified the log level to hide their tracks.

- Application events such as start and/or stop that were unscheduled would also be events to monitor and examine for possible security implications.

- OS events on the OpenStack service machines such as user logins, restarts also provide valuable insight into use/misuse

- Being able to detect the load on the OpenStack servers also enables responding by way of introducing additional servers for load balancing to ensure high availability.

- Other events that are actionable are networking bridges going down, ip tables being flushed on nova compute nodes and consequential loss of access to instances resulting in unhappy customers.

- To reduce security risks from orphan instances on a user/tenant/ domain deletion in the Identity service there is discussion to generate notifications in the system and have OpenStack components respond to these events as appropriate such as terminating instances, disconnecting attached volumes, reclaiming CPU and storage resources etc.

A cloud will host many virtual instances, and monitoring these instances goes beyond hardware monitoring and log files which may just contain CRUD events.

Security monitoring controls such as intrusion detection software, antivirus software, and spyware detection and removal utilities can generate logs that show when and how an attack or intrusion took place. Deploying these tools on the cloud machines provides value and protection. Cloud

users, those running instances on the cloud may also want to run such tools on their instances.

References

http://www.mirantis.com/blog/openstack-monitoring/

http://blog.sflow.com/2012/01/host-sflow-distributed-agent.html

http://blog.sflow.com/2009/09/lan-and-wan.html

http://blog.sflow.com/2013/01/rapidly-detecting-large-flows-sflow-vs.html

46. Case Studies

In this case study we discuss how Alice and Bob would address monitoring and logging in the public vs a private cloud. In both instances, time synchronization and a centralized store of logs become extremely important for performing proper assessments and troubleshooting of anomalies. Just collecting logs is not very useful, a robust monitoring system must be built to generate actionable events.

Alice's Private Cloud

In the private cloud, Alice has a better understanding of the tenants requirements and accordingly can add appropriate oversight and compliance on monitoring and logging. Alice should identify critical services and data and ensure that logging is turned at least on those services and is being aggregated to a central log server. She should start with simple and known use cases and implement correlation and alerting to limit the number of false positives. To implement correlation and alerting, she sends the log data to her organization's existing SIEM tool. Security monitoring should be an ongoing process and she should continue to define use cases and alerts as she has better understanding of the network traffic activity and usage over time.

Bob's Public Cloud

When it comes to logging, as a public cloud provider, Bob is interested in logging both for situational awareness as well as compliance. That is, compliance that Bob as a provider is subject to as well as his ability to provide timely and relevant logs or reports on the behalf of his customers for their compliance audits. With that in mind, Bob configures all of his instances, nodes, and infrastructure devices to perform time synchronization with an external, known good time device. Additionally, Bob's team has built a Django based web applications for his customers to perform self-service log retrieval from Bob's SIEM tool. Bob also uses this SIEM tool along with a robust set of alerts and integration with his CMDB to provide operational awareness to both customers and cloud administrators.

47. Compliance Overview

An OpenStack deployment may require compliance activities for many purposes, such as regulatory and legal requirements, customer need, privacy considerations, and security best practices. Compliance, when done correctly, unifies and strengthens the other security topics discussed in this guide. This chapter has several objectives:

- Review common security principles.
- Discuss common control frameworks and certification resources to achieve industry certifications or regulator attestations.
- Act as a reference for auditors when evaluating OpenStack deployments.
- Introduce privacy considerations specific to OpenStack and cloud environments.

Security Principles

Industry standard security principles provide a baseline for compliance certifications and attestations. If these principles are considered and referenced throughout an OpenStack deployment, certification activities may be simplified.

1. *Layered Defenses*: Identify where risks exist in a cloud architecture and apply controls to mitigate the risks. In areas of significant concern, layered defences provide multiple complementary controls to further mitigate risk. For example, to ensure adequate isolation between cloud tenants, we recommend hardening QEMU, using a hypervisor with SELinux support, enforcing mandatory access control policies, and reducing the overall attack surface. The foundational principle is to harden an area of concern with multiple layers of defense such that if any one layer is compromised, other layers will exist to offer protection and minimize exposure.
2. *Fail Securely*: In the case of failure, systems should be configured to fail into a closed secure state. For example, SSL certificate verification should fail closed by severing the network connection if the CNAME doesn't match the server's DNS name. Software often fails open in this

situation, allowing the connection to proceed without a CNAME match, which is less secure and not recommended.

3. *Least Privilege*: Only the minimum level of access for users and system services is granted. This access is based upon role, responsibility and job function. This security principal of least privilege is written into several international government security policies, such as NIST 800-53 Section AC-6 within the United States.

4. *Compartmentalize*: Systems should be segregated in a such way that if one machine, or system-level service, is compromised the security of the other systems will remain intact. Practically, the enablement and proper usage of SELinux helps accomplish this goal.

5. *Promote Privacy*: The amount of information that can be gathered about a system and its users should be minimized.

6. *Logging Capability*: Appropriate logging is implemented to monitor for unauthorized use, incident response and forensics. It is highly recommended that selected audit subsystems be Common Criteria certified, which provides non-attestable event records in most countries.

48. Understanding the Audit Process

Information system security compliance is reliant on the completion of two foundational processes:

1. **Implementation and Operation of Security Controls**Aligning the information system with in-scope standards and regulations involves internal tasks which must be conducted before a formal assessment. Auditors may be involved at this state to conduct gap analysis, provide guidance, and increase the likelihood of successful certification.

2. **Independent Verification and Validation**Demonstration to a neutral third-party that system security controls are implemented and operating effectively, in compliance with in-scope standards and regulations, is required before many information systems achieve certified status. Many certifications require periodic audits to ensure continued certification, considered part of an overarching continuous monitoring practice.

Determining Audit Scope

Determining audit scope, specifically what controls are needed and how to design or modify an OpenStack deployment to satisfy them, should be the initial planning step.

When scoping OpenStack deployments for compliance purposes, consider prioritizing controls around sensitive services, such as command and control functions and the base virtualization technology. Compromises of these facilities may impact an OpenStack environment in its entirety.

Scope reduction helps ensure OpenStack architects establish high quality security controls which are tailored to a particular deployment, however it is paramount to ensure these practices do not omit areas or features from security hardening. A common example is applicable to PCI-DSS guidelines, where payment related infrastructure may be scrutinized for security issues, but supporting services are left ignored, and vulnerable to attack.

When addressing compliance, you can increase efficiency and reduce work effort by identifying common areas and criteria that apply across multiple certifications. Much of the audit principles and guidelines discussed in this book will assist in identifying these controls, additionally a number of external entities provide comprehensive lists. The following are some examples:

The Cloud Security Alliance Cloud Controls Matrix (CCM) assists both cloud providers and consumers in assessing the overall security of a cloud provider. The CSA CMM provides a controls framework that map to many industry-accepted standards and regulations including the ISO 27001/2, ISACA, COBIT, PCI, NIST, Jericho Forum and NERC CIP.

The SCAP Security Guide is another useful reference. This is still an emerging source, but we anticipate that this will grow into a tool with controls mappings that are more focused on the US federal government certifications and recommendations. For example, the SCAP Security Guide currently has some mappings for security technical implementation guides (STIGs) and NIST-800-53.

These control mappings will help identify common control criteria across certifications, and provide visibility to both auditors and auditees on problem areas within control sets for particular compliance certifications and attestations.

Internal Audit

Once a cloud is deployed, it is time for an internal audit. This is the time compare the controls you identified above with the design, features, and deployment strategies utilized in your cloud. The goal is to understand how each control is handled and where gaps exist. Document all of the findings for future reference.

When auditing an OpenStack cloud it is important to appreciate the multi-tenant environment inherent in the OpenStack architecture. Some critical areas for concern include data disposal, hypervisor security, node hardening, and authentication mechanisms.

Prepare for External Audit

Once the internal audit results look good, it is time to prepare for an external audit. There are several key actions to take at this stage, these are outlined below:

- Maintain good records from your internal audit. These will prove useful during the external audit so you can be prepared to answer questions about mapping the compliance controls to a particular deployment.
- Deploy automated testing tools to ensure that the cloud remains compliant over time.
- Select an auditor.

Selecting an auditor can be challenging. Ideally, you are looking for someone with experience in cloud compliance audits. OpenStack experience is another big plus. Often it is best to consult with people who have been through this process for referrals. Cost can vary greatly depending on the scope of the engagement and the audit firm considered.

External Audit

This is the formal audit process. Auditors will test security controls in scope for a specific certification, and demand evidentiary requirements to prove that these controls were also in place for the audit window (for example SOC 2 audits generally evaluate security controls over a 6-12 months period). Any control failures are logged, and will be documented in the external auditors final report. Dependent on the type of OpenStack deployment, these reports may be viewed by customers, so it is important to avoid control failures. This is why audit preparation is so important.

Compliance Maintenance

The process doesn't end with a single external audit. Most certifications require continual compliance activities which means repeating the audit process periodically. We recommend integrating automated compliance verification tools into a cloud to ensure that it is compliant at all times. This should be in done in addition to other security monitoring tools. Remember that the goal is both security *and* compliance. Failing on either of these fronts will significantly complicate future audits.

49. Compliance Activities

There are a number of standard activities that will greatly assist with the compliance process. In this chapter we outline some of the most common compliance activities. These are not specific to OpenStack, however we provide references to relevant sections in this book as useful context.

Information Security Management System (ISMS)

An Information Security Management System (ISMS) is a comprehensive set of policies and processes that an organization creates and maintains to manage risk to information assets. The most common ISMS for cloud deployments is ISO/IEC 27001/2, which creates a solid foundation of security controls and practices for achieving more stringent compliance certifications.

Risk Assessment

A Risk Assessment framework identifies risks within an organization or service, and specifies ownership of these risks, along with implementation and mitigation strategies. Risks apply to all areas of the service, from technical controls to environmental disaster scenarios and human elements, for example a malicious insider (or rogue employee). Risks can be rated using a variety of mechanisms, for example likelihood vs impact. An OpenStack deployment risk assessment can include control gaps that are described in this book.

Access & Log Reviews

Periodic access and log reviews are required to ensure authentication, authorization, and accountability in a service deployment. Specific guidance for OpenStack on these topics are discussed in-depth in the logging section.

Backup and Disaster Recovery

Disaster Recovery (DR) and Business Continuity Planning (BCP) plans are common requirements for ISMS and compliance activities. These plans must be periodically tested as well as documented. In OpenStack key areas are found in the management security domain, and anywhere that single points of failure (SPOFs) can be identified. See the section on secure backup and recovery for additional details.

Security Training

Annual, role-specific, security training is a mandatory requirement for almost all compliance certifications and attestations. To optimise the effectiveness of security training, a common method is to provide role specific training, for example to developers, operational personnel, and non-technical employees. Additional cloud security or OpenStack security training based on this hardening guide would be ideal.

Security Reviews

As OpenStack is a popular open source project, much of the codebase and architecture has been scrutinized by individual contributors, organizations and enterprises. This can be advantageous from a security perspective, however the need for security reviews is still a critical consideration for service providers, as deployments vary, and security is not always the primary concern for contributors. A comprehensive security review process may include architectural review, threat modelling, source code analysis and penetration testing. There are many techniques and recommendations for conducting security reviews that can be found publicly posted. A well-tested example is the Microsoft SDL, created as part of the Microsoft Trustworthy Computing Initiative.

Vulnerability Management

Security updates are critical to any IaaS deployment, whether private or public. Vulnerable systems expand attack surfaces, and are obvious targets for attackers. Common scanning technologies and vulnerability notification services can help mitigate this threat. It is important that scans are authenticated and that mitigation strategies extend beyond simple perimeter hardening. Multi-tenant architectures such as OpenStack are particularly prone to hypervisor vulnerabilities, making this a critical part of the system for vulnerability management. See the section on instance isolation for additional details.

Data Classification

Data Classification defines a method for classifying and handling information, often to protect customer information from accidental or deliberate theft, loss, or inappropriate disclosure. Most commonly this involves classifying information as sensitive or non-sensitive, or as personally identifiable information (PII). Depending on the context of the deployment various other classifying criteria may be used (government, health-care etc). The underlying principle is that data classifications are clearly defined and in-use. The most common protective mechanisms include industry standard encryption technologies. See the data security section for additional details.

Exception Process

An exception process is an important component of an ISMS. When certain actions are not compliant with security policies that an organization has defined, they must be logged. Appropriate justification, description and mitigation details need to be included, and signed off by appropriate authorities. OpenStack default configurations may vary in meeting various compliance criteria, areas that fail to meet compliance requirements should be logged, with potential fixes considered for contribution to the community.

50. Certification & Compliance Statements

Compliance and security are not exclusive, and must be addressed together. OpenStack deployments are unlikely to satisfy compliance requirements without security hardening. The listing below provides an OpenStack architect foundational knowledge and guidance to achieve compliance against commercial and government certifications and standards.

Commercial Standards

For commercial deployments of OpenStack, it is recommended that SOC 1/2 combined with ISO 2700 1/2 be considered as a starting point for OpenStack certification activities. The required security activities mandated by these certifications facilitate a foundation of security best practices and common control criteria that can assist in achieving more stringent compliance activities, including government attestations and certifications.

After completing these initial certifications, the remaining certifications are more deployment specific. For example, clouds processing credit card transactions will need PCI-DSS, clouds storing health care information require HIPAA, and clouds within the federal government may require FedRAMP/FISMA, and ITAR, certifications.

SOC 1 (SSAE 16) / ISAE 3402

Service Organization Controls (SOC) criteria are defined by the American Institute of Certified Public Accountants (AICPA). SOC controls assess relevant financial statements and assertions of a service provider, such as compliance with the Sarbanes-Oxley Act. SOC 1 is a replacement for Statement on Auditing Standards No. 70 (SAS 70) Type II report. These controls commonly include physical data centers in scope.

There are two types of SOC 1 reports:

- Type 1 – report on the fairness of the presentation of management's description of the service organization's system and the suitability of the design of the controls to achieve the related control objectives included in the description as of a specified date.

- Type 2 – report on the fairness of the presentation of management's description of the service organization's system and the suitability of the design and operating effectiveness of the controls to achieve the related control objectives included in the description throughout a specified period

For more details see the AICPA Report on Controls at a Service Organization Relevant to User Entities' Internal Control over Financial Reporting.

SOC 2

Service Organization Controls (SOC) 2 is a self attestation of controls that affect the security, availability, and processing integrity of the systems a service organization uses to process users' data and the confidentiality and privacy of information processed by these system. Examples of users are those responsible for governance of the service organization; customers of the service organization; regulators; business partners; suppliers and others who have an understanding of the service organization and its controls.

There are two types of SOC 2 reports:

- Type 1 – report on the fairness of the presentation of management's description of the service organization's system and the suitability of the design of the controls to achieve the related control objectives included in the description as of a specified date.

- Type 2 – report on the fairness of the presentation of management's description of the service organization's system and the suitability of the design and operating effectiveness of the controls to achieve the related control objectives included in the description throughout a specified period.

For more details see the AICPA Report on Controls at a Service Organization Relevant to Security, Availability, Processing Integrity, Confidentiality or Privacy.

SOC 3

Service Organization Controls (SOC) 3 is a trust services report for service organizations. These reports are designed to meet the needs of users who want assurance on the controls at a service organization related to security, availability, processing integrity, confidentiality, or privacy but do not have the need for or the knowledge necessary to make effective use of a SOC 2 Report. These reports are prepared using the AICPA/Canadian Institute of Chartered Accountants (CICA) Trust Services Principles, Criteria, and Illustrations for Security, Availability, Processing Integrity, Confidentiality, and Privacy. Because they are general use reports, SOC 3 Reports can be freely distributed or posted on a website as a seal.

For more details see the AICPA Trust Services Report for Service Organizations.

ISO 27001/2

The ISO/IEC 27001/2 standards replace BS7799-2, and are specifications for an Information Security Management System (ISMS). An ISMS is a comprehensive set of policies and processes that an organization creates and maintains to manage risk to information assets. These risks are based upon the confidentiality, integrity, and availability (CIA) of user information. The CIA security triad has been used as a foundation for much of the chapters in this book.

For more details see ISO 27001.

HIPAA / HITECH

The Health Insurance Portability and Accountability Act (HIPAA) is a United States congressional act that governs the collection, storage, use and destruction of patient health records. The act states that Protected Health Information (PHI) must be rendered "unusable, unreadable, or indecipherable" to unauthorized persons and that encryption for data 'at-rest' and 'inflight' should be addressed.

HIPAA is not a certification, rather a guide for protecting healthcare data. Similar to the PCI-DSS, the most important issues with both PCI and HIPPA is that a breach of credit card information, and health data, do not occur. In the instance of a breach the cloud provider will be scrutinized for

compliance with PCI and HIPPA controls. If proven compliant, the provider can be expected to immediately implement remedial controls, breach notification responsibilities, and significant expenditure on additional compliance activities. If not compliant, the cloud provider can expect on-site audit teams, fines, potential loss of merchant ID (PCI), and massive reputational impact.

Users or organizations that possess PHI must support HIPAA requirements and are HIPAA covered entities. If an entity intends to use a service, or in this case, an OpenStack cloud that might use, store or have access to that PHI, then a Business Associate Agreement must be signed. The BAA is a contract between the HIPAA covered entity and the OpenStack service provider that requires the provider to handle that PHI in accordance with HIPAA requirements. If the service provider does not handle the PHI (e.g. with security controls and hardening) then they are subject to HIPAA fines and penalties.

OpenStack architects interpret and respond to HIPAA statements, with data encryption remaining a core practice. Currently this would require any protected health information contained within an OpenStack deployment to be encrypted with industry standard encryption algorithms. Potential future OpenStack projects such as Swift object encryption will facilitate HIPAA guidelines for compliance with the act.

For more details see the Health Insurance Portability And Accountability Act.

PCI-DSS

The Payment Card Industry Data Security Standard (PCI DSS) is defined by the Payment Card Industry Standards Council, and created to increase controls around card holder data to reduce credit card fraud. Annual compliance validation is assessed by an external Qualified Security Assessor (QSA) who creates a Report on Compliance (ROC), or by a Self-Assessment Questionnaire (SAQ) dependent on volume of card-holder transactions.

OpenStack deployments which stores, processes, or transmits payment card details are in scope for the PCI-DSS. All OpenStack components that are not properly segmented from systems or networks that handle payment data fall under the guidelines of the PCI-DSS. Segmentation in the context of PCI-DSS does not support multi-tenancy, but rather physical separation (host/network).

For more details see PCI security standards.

Government Standards

FedRAMP

"The Federal Risk and Authorization Management Program (FedRAMP) is a government-wide program that provides a standardized approach to security assessment, authorization, and continuous monitoring for cloud products and services". NIST 800-53 is the basis for both FISMA and FedRAMP which mandates security controls specifically selected to provide protection in cloud environments. FedRAMP can be extremely intensive from specificity around security controls, and the volume of documentation required to meet government standards.

For more details see http://www.gsa.gov/portal/category/102371.

ITAR

The International Traffic in Arms Regulations (ITAR) is a set of United States government regulations that control the export and import of defense-related articles and services on the United States Munitions List (USML) and related technical data. ITAR is often approached by cloud providers as an "operational alignment" rather than a formal certification. This typically involves implementing a segregated cloud environment following practices based on the NIST 800-53 framework, as per FISMA requirements, complemented with additional controls restricting access to "U.S. Persons" only and background screening.

For more details see http://pmddtc.state.gov/regulations_laws/itar_official.html.

FISMA

The Federal Information Security Management Act requires that government agencies create a comprehensive plan to implement numerous government security standards, and was enacted within the E-Government Act of 2002. FISMA outlines a process, which utilizing multiple NIST publications, prepares an information system to store and process government data.

This process is broken apart into three primary categories:

- **System Categorization**The information system will receive a security category as defined in Federal Information Processing Standards

Publication 199 (FIPS 199). These categories reflect the potential impact of system compromise.

- **Control Selection**Based upon system security category as defined in FIPS 199, an organization utilizes FIPS 200 to identify specific security control requirements for the information system. For example, if a system is categorized as “moderate” a requirement may be introduced to mandate “secure passwords.”
- **Control Tailoring**Once system security controls are identified, an OpenStack architect will utilize NIST 800-53 to extract tailored control selection, e.g. specification of what constitutes a “secure password.”

51. Privacy

Privacy is an increasingly important element of a compliance program. Businesses are being held to a higher standard by their customers, who have increased interest in understanding how their data is treated from a privacy perspective.

An OpenStack deployment will likely need to demonstrate compliance with an organization's Privacy Policy, with the U.S. – E.U. Safe Harbor framework, the ISO/IEC 29100:2011 privacy framework or with other privacy-specific guidelines. In the U.S. the AICPA has defined 10 privacy areas of focus, OpenStack deployments within a commercial environment may desire to attest to some or all of these principles.

To aid OpenStack architects in the protection of personal data, it is recommended that OpenStack architects review the NIST publication 800-122, titled "*Guide to Protecting the Confidentiality of Personally Identifiable Information (PII)*." This guide steps through the process of protecting:

> "*any information about an individual maintained by an agency, including (1) any information that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and (2) any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information*"

Comprehensive privacy management requires significant preparation, thought and investment. Additional complications are introduced when building global OpenStack clouds, for example navigating the differences between U.S. and more restrictive E.U. privacy laws. In addition, extra care needs to be taken when dealing with sensitive PII that may include information such as credit card numbers or medical records. This sensitive data is not only subject to privacy laws but also regulatory and governmental regulations. By deferring to established best practices, including those published by governments, a holistic privacy management policy may be created and practiced for OpenStack deployments.

52. Case Studies

In this case study we discuss how Alice and Bob would address common compliance requirements. The preceding chapter refers to a wide variety of compliance certifications and standards. Alice will address compliance in a private cloud, while Bob will be focused on compliance for a public cloud.

Alice's Private Cloud

Alice is building an OpenStack private cloud for the United States government, specifically to provide elastic compute environments for signal processing. Alice has researched government compliance requirements, and has identified that her private cloud will be required to certify against FISMA and follow the FedRAMP accreditation process, which is required for all federal agencies, departments and contractors to become a Certified Cloud Provider (CCP). In this particular scenario for signal processing, the FISMA controls required will most likely be FISMA High, which indicates possible "severe or catastrophic adverse effects" should the information system become compromised. In addition to FISMA Moderate controls Alice must ensure her private cloud is FedRAMP certified, as this is a requirement for all agencies that currently utilize, or host federal information within a cloud environment.

To meet these strict government regulations Alice undertakes a number of activities. Scoping of requirements is particularly important due to the volume of controls that must be implemented, which will be defined in NIST Publication 800-53.

All technology within her private cloud must be FIPS certified technology, as mandated within NIST 800-53 and FedRAMP. As the U.S. Department of Defense is involved, Security Technical Implementation Guides (STIGs) will come into play, which are the configuration standards for DOD IA and IA-enabled devices / systems. Alice notices a number of complications here as there is no STIG for OpenStack, so she must address several underlying requirements for each OpenStack service, e.g. the networking SRG and Application SRG will both be applicable (list of SRGs). Other critical controls include ensuring that all identities in the cloud use PKI, that SELinux is enabled, that encryption exists for all wire-level communications, and that continuous monitoring is in place and clearly documented. Alice is not

concerned with object encryption, as this will be the tenants responsibility rather than the provider.

If Alice has adequately scoped and executed these compliance activities, she may begin the process to become FedRAMP compliant by hiring an approved third-party auditor. Typically this process takes up to 6 months, after which she will receive an Authority to Operate and can offer OpenStack cloud services to the government.

Bob's Public Cloud

Bob is tasked with compliance for a new OpenStack public cloud deployment, that is focused on providing cloud services to both small developers and startups, as well as large enterprises. Bob recognizes that individual developers are not necessarily concerned with compliance certifications, but to larger enterprises certifications are critical. Specifically Bob desires to achieve SOC 1, SOC 2 Security, as well as ISO 27001/2 as quickly as possible. Bob references the Cloud Security Alliance Cloud Control Matrix (CCM) to assist in identifying common controls across these three certifications (such as periodic access reviews, auditable logging and monitoring services, risk assessment activities, security reviews, etc). Bob then engages an experienced audit team to conduct a gap analysis on the public cloud deployment, reviews the results and fills any gaps identified. Bob works with other team members to ensure that these security controls and activities are regularly conducted for a typical audit period (~6-12 months).

At the end of the audit period Bob has arranged for an external audit team to review in-scope security controls at randomly sampled points of time over a 6 month period. The audit team provides Bob with an official report for SOC 1 and SOC 2, and separately for ISO 27001/2. As Bob has been diligent in ensuring security controls are in place for his OpenStack public cloud, there are no additional gaps exposed on the report. Bob can now provide these official reports to his customers under NDA, and advertise that he is SOC 1, SOC 2 and ISO 27001/2 compliant on his website.

Glossary

ACL
: See access control list.

AMQP
: Advanced Message Queue Protocol. An open Internet protocol for reliably sending and receiving messages. It enables building a diverse, coherent messaging ecosystem.

API
: Application programming interface.

BMC
: Baseboard Management Controller. The intelligence in the IPMI architecture, which is a specialized micro-controller that is embedded on the motherboard of a computer and acts as a server. Manages the interface between system management software and platform hardware.

CA
: Certificate Authority or Certification Authority. In cryptography, an entity that issues digital certificates. The digital certificate certifies the ownership of a public key by the named subject of the certificate. This allows others (relying parties) to rely upon signatures or assertions made by the private key that corresponds to the certified public key. In this model of trust relationships, a CA is a trusted third party for both the subject (owner) of the certificate and the party relying upon the certificate. CAs are characteristic of many public key infrastructure (PKI) schemes.

Chef
: An operating system configuration management tool supporting OpenStack deployments.

CMDB
: Configuration Management Database.

DHCP
: Dynamic Host Configuration Protocol. A network protocol that configures devices that are connected to a network so they can communicate on that network by using the Internet Protocol (IP). The protocol is implemented in a client-server model where DHCP clients request configuration data such as, an IP address, a default route, and one or more DNS server addresses from a DHCP server.

Django
: A web framework used extensively in horizon.

DNS
: Domain Name Server. A hierarchical and distributed naming system for computers, services, and resources connected to the Internet or a private network. Associates a human-friendly names to IP addresses.

Puppet
: An operating system configuration management tool supported by OpenStack.

Qpid
: Message queue software supported by OpenStack, an alternative to RabbitMQ.

RabbitMQ
: The default message queue software used by OpenStack.

ZeroMQ
: Message queue software supported by OpenStack, an alternative to RabbitMQ. Also spelled 0MQ.