Introduction

Cloud computing’s history is one punctuated by a focus on security, typically the question being whether cloud environments can be trusted as the de-facto choice in an enterprise’s infrastructure. This is especially important in a hybrid or multi-cloud architecture, where one or more public clouds are used alongside a private or on-premises set-up.

The rise of the cloud as a necessity for enterprises looking to effectively compete places an emphasis on clouds meeting strict security and compliance requirements. More often than not, security requirements are only addressed as an afterthought during the implementation and rollout of OpenStack. This is where surprises happen as the implementation design usually does not refer to the more stringent security requirements of the CSO. Although OpenStack has been known to be the basis for a more secure, robust offering, that is better suited to highly regulated industries such as healthcare, government, banking and telecommunications, it requires careful consideration in the design phase of the project.

Even within OpenStack, there are some distributions that are more secure than others. For example, Charmed OpenStack goes beyond OpenStack’s base level of built-in security and applies OpenStack security recommendations which are documented in the OpenStack Security Guide to the extent that is applicable out of the box.

Ubuntu when used as the base operating system for OpenStack comes with same default security configuration turned on already. Interested parties can inspect the concrete measures which are taken in each release under current support on the Ubuntu Wiki. For more comprehensive information around all things Canonical offers around security, please see our website at https://www.ubuntu.com/security.
Contents

In this white paper we highlight some of the advanced security features that Canonical brings to OpenStack clouds, ensuring that they meet the highest levels of security for our customers. We focus on use cases which go beyond the simple deployment of a secure OpenStack and highlight how we solve enterprise requirements in the field every day. The most common such security requirements are encryption at rest, and the question of certificate lifecycle management. Both are not solved by OpenStack itself, but require integration with state of the art 3rd party open source applications.

OpenStack – Encryption at Rest
The Problem
The Solution
Conclusion

OpenStack – Certificate Lifecycle management
The Problem
The Solution
Conclusion
OpenStack – Encryption at Rest

The Problem

What happens to your data when someone steals a disk from your data centre? Or a server? Or even a rack of servers? Can the perpetrator of this crime actually access anything sensitive? And how big a fine is my company going to get if this actually happens?

These are all questions that companies find themselves asking, especially with the introduction of new data protection regulation such as GDPR.

In the context of an Ubuntu OpenStack Cloud, there are multiple places where users of the cloud may end up placing potentially sensitive and personally identifying data:

- **Ephemeral instance storage** – data placed in a Nova managed virtual disk associated with an instance that resides directly on a hypervisor server.

- **Persistent block storage** – data placed in a Cinder managed block device provided by a storage solution such as Ceph, attached to an instance in the cloud.

- **Object storage** – data placed in Swift (or a Ceph RADOS Gateway) via its REST API.

All of these functions of the cloud end up placing data directly on disk; if that data is not encrypted, it’s vulnerable to all of the physical loss scenarios detailed above. In principle it would be possible to encrypt the data in the client application, however, client-side encryption requires refactoring of workloads and introduces inhibiting complexity to the application stack.
The Solution

So how do we mitigate the risk of one of these scenarios resulting in a compromise of data security in an Ubuntu OpenStack Cloud?

Encryption of data in Linux is actually pretty easy – dm-crypt + LUKS (the on-disk format for managing encryption keys) provide highly optimised kernel level support for the encryption of block devices. However, that’s just part of the solution – security of the keys used for encryption of block devices is also a critical part of any encryption-at-rest solution, requiring a key management solution (KMS) with a good level of role based access control (RBAC) and strong unseal semantics to mitigate against all physical loss scenarios. In addition the block device encryption keys need to be available during day-to-day operations to support automatic unlocking of encrypted block devices without being stored on local disks – for example during server reboots to support security patching operations.

For Ubuntu OpenStack, we chose to use Hashicorp Vault for this function. Vault provides strong security semantics for master encryption key unsealing on startup in the form of multiple unlock keys with configurable quorum levels. As part of an OpenStack Cloud deployment, we ensure that the unseal keys for Vault are never stored within the systems hosting the cloud; these are always held outside of the deployment by multiple third party owners/systems. This ensures that in the event of a server hosting Vault being powered off, Vault won’t service requests until its been unsealed, which is an operation external to the boot up of the server and Vault service.

Vault also provides a highly granular level of access control, allowing us to limit access to stored encryption keys to the system that actually hosts the physical block device. For example, servers that host OpenStack instances on encrypted block devices cannot access the keys used to encrypt block devices used to support Ceph on a different server.

Canonical wrote vaultlocker to integrate this Vault based key management approach into the three storage touchpoints within an OpenStack Cloud.

In terms of RBAC, each server that needs to provide block device encryption is provided with a Vault AppRole and Secret which is specific to the server; this AppRole is CIDR limited to the server’s IP address, and only allows access to a server hostname specific path in the secrets backend – which is where the encryption keys for each block device are stored.
In a typical Ubuntu OpenStack Cloud deployment the setup of the AppRole and associated secret is orchestrated using Juju and the OpenStack Charms using a Vault wrapped response for the secret so that only the consuming service has direct access to the secret i.e. the Vault charm and Juju never actually have visibility of the secret. Vaultlocker also ensures that encryption keys are never stored on local disk; during encryption the key is generated and held in memory and then stored directly in Vault, and during subsequent unlock operations the key is read from Vault into memory and then used to open the LUKS based block device.

For each touchpoint (Nova, Cinder+Ceph and Swift) vaultlocker is used to encrypt (on setup) and unlock (on boot) an underlying block device, which can then be mounted and used as a filesystem managed via /etc/fstab (as for Nova and Swift use-cases) or just presented to the system for other tools to use – such as Ceph’s ceph-volume tooling for bootstrapping Ceph OSD’s into a Ceph cluster.
Conclusion

By using Vault and vaultlocker, Charmed OpenStack Clouds provide a single, consistent approach to block device encryption and key management for all security touchpoints, ensuring the security of data-at-rest in the event of pretty much any type of physical device loss in your data centre. This feature is available as part of BootStack, Canonical’s managed service offering for Ubuntu OpenStack, or as part of the general feature set of any Charmed OpenStack Cloud – check out the OpenStack Charm Deployment Guide sections on using Vault and encryption-at-rest to get started or contact us to enquire about Canonical’s managed OpenStack service.
In this day and age there is simply no excuse for sending any remotely private data over the network in clear text and this certainly applies to an OpenStack deployment. When using the command line interface you may be completely unaware that your credentials are flying over the wire in clear text. OpenStack does not mandate the use of secured endpoints when communicating with it, in fact, it does not even warn you.

In the early stages of OpenStack adoption it may seem tempting to leave the control plane unsecured. Perhaps the workloads initially earmarked for OpenStack do not contain sensitive data or the deployment starts out as an R&D exercise. However, projects and requirements evolve and what might seem fine today will almost certainly not be tomorrow. Securing your deployment is always the right approach, exposing credentials in clear text over the wire is simply very bad practice.
The Problem

Perhaps the solution is to create your own internal certificate authority (CA) and use that to generate and sign certificates? Any system administrator worth their salt can do this and then use the generated keys and certificates to encrypt the OpenStack endpoints.

However, this approach has its own set of problems:

• A client, either user or other OpenStack service, will not trust a self-signed certificate by default. To establish the trust they need to install the corresponding CA file. That means it needs distributing to every users machine and installing on almost every server supporting the OpenStack cloud.

Maybe your company already has an internal CA that can be used and all the users already have the corresponding CA certificate installed. That deals with the trust issue which just leaves you managing the certificates inside the Openstack cloud.

However, that is no small task:

• An OpenStack deployment typically has at least ten services. Each of these services exposes endpoints, each endpoint needs a certificate. In addition there may well be three or more machines running the service for resilience. That means there could be ten certificates over thirty machines or containers that need managing. Common services that need securing are:
  - Glance, for managing VM images
  - Keystone, identity management
  - Nova, compute service
  - Neutron, network service
  - Placement, place VMs on compute resource
  - Swift, object storage
  - Cinder, block storage
  - Ceilometer, data collection service
  - Gnocchi, time series database
  - Aodh, alarming service

• Certificates expire and when they do a new one needs to be generated and installed. Manually managing the certificate lifecycle wastes time and the impact of failing to do it will, in all likelihood, lead to a service outage as it recently did for the telecom company O2.

• Managing the certificates is not limited to their expiry date. A change in the endpoint address, due to an IP address or domain name change for example) will invalidate the certificate and require them to be reissued. Once again, this is a manual overhead wasting an administrators time.
The Solution

Ubuntu OpenStack Cloud ensures that the cloud is secured but takes away the time and pain of having to administer the certificates manually. This is achieved using charms. Charms are sets of scripts for deploying and operating software. With event handling built in, they can declare interfaces that fit charms for other services, so that relationships can be formed. These relationships are a sophisticated mechanism for passing data between applications. By using a relationship an application can request a certificate to secure its endpoint. Now all that is needed is an application which can respond to these requests.

For Ubuntu OpenStack, we chose to use Hashicorp Vault. Vault provides a feature rich Public Key Interface (PKI) allowing certificates to be generated on demand. The Vault charm, authored by Canonical, takes the certificate requests from the OpenStack services, translates them to API calls to Vault's PKI interface and then passes them back to the requesting application via the relation. The charm for each OpenStack service knows how to install the certificate and chain received from Vault and how to update the service catalogue to ensure all future requests are secured.

The problem of clients trusting the certificates presented by OpenStack services is also taken care of. Vault supports creating an intermediate CA. An intermediate CA is a CA which has been signed by another trusted CA and can be used to sign certificates itself. This means a company’s existing CA (Active Directory for example) can be used to sign an intermediate CA which Vault can use to sign the application certificate requests. When a user interacts with OpenStack they are presented with the applications certificate and a chain file which proves to the client that if they trust the companies CA they should trust the application certificate.

An intermediate certificate does not have to be used. Vault can act as a root CA too. In this case all the OpenStack applications will trust each other because they will have installed the root certificate provided by their relationship with Vault but users will need to collect the certificate from Vault and install it locally.
Using Vault in the OpenStack deployment makes initial installation easier but it also helps with on-going maintenance. Here are a few typical scenarios.

**Scaling out an existing service:** Perhaps a new Glance server is needed to handle load. Use Juju to add another unit to the glance application and as the new unit is provisioned it will request a new certificate from Vault and will come up with a the new certificate installed and ready.

**Adding Fault tolerance:** There is currently only one unit of a service but to enable fault tolerance, more units are to be added and a Virtual IP (VIP) used as the service endpoint. Adding the VIP configuration will trigger existing units and new units to request and install a new certificate which includes a Subject Alternate Name entry for the VIP.

**OpenStack service certificates are expiring and need to be re-issued:** The Vault charm provides a ‘reissue-certificates’ action. Running this action will cause Vault to re-issue certificates and update its relations. The OpenStack charms will notice the new certificates and install them.

**Changing Certificate Authority:** Perhaps a PoC cloud used the Vault root CA for convenience but now it is to be rolled out to a wider audience and the CA needs to be changed to an intermediate signed by the companies central CA. The Vault charm provides actions for this. Using the vault charm actions the CA can be replaced and new certificates issued.
Conclusion

Ubuntu OpenStack Cloud integrates seamlessly with Vault to provide an OpenStack with secured endpoints that require minimal input from administrators. The ongoing management of that cloud is further automated allowing certificates to be simply reissued or new ones generated for new services or machines. Businesses implement essential TLS but without the administrative overhead of managing large numbers of certificates.

Credentials should never be passed over the network in clear text. Users should be able to take it for granted that the service they are talking is not being spoofed. TLS solves these issues and Ubuntu OpenStack Cloud allows it to be implemented with ease. So as Chrome and Firefox start warning users to every http connection let us respond to this call and say from this day forth there will be no OpenStack deployments with http endpoints on our watch.

Note: The OpenStack cli does provide a "-insecure" option which can be used to ignore trust chains. However, using an option like this is akin to telling users to ignore invalid certificate errors in browsers, it’s a very bad habit and one worth avoiding.