ETSI GS NFV-SEC 002 V1.1.1 (2015-08)

Network Functions Virtualisation (NFV);
NFV Security;
Cataloguing security features in management software

Reference
DGS/NFV-SEC002

Keywords
NFV, open source, security

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document gives a survey of the security features in the open source management software relevant to NFV, in particular OpenStack™ [i.1] as the first case study. It addresses the OpenStack modules that provide security services (such as authentication, authorization, confidentiality protection, integrity protection, and logging) together with the full graphs of their respective dependencies down to the ones that implement cryptographic protocols and algorithms. It also identifies a set of recommendations on the use of and enhancements to OpenStack as pertinent to NFV.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

Not applicable.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the reference document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] OpenStack.

NOTE: http://www.openstack.org/.

[i.2] United States Computer Emergency Readiness Team.

NOTE: http://www.us-cert.gov/.

[i.3] ETSI GS NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[i.4] ETSI GS NFV-SEC 001: "Network Functions Virtualisation (NFV); NFV Security; Problem Statement".

[i.5] ETSI GS NFV 004: "Network Functions Virtualisation (NFV); Virtualisation Requirements".

[i.6] ETSI GS NFV-MAN 001: "Network Functions Virtualisation (NFV); Management and Orchestration", (work in progress).

[i.7] Memcached.

NOTE: http://memcached.org/.

[i.8] OpenID Connect.

NOTE: http://openid.net/connect/.

[i.9] IETF Application Bridging for Federated Access Beyond web (ABFAB) Working Group.

NOTE: http://tools.ietf.org/wg/abfab/charters.

[i.10] IETF RFC 5905 (June 2010): "Network Time Protocol Version 4: Protocol and Algorithms Specification".

NOTE: https://tools.ietf.org/html/rfc5905.

[i.11] IEEE 1588-2008 (July 2008): "IEEE Standard for a Precision Clock Synchronization for Networked Measurement and Control Systems".

[i.12] The OpenStack Security Guide.

NOTE: http://docs.openstack.org/sec/.

[i.13] Trusted Computing Group: Storage Work Group Storage Security Subsystem Class: Opal.

NOTE: http://www.trustedcomputinggroup.org/resources/storage_work_group_storage_security_subsystem_class_opal.

[i.14] IETF RFC 3164: "The BSD syslog Protocol".

[i.15] IETF RFC 5424: "The Syslog Protocol".

[i.16] IETF RFC 5280: "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile".

[i.17] FIPS PUB 186-4: "Digital signature Standard".

[i.18] DMTF: "Cloud Auditing Data Federation (CADF)".

NOTE: Available at: http://www.dmtf.org/standards/cadf.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in ETSI GS NFV 003 [i.3] apply.

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in [i.3] and the following apply:

AMQP	Advanced Message Queuing Protocol
AH	Authentication Header
API	Application Program Interface
ARP	Address Resolution Protocol
CADF	Cloud Auditing Data Federation
CMS	Cryptographic Message Syntax
DHCP	Dynamic Host Configuration Protocol
DMTF	Distributed Management Task Force
ESP	Encapsulating Security Payload
GRE	Generic Route Encapsulation
HMAC	Hashed Message Authentication Code
HTTP	HyperText Transfer Protocol
IKE	Internet Key Exchange
IP	Internet Protocol
JSON	JavaScript Object Notation
KVS	Key Value Stores

LDAP	Lightweight Directory Access Protocol
LUKS	Linux Unified Key Setup
MAC	Media Access Control
MAC/IP	Media Access Control / Internet Protocol
NSS	Network Security Services
NTP	Network Time Protocol
PEM	Privacy Enhanced Mail
PKI	Public Key Infrastructure
PTP	Precision Time Protocol
RPC	Remote Procedure Call
SAML	Security Assertion Mark-up Language
SASL	Simple Authentication and Security Layer
SED	Self Encrypting Drive
SQL	Structured Query Language
SR-IOV	Single Root Input Output Virtualization
SSH	Secure SHell
SSL	Secure Socket Layer
TCP	Transfer Control Protocol
URI	Uniform Resource Identifier
UUID	Universally Unique IDentifier
VLAN	Virtual LAN
VM	Virtual Machine
VNC	Virtual Network Computing
VPN	Virtual Private Network
VXLAN	Virtual eXtensible Local Area Network
WSGI	Web Server Gateway Interface

4 Introduction

Building on open source software can help advance certain goals of NFV, such as accelerated time-to-market and improved interoperability. To do so effectively calls for having a knowledge base of the security features and cryptographic algorithms supported in each relevant code base. In particular, NFV applications are subject to privacy and security regulations. The knowledge base helps shed light on how to best apply the pertinent software and on enhancements necessary to meet the NFV security needs. It is also useful for other reasons. Chief among them are:

- export control of cryptographic software;
- compliance with procurement processes;
- follow-up on alerts from US-CERT [i.2] and other similar organizations; and
- determination of the relevant elements for security analytics.

Such a knowledge base is of particular importance in the area of management and orchestration, which plays a critical role in NFV security.

The present document addresses OpenStack, a widely adopted cloud operating system, as the first case study. It aims to cover all applicable aspects of information and network security, including:

- Identity and access management
- Communication security
- Stored data security
- Firewalling, zoning, and topology hiding
- Availability

- Logging and monitoring
- Compute isolation

NOTE: OpenStack™ is a set of open source tools for building and managing cloud-computing software platforms for public and privvate clouds.

It consists of a group of interrelated projects that control pools of processing, storage, and networking resources throughout a data center e.g. Neutron, Nova, Keystone, Barbican, Swift, Glance, Trove, Cinder, etc.

The present document describes the OpenStack modules that provide security services (e.g. authentication, authorization, confidentiality protection and integrity protection) together with their respective dependencies on cryptographic protocols and algorithms. It also makes a set of recommendations on the use of and enhancements to OpenStack as pertinent to NFV. The case study takes into account the issues identified in ETSI GS NFV-SEC 001 [i.4] and the related requirements specified in ETSI GS NFV 004 [i.5] and ETSI GS NFV-MAN 001 [i.6].

5 Identity and access management

5.1 General

Keystone is the component in OpenStack that provides centralized authentication and authorization. It is used by all OpenStack components for API access control. Hence, at a high level, a user is authenticated by Keystone first before gaining access to any other service (Keystone may employ an external authentication system). Upon successful authentication, the user is given a temporary token. From this point on, to get a service, the user includes the token in the service request. The user can receive the service if and only if the token is validated and if the user has the proper roles.

Keystone is organized as a set of internal services, including the identity service, token service, and catalog service. The identity service handles user authentication and user-data validation. The following constructs are basic to the service:

- User, which may be a person or a process using an OpenStack service.
- Project (or tenant), which owns a set of OpenStack resources. A project shall be assigned a domain.
- Group, which is a set of users. A group shall be assigned a domain. A user may be assigned one or multiple groups.
- Domain, which is a set of users, groups, and projects.
- Role, which specifies a set of rights and privileges. Roles can be granted at either the domain or project level. A group may be assigned one or multiple roles on a domain. A user may be assigned one or multiple roles on a project or domain. An example role is *admin*. A user shall have a role assigned to have access to a resource.

The identity service supports basic management of user data (e.g. create, read, update and delete). It also has the flexibility to use a pluggable authentication or authorization module through a backend. Common backends include Lightweight Directory Access Protocol (LDAP) servers, SQL databases and Key Value Stores (KVS). Keystone uses an SQL backend by default.

The identity service is accessible through a REST API. The corresponding API endpoint is, in fact, the entry point to all services. An endpoint is a network-accessible address in the form of a Uniform Resource Identifier (URI). The identity service may support a separate endpoint for administrative purposes. It goes without saying that the transport of all API access transactions needs to be protected. In general, access control is based on configurable policy stored in a JSON file. Other components in OpenStack can further customize the policy according to their respective service contexts. Keystone supports an SQL policy backend.

The token service deals with token management and validation. It relies on a database to store tokens and the associated data, including the token revocation list (or token revocation events) and per-token information (e.g. lifespan and scope). The scope of a token is determined by a combination of projects (or domains) and roles associated with the user. An unscoped token does not include a specified role. Such a token may be issued during the initial authentication of the user, who can then use the token to discover accessible projects and then exchange it for a scoped token.

As the basis for service access, tokens shall be protected from forgery, and from unauthorized access and alteration in transit and at rest. The token service also provides protection in this regard. Several types of tokens are supported, including Public Key Infrastructure (PKI) and Universally Unique IDentifier (UUID). The token type in use as well as other specifics (e.g. token lifespan) is configurable. The default token type is UUID. Depending on the token type, it may be useful to cache tokens to enhance performance. OpenStack services can be configured to this end. When used, token caches need to be protected and expiration times need to be set appropriately. Custom token types are also possible through external modules.

The catalogue service manages a registry of all OpenStack services. It allows a user to discover the entitled services and the corresponding endpoints. Services can be organized in terms of regions, while endpoints classified as public, internal or administrative. It is also possible to have tenant-specific endpoints. Keystone supports an SQL catalogue backend.

5.2 PKI tokens

5.2.0 General

A PKI token is a Cryptographic Message Syntax (CMS) string, essentially data that are digitally signed and base64 encoded. The specifics of the data signed are context-dependent. They may include information on, for example, the user, tenant, role, trust, timestamp and entitled services. One characteristic of such a token is its long length. It is possible that a PKI token is too long to fit into either a header or URI. To reduce the token size, Keystone supports compression through `zlib`. Still the size of a compressed PKI token is much larger than that of a UUID token.

PKI tokens are verifiable by any API endpoints as long as they have access to Keystone's signing certificate, the information for verifying the signing certificate (i.e. the certificate chain and certificate revocation list), and the token revocation list (or revocation event records). Keystone provides an API for retrieval of relevant signing certificates. Decentralized token validation reduces the chance of Keystone becoming a bottleneck. For this reason, PKI had been the default token type since the Grizzly release. Nevertheless, it has been changed back to UUID in the Juno release based on deployment experience. The concerns are largely due to the large size of PKI tokens.

5.2.1 PKI set-up

Keystone provides the utility for generating the signing key, the corresponding certificate and the certificate chain that are required for token generation and management. The required material may be externally generated and imported. Either way, it is stored in separate files in the Privacy Enhanced Mail (PEM) format in the directories as specified in the Keystone configuration file (i.e. `keystone.conf`). Keystone does not support encryption of private key files but relies on the access control mechanisms of the underlying operating system to protect such files.

The Keystone utility for generating signing keys and certificates is the command `keystone-manage pki_setup`, which is based on OpenSSL. The key size and certificate lifespan are configurable through `keystone.conf`. The signature algorithm in use is RSA-SHA256. RSA is hardcoded in `keystone/common/openssl.py` and SHA256 in `keystoneclient/common/cms.py`.

5.2.2 Token generation

Table 1

Cryptographic module used	`openssl cms -sign -signer /etc/keystone/ssl/certs/signing_cert.pem -inkey /etc/keystone/ssl/private/signing_key.pem -outform PEM -nosmimecap -nodetach -nocerts –noattr –md –sha256`
Signature algorithm	default RSA-SHA1 (with `key-size = 2048`)
Configurability	configurable through the signing certificate and key as part of PKI setup
Invoking module	`keystone/common/cms.py/cms_sign_text()` or `keystoneclient/common/cms.py/cms_sign_text()`

The token lifespan is configurable through `keystone.conf`. The default is one hour.

5.2.3 Token verification

A PKI token is valid if satisfying at least the following criteria:

- It has not expired.
- It had not been revoked.
- The signing certificate and token signature are valid. (The certificate chain is valid and no certificates have been revoked.)

Table 2

Cryptographic module used	`openssl cms -verify -certifle /etc/keystone/ssl/certs/signing_cert.pem –CAfile /etc/keystone/ssl/certs/ca.pem -inform PEM -nosmimecap -nodetach -nocerts -noattr`
Signature algorithm	default RSA-SHA1 (with `key-size = 2048`)
Configurability	configurable through the signing certificate and key as part of PKI setup
Invoking module	`keystone/common/cms.py/cms_verify()` or `keystoneclient/common/cms.py/cms_verify()`

5.2.4 Token indexing

To reference PKI tokens, their hash values are used. The hash algorithm is configurable.

Table 3

Cryptographic module used	`hashlib.py`
Hash algorithm	default MD5 (expected to change to SHA256)
Configurability	The hash algorithm is configurable via `hash_algorithms` in `keystone.conf`.
Invoking module	`keystone/common/cms.py/cms_hash_token()` or `keystoneclient/common/cms.py/cms_hash_token()`

5.3 UUID tokens

UUID tokens are randomly generated strings that are verifiable by Keystone only. The use of such tokens necessitates a persistent storage backend. A UUID token is valid if there is a non-expired matching token in the backend.

Table 4

Cryptographic module used	`uuid.uuid4().hex` (from `uuid.py`)
Random number generation	`/dev/urandom` or a pseudo-random number generator
Configurability	Platform-dependent
Invoking module	`keystone.token.providers.uuid.py`
NOTE 1: `uuid.uuid4()` makes use of the available UUID library from the platform (e.g. `libuuid` on Linux). If both `uuid_generate_rand()` and `uuid_generate_time()`are available, the former is the first choice. `uuid_generate_rand()`builds on `/dev/urandom, while uuid_generate_time()` is based on the local current time, the MAC address (if available), and a pseudo-random number. If no UUID library is available, `uuid.uuid4()` constructs UUIDs based on `/dev/urandom` directly. If `/dev/urandom` is unavailable, it constructs UUIDs based on the pseudo-random number generators implemented in `random.py`, which are not deemed cryptographically secure.	
NOTE 2: `/dev/urandom` is not as strong as `/dev/random.` The latter will return bits only if there is sufficient entropy in the input. This is known as blocking. In contrast, the former is non-blocking.	
NOTE 3: `Pseudo-random number generation should be NIST FIPS-186-4 [i.17] compliant.`	

5.4 Trusts

Keystone further supports delegation through the construct of *trust*, which is not cryptographically based. The construct is particularly relevant to services (e.g. Heat) involving deferred operations. In these cases, tokens alone are insufficient; they are short lived.

A trust is between two parties: the *trustor* and *trustee*. Only the trustor can create a trust, which represents the trustor's consent to let the trustee act on the trustor's behalf. The trustee has exclusive use of the trust. To carry out a delegated task, the trustee uses a somewhat special token that reflects its delegated role. The special token is known as a trust token, which is just a normal token with additional trust-related information. The trustee (after being authenticated) obtains a trust token from Keystone by presenting a trust. For example, a Heat service user gets a token from Keystone based on a trust with Alice for accessing Nova and uses the token to create a new instance of Alice's stack on her behalf in response to an auto-scaling or a reporting event.

A trust has a limited scope. In other words, only a sub-set of the trustor's privileges is delegated to the trustee. The delegated set of privileges is a combination of a project and a list of roles associated with the trustor. In Heat, the default role is `heat_stack_owner`. This is configurable through `heat.conf`. A token derived from a trust will have the same scope as the trust.

Trusts are immutable once created. To update a trust relationship, the trustor deletes the old trust and creates a new one. Upon deletion of the old token, any tokens derived from it are revoked. If the trustor loses any delegated privileges, the trust becomes invalid.

A trust may have a limited lifetime. If the expiration time is not specified, the trust is valid until it is explicitly revoked. The default is infinite lifetime and it is hardcoded.

A trust may be constrained in terms of the number of times that it can be used for getting a token. The default is no cap and it is hardcoded.

A trust may allow "impersonation" so that the trustee appears as the trustor to services validating a trust token. (Impersonation as meant in OpenStack is a misnomer, given that it refers to authorized use of an identity.) Impersonation is enabled by default. This is configurable through `keystone.conf`. Note that a trust token carries a trust object containing all the related information, including the impersonation flag. So an impersonator is known in effect. When impersonation is enabled, logs need to capture the information necessary for auditing and non-repudiation purposes.

Delegation may be recursive so that the original trustee further delegates another trustee, becoming a trustor as a result. This is controlled by the flag `allow_redelegation`, which is disabled by default. In the case of recursive delegation, the trust construct further includes an attribute for specifying the depth of delegation, or the length of the re-delegation chain. A re-delegation depth of the value zero means that further delegation is impossible. The depth of re-delegation is configurable through `keystone.conf`. Recursive delegation is applicable to VM lifecycle management (as implemented in Solum).

By default, the authentication method for deferred operations is set to passwords instead of trusts. This requires storage of passwords in yet another backend, which adds complications. This is configurable through `heat.conf`.

The default backend for trusts is SQL. This is configurable through `keystone.conf`. The database keeps track of, among other things, the expiration time, deletion time and remaining quota of each trust. Trusts, as designed, do not employ any cryptographic measures, and are vulnerable to forgery and tampering. Proper controls needs to be in place to protect the trust information and support auditing and non-repudiation.

Trusts can be queried at the identity API endpoint. The access control policy is configurable through a Keystone policy file (typically `policy.json`). For obvious reasons, only a trustor or trustee may issue queries. The related policy should be set accordingly.

A trust is identified by a UUID, which is generated the same way as for an UUID token.

5.5 Token storage

Keystone supports the following back-ends for token storage in addition to the default SQLite database in some ditstributions:

- SQL database (default).

  The storage is persistent. Expired tokens are not automatically removed from the backend. To purge such tokens, the `keystone-manage token_flush` command can be used. Protection of the tokens in the database is implementation-dependent.

- Key Value Store (KVS).

  The base implementation keeps tokens in memory. It is primarily for testing support.

- memcached [i.7].

  This is a key-value store in support of caching. The storage is in-memory, distributed, and ephemeral. Expired tokens are removed automatically. Cached tokens may be protected through HMAC or HMAC together with encryption. This is configurable through `keystone.conf` or the configuration file of another OpenStack component (e.g. `heat.conf`).

Table 5

Cryptographic module used	`Crypto.hash` and `Crypto.Cipher` in the Python Cryptography Toolkit (`pycrypto`)
Digest algorithm	SHA384
Encryption algorithm	AES (with 128-bit keys)
Key derivation	HMAC-based
Configurability	Limited. • The digest and encryption algorithms are hard-coded. • The HMAC and encryption keys are derived from a configurable secret key (`memcache_secret_key`) in a configuration file, such as `heat.conf`. • The HMAC key size is hard-coded to 128 bits. • The encryption key size is hard-coded to 128 bits.
Invoking module	`keystoneclient.middleware.memcache_crypt.py`
NOTE 1: Secret key protection relies on the access control mechanisms of the underlying platform. NOTE 2: memcached supports Simple Authentication and Security Layer (SASL) as an option but the default is no authentication.	

5.6 Token Transport

Keystone supports secure transport of tokens through SSL/TLS, although SSL/TLS is disabled by default. SSL/TLS connections can be effected between Keystone and its user as well as between Keystone and its backend. The information required for SSL/TLS operations is provisioned in the files as specified in `keystone.conf`.

Table 6

Cryptographic module used	OpenSSL library (the SSL/TLS-specific part)
Cipher suite	Negotiated during the handshake
Configurability	• The list of candidate cipher suites is configurable through openssl.cnf (that is part of the underlying platform) and openssl.conf (that is part of OpenStack). • Client authentication based on certificates is configurable through `keystone.conf`.
Invoking module	`keystoneclient.session.Session()`
NOTE: Keystone may run as part of an HTTP server. In this case, a different SSL/TLS library may be used as dictated by the HTTP server.	

5.7 Identity federation

Through identity federation Keystone can outsource identity management to an external provider known as an identity provider. The assumption here is that the identity provider is trusted. It is, thus, straightforward for an operator to reuse its existing identity management system in a monolithic NFV deployment scenario. There is no need for provisioning existing users in Keystone. In the common nomenclature of identify federation, the user is called the principal and Keystone the relying party.

The involvement of a third party to vouch for the authenticity of the user results in a new authentication workflow. When receiving an authentication request, Keystone, instead of handling the request itself, redirects it to the identity provider (typically through the user agent). Upon receiving the redirected request, the identity provider performs the steps to authenticate the user, and then redirects the result as an attestation to Keystone (through the user agent). If the user is attested authentic, Keystone generates an un-scoped token, based on which the user can find out the accessible projects and obtain another token with a proper scope. Tokens generated for a federated user are distinguishable. They carry information related to federation, such as the name of the identity provider, the identity federation protocol, and the associated groups.

Keystone has the flexibility to support multiple identity federation protocols, such as the Security Assertion Mark-up Language (SAML) 2.0, OpenID Connect [i.8], Kerberos, and Application Bridging for Federated Access Beyond web (ABFAB) [i.9]. Support for SAML 2.0 is already available. Its use is configurable through `keystone.conf` (i.e. adding `saml2` to the list of authentication methods). Support for other protocols is in the works.

Keystone supports management of identity providers, including creation, deletion, update, and discovery. An identity provider may not manage users in terms of the same attributes as Keystone. So attribute mapping is necessary. To this end, Keystone supports management of attribute mapping on a protocol basis for each identity provider. Mapping is done through a set of rules. Each rule maps a local attribute in Keystone to a remote attribute in the identity provider. Typically a federated user is mapped to a group in Keystone and given the roles associated with the group. The default backend for identity providers and mappings is SQL.

Identity federation is also applicable to token verification across two different OpenStack clouds. As a result, a token generated by one cloud can be used to access another cloud. Such Keystone federation is under development. Again, trust between the two OpenStack clouds is assumed and it is established through a separate process.

5.8 Identity API Access Control

API access control in Keystone is policy-based. To a certain degree, policy is configurable through a JSON file; what gets enforced (or coded) is not always consistent. In the policy file, a set of rudimentary rules can be specified for each call in the identity API. A rule prescribes a required role (e.g. `admin`) or a required matching condition for an attribute (e.g. `user_id`) of the token and another parameter (e.g. `trust.trustor_user_id`) passed in the request. If a rule is expected but undefined, the default rule that requires the admin role applies. The name and location of the JSON file is configurable through `keystone.conf`. The default name is `policy.json`.

Each OpenStack module has its own policy file. There is no inbuilt mechanism for consistency across OpenStack in terms of policy definition, decision, and enforcement. In general, protection of policy files is critical. There shall be strict access control and audit trails.

5.9 Password Hashing

Keystone provides native support for password management, including password hashing.

Table 7

Cryptographic module used	`passlib.hash.sha512_crypt.encrypt()`
Hashing algorithm	hard coded (SHA512)
Configurability	The number of rounds of SHA-512 to use in generation of the salt is configurable via `CONF.crypt_strength (default=40000)` in `keystone.conf`.
Invoking module	`passlib.hash.sha512_crypt.encrypt()`
NOTE: In Passlib, the default number of rounds is 100 000.	

5.10 Time Synchronization

Given that token expiration is a component of Identity and Access Management, time synchronization among the servers making up an OpenStack cloud is critical. Time synchronization is not provided by OpenStack, and shall be configured through the underlying operating system services, such as the Network Time Protocol (NTP) [i.10] or the Precision Time Protocol (PTP) [i.11]. The time service shall also be securely configured to prevent servers from being attacked in a way to cause an expired token from being considered still valid.

6 Communication Security

Communication security in an OpenStack-based infrastructure can be effected through means such as TLS, the Secure Shell (SSH) protocol, and VPNaaS.

TLS supports confidentiality and integrity of communication over TCP, and authentication of communicating peers. For TCP-based services in OpenStack, the use of TLS can be controlled through configuration (e.g. through the `use_ssl` flag in a service configuration file). But TLS is not necessarily required by default where it is applicable, such as access to Glance, Neutron and Virtual Network Computing (VNC) connections. OpenStack further allows TLS connections to be set up when certificates cannot be verified. This may be controlled by the `api_insecure` or `insecure` flag, which is disabled by default. The OpenStack proper supports TLS through the OpenSSL library, which is configurable to forbid its predecessor SSL and the cipher suites that are known to be vulnerable.

The security functions supported by SSH are similar to those by TLS. A key difference is that SSH does not require certificate-based authentication for servers. SSH is typically used for remote login and command execution. In the OpenStack proper, the following controls are available for SSH:

- Injection of SSH key pairs to guest virtual machines for access by tenants [i.10]. This can be done by the metadata service or by the hypervisor. In the latter case, the `inject_key` flag provides the control and it is disabled by default. Note that the specific SSH software in use is application dependent.
- Host key checking upon first connection to a storage backend. Certain drivers in Cinder (the block storage service) support issuing commands for execution on a storage backend through SSH. The `strict_ssh_host_key_policy` flag controls whether an SSH client is to trust the public key of a newly encountered SSH server. It is disabled by default. If it is enabled, Cinder will allow connection to a backend only if the first host key presented is specified in the `ssh_hosts_key_file` file. Cinder supports an SSH client based on the Paramiko library.

VPNaaS is part of Neutron, which provides the networking service in OpenStack. To date, VPNaaS supports site-to-site IPsec tunnels. Standard transform protocols (i.e. ESP and AH) and modes (i.e. *tunnel* and *transport*) are supported. The defaults are hardcoded to ESP and *tunnel*. The supported authentication algorithm is SHA-1; and encryption algorithms are 3DES, AES-128, AES-192 and AES-256 (with the default hardcoded to AES-128). These are also applicable to the Internet Key Exchange (IKE) protocol. For key exchange specifically, the supported phase 1 operation is limited to Main Mode (which is reasonable) and the supported authentication method is limited to pre-shared secrets stored in files. In addition, the supported Diffie-Hellman Groups are limited to 2 (1 024-bit modulus), 5 (1 536-bit modulus), and 14 (2 048-bit modulus). Here the default is hardcoded to Group 5, which provides the minimum protection to AES-128.

The IPsec driver to use is configurable. The default driver is based on Openswan, which, in turn, uses the Network Security Services (NSS) libraries for cryptographic operations.

Note that it is possible to secure the communication between two compute hosts in general through mechanisms such as IPsec.

7 Stored data security

7.1 Block Storage Encryption

Presently OpenStack provides limited native support for storage encryption. Only volumes in block storage can be encrypted. Furthermore, secure key management, which is essential, is still in development under the Barbican project. (Cinder is the module in OpenStack providing the persistent block storage service.)

To date, encryption (as well as decryption) is handled by Nova through the Linux kernel *dm-crypt* module. A provisional configuration-based key manager is used by default to provide a single encryption key for all volumes. Under such a situation, the secrecy of encrypted volumes depends on the secrecy of the key. Given that this key is specified in the clear in a configuration file, access to the file shall be tightly controlled and monitored.

In the long run, a full-fledged key manager is expected to have the capabilities to generate individual volume encryption keys on demand and store them securely, among other things. To mitigate the impact of a compromise, the key manager also aims to avoid storing any information that can link keys to volumes. Cinder can already keep the key-volume association information as part of the volume metadata.

dm-crypt offers transparent encryption of block devices. To a virtual machine, a normal block device is attached, and I/O operations function as usual. *dm-crypt* provides mapping to the backend device with the required encryption and decryption using the cryptographic primitives of the kernel crypto API. The supported algorithms and modes in the kernel crypto API are platform dependent. The default for *dm-crypt* is also platform dependent.

An administrator may elect to use an encrypted Cinder volume and set the cipher, key size and other encryption specifics using a command-line interface through the Horizon dashboard. In this case, the interface to *dm-crypt* may be through the plain *cryptsetup* utility or its extension with Linux Unified Key Setup (LUKS). This is also settable by the administrator through the command line interface. Both *cryptsetup* and *cryptsetup-LUKS* use the user-space cryptographic module *libgcrypt* for key generation.

NOTE 1: *cryptosetup-LUKS* uses a passphrase to derive a key for encrypting a volume encryption key that it generates separately. In contrast, *cryptosetup* deals with only volume encryption keys. It may use a passphrase to derive a volume encryption key or import the key generated elsewhere.

NOTE 2: Both *cryptosetup* and *cryptosetup-LUKS* need to run as root.

NOTE 3: In the case of the configuration-based key manager, the fixed key serves as the encryption key to *cryptosetup* and the passphrase to c*ryptosetup-LUKS*.

Encryption of ephemeral storage is supported similarly. Here whether to encrypt the ephemeral storage of a virtual machine instance, the cipher to use, and key size is configurable through the corresponding options in the `ephemeral_storage_encryption` group in `nova.conf`. Encryption is disabled by default. If it is enabled, the default cipher is `aes-xts-plain64` and key size 512 bits. Eventually it will be possible for each instance to have an individual encryption key. The key can be fetched from the key manager. For the time being, the default key manager is configuration-based and supports a single key for all instances through `nova.conf`.

Encryption of ephemeral storage makes tenant data unreadable after a virtual machine has been terminated. An administrator can still access tenant data, when the machine is running, paused, suspended or powered off. To prevent data breach, steps are further taken to disconnect the dm-crypt device and flushing the encryption key from memory, when the virtual machine is suspended or powered off.

7.2 Logical Volume Sanitization

The Cinder logical volume manager supports volume wiping to mitigate leakage of sensitive information. Presently, there are two options available for the underlying wiping mechanism. In the first option, a volume is overwritten once with all zeros through the Linux utility `dd`. In the second option, a volume is overwritten three times with "random" data through the Linux utility `shred`. Wiping is applicable to "thick" volumes only. A parameter in the `cinder.conf` controls whether to apply wiping and, if so, the option to use. The default is to use overwriting with all zeros in one pass.

NOTE 1: `shred` assumes that data are written *in place* (i.e. at the same physical location). It is ineffective for cases where this assumption is invalid, such as backend solid-state storage devices.

NOTE 2: `shred` needs to run as root.

NOTE 3: Logical volume sanitization is a matter separate from end-of-life sanitization. A storage device with logical volume sanitation still needs to be disposed according to the best practices (e.g. physically destroyed) when reaching the end of life.

NOTE 4: For thin volumes, overwriting with zeros is implied. An attempt to read a block not yet written will return zeros.

NOTE 5: Encrypting volumes will ease the need for data erasure operations. Here the problem is relegated to secure deletion of encryption keys.

8 Firewalling, zoning and topology hiding

8.1 Security group

Security groups are the primary mechanism that tenants can use to control network traffic from and to virtual machines or network interfaces. A security group is defined by a set of rules. A rule consists of specific conditions (mainly pertaining to the type, source and destination of traffic) and the action (e.g. drop, reject, or accept) to be taken if the conditions are satisfied. For example, a rule could be specified to allow all outgoing traffic, support anti-spoofing of MAC addresses, and block illegitimate DHCP messages. (A rule may even reference a security group as a traffic source. This can shield dynamic IP addresses and reduce the churn of security group rules.) Overall, traffic is allowed only if there is a rule permits it. Security groups are tenant-specific. Virtual machines (or network interfaces) are assigned security groups when they are created.

Security groups may be provided by Nova-network or Neutron via configuration. Either way, the underlying implementation is based on Linux *iptables*. By default, security groups are provided by Nova-network. But Neutron is the recommended provider because of its advanced features and flexibility to use external plug-ins.

In Nova, whether to delegate security groups to another component is controlled by two configuration parameters: `firewall_driver` and `security_group_api`. To let Neutron handle security groups, the two parameters need to be set to the dummy, no-operation driver (i.e. `nova.virt.firewall.NoopFirewallDriver`) and `neutron`, respectively. (By default, `firewall_driver` is set to the *libvirt iptables* driver and `security_group_api` to `nova`.) In addition, the Neutron plug-in or agent in force shall be configured with a functional firewall driver and with the `enable_security_group` flag enabled. Note that the `enable_security_group` flag is enabled by default. To avoid potential conflicts, native Neutron agents (e.g. the Linux Bridge and Open vSwitch agents) are configured with the no-operation firewall driver by default.

In Neutron, security groups are applied to virtual network interfaces (a.k.a. Neutron ports). (In Nova, security groups are applied to virtual machines. As a result, all network interfaces on a virtual machine will have the same security groups.) A Neutron port may be associated with one or more security groups upon creation. If it is not explicitly assigned a security group, the tenant's *default* security group applies. By default, the *default* security group allows all egress traffic (subject to anti-spoofing of MAC/IP addresses and DHCP messages), but limits ingress traffic to only that from a security group member and an essential service (e.g. ICMPv6 for route advertisement). The *default* security group (like other security groups) is customizable on a per-tenant basis.

Neutron security groups prevent traffic to pass through an intermediate virtual machine. To support virtual network functions such as routers and firewalls, the port construct has been extended with the attribute `port_security_enabled`. The attribute is essentially a flag and it is enabled by default. In this case, security-group operations work the same way as before. If the flag is disabled, the port cannot be assigned a security group or an allowed address pair. The flag is set upon port-creation request. Only a user with a privileged role (such as the cloud administrator or owner) can issue such requests. Given that security-group and anti-spoofing rules no longer apply, the resulted ports will need to be monitored with a separate mechanism for detection of anomalies, such as address spoofing.

OpenStack provides a quota mechanism to limit resource utilization by tenants. For security groups Neutron supports configuration of quotas through the following parameters:

- `quota_security_group_rule` (the number of rules per security group with *100* as the default)
- `quota_security_group` (the number of allowed security groups per project with *10* as the default)

Similar controls are available in Nova:

- `quota_security_group_rules` (the number of rules per security group with *20* as the default)
- `quota_security_groups` (the number of allowed security groups per project with *10* as the default)

In the Nova case, there is another configurable flag (i.e. `allow_same_net_traffic`) to globally control whether virtual machines on the same subnet may communicate with each other. The flag is enabled by default. If the `allow_same_net_traffic` flag is disabled, only the defined security groups apply.

8.2 Anti-spoofing

Nova-network supports anti-spoofing of MAC addresses, IP addresses, ARP messages and DHCP messages through the *libvirt* network filter feature. Neutron is expected to provide equivalent support over time, although it cannot counter spoofing of ARP messages to date. (Anti-spoofing of ARP messages in Neutron will be implemented based on *ebtables*.) In general, regardless of the networking service in use, anti-spoofing is enforced by default.

8.3 Network Address Translation

In OpenStack, IP addresses are classified as fixed or floating. Fixed IP addresses are private, not routable externally. A virtual machine is automatically assigned a private IP address upon creation in Nova. The IP address stays with the virtual machine until it is terminated. In contrast, a floating IP address can be assigned to a running virtual machine and be disassociated from it at any time. Floating IP addresses are typically public, routable. Nova allows a virtual machine to be assigned a public IP address upon creation as well. This is controlled by the flag `auto_assign_floating_ip` in `nova.conf`. By default, the flag is disabled.

NOTE: A prerequisite for Nova to handle public IP addresses is that the public interface to be bound to such addresses has been configured. The interface is specified by the `public_interface` parameter in `nova.conf`.

Network address translation makes it possible for a virtual machine with a private IP address to communicate with a host on a public network. It may be provided by Nova-network or Neutron. In the case of Nova-network, the implementation is based on Linux *iptables*. In particular, the configuration parameter `force_snat_range` specifies the destination floating IP range to which traffic shall be subject to source network address translation.

In the case of Neutron, the L3 agent handles network address translation. Neutron supports dynamic provision and configuration of virtual routers. A router may serve as a gateway connecting one or more private networks to a public network. To support and isolate multiple routers on the same host, the L3 agent builds on Linux network namespaces. Hence, each router has its own namespace, which makes it possible for tenants to have overlapping IP addresses. Network address translation is effected by modifying the *iptables* in the router namespace.

8.4 Network isolation

Both Nova-network and Neutron support VLAN for segregating traffic of different tenants. Neutron further supports VXLAN, and GRE for bridging VLANs across different networks. Neutron also has the flexibility to use plug-ins to support additional L2 tunneling technologies.

8.5 Firewall-as-a-service

Neutron supports Firewall-as-a-Service through a native or external driver. The firewalls supported are at network perimeters. The native driver is based on *iptables*. It supports configuration of related quotas through the following parameters:

- `quota_firewall` (the number of allowed firewall per tenant with *1* as the default).
- `quota_firewall_policy` (the number of allowed firewall policy per tenant with *1* as the default).
- `quota_firewall_rule` (the number of allowed firewall rules per tenant with *100* as the default).

NOTE: Firewall-as-a-Service is still considered experimental as of the Kilo release.

9 Availability

In general, OpenStack deployments rely on external technologies to achieve high availability as well as to mitigate denial-of-service attacks. But the OpenStack proper does provide native support for certain related features.

Compute hosts can be organized into zones of independent availability properties. The organization criteria are flexible. They typically have to do with geo-location, network segmentation, power source, and certain hardware attributes. Hence, a user can request to place virtual machines in more than one zone to achieve high availability. The availability zone of a compute host is controlled by the configuration parameter `default_availability_zone` (with the default value `nova`) in `nova.conf`. Another configuration parameter `default_schedule_zone` controls the available zone where a new virtual machine is to be provisioned. By default, a new virtual machine is provisioned in one of the default availability zones.

Block storage can be organized into availability zones as well. The availability zone of a storage node is controlled by the configuration parameter `storage_availability_zone` (with the default value `nova`) in `cinder.conf`. Another configuration parameter `default_availability_zone` controls the available zone where a new volume is to be provisioned. By default, a new volume is provisioned in the zone specified by `storage_availability_zone`. In addition, the availability zone of a cloned volume can be controlled by the flag `cloned_volume_same_az.` By default, the flag is enabled so that a new volume resulted from cloning is in the same availability zone as that of the source volume. Finally, Nova allows configuration of whether to limit a virtual machine and an attached volume to be in the same availability zone through the flag `cross_az_attach`. By default, the flag is enabled.

Swift (the object storage service) supports high availability natively. It is cluster-based and stores multiple copies of every object as separately as possible. Swift can distinguish the locations where objects are placed by region, zone, server, and disk drive. Regions and zones are logical, definable by administrators. A region usually has a well-defined geographical boundary, while a zone is akin to an availability zone in Nova (or Cinder). An administrator specifies the number of regions as well as the number of copies to be stored for each object when creating the initial map of a cluster with a Swift utility. (The recommended number of copies for each object is 3.) The servers and disk drives are each assigned a region and a zone when they are added to a cluster. Again, this is done by the administrator with a special utility. Eventually, the administrator will need to create a new *ring* (or rebalance an old *ring*) to reflect the assignment. *Ring* is a fundamental construct in Swift for locating objects. It is based on a modified version of consistent hashing.

Neutron supports high availability for virtual routers based on the Virtual Router Redundancy Protocol. Whether to enable high availability is controlled by the flag `l3_ha` in `nova.conf`. By default, the flag is disabled. Another configuration parameter (`max_l3_agents_per_router`) controls the number of L3 agents to host a virtual router. The number shall be at least 2 (the default value) and cannot exceed the number network nodes in a deployment. Yet another configuration parameter (`min_l3_agents_per_router`) controls the minimum number of L3 agents that shall be up when a highly available virtual router is created. The number shall be at least 2 (the default value).

Neutron also supports multiple DHCP agents per network. The configuration parameter `dhcp_agents_per_network` controls the number of DHCP agents per network. The default value for the parameter is 1. To achieve high availability, more than one DHCP agent per network is required. Note that the Neutron DHCP agent provides a control for the maximum number of DHCP leases via the configuration parameter `dnsmasq_lease_max`. The control helps mitigate denial-of-service attacks.

10 Logging and monitoring

10.1 Logging

The native code of OpenStack uses the Python logging module for application logging. The module provides standard features, such as a common set of logging levels for classifying the logged events, log file rotation, and runtime logging configuration. The logging levels, in the ascending order, include DEBUG, INFO, WARNING, ERROR, and CRITICAL. The actual logs generated at run time are configurable. A log entry is generated for an event if the logging level assigned to it is higher than the configured level. So the total amount of information logged is increasing with the logging level.

It is also possible to add custom logging levels according to application needs. To date, OpenStack has defined a custom logging level, AUDIT. But its distinction from other levels is unclear and it is rarely used.

NOTE: According to https://review.openstack.org/#/c/91446/, the plan is to get rid of AUDIT.

OpenStack provides an overlay for logging configuration on a per module basis through a module-specific configuration file (e.g. the `keystone.conf` file for Keystone). Among other things, the logging level, where to keep logs, and the formats of logged messages are configurable.

The logging level is set to WARNING by default for most modules. It may be changed to INFO or DEBUG. The logging level for a set of third-party Python libraries is also configurable through the same configuration files. The set consists of the following libraries together with their default logging levels:

- amqp=WARN
- amqplib=WARN
- boto=WARN
- iso8601=WARN
- keystonemiddleware=WARN
- oslo.messaging=INFO
- qpid=WARN
- requests.packages.urllib3.connectionpool=WARN

- requests.packages.urllib3.util.retry=WARN
- routes.middleware=WARN
- sqlalchemy=WARN
- stevedore=WARN
- suds=INFO
- urllib3.connectionpool=WARN
- urllib3.util.retry=WARN
- websocket=WARN

In general, developers control what events to log, and the message and logging level for each event. Because of the large number of developers involved, there is no consistent use of logging levels throughout a module or across modules in OpenStack. For example, a user token validation error is logged at the INFO level in one place, while lack of credentials is logged at the ERROR level in another place. It has also been known that the logs of an OpenStack deployment contain stack traces and errors even in a normal condition. Logs are useful in several aspects, such as debugging, troubleshooting, auditing, and yielding insights. Prudent and consistent use of logging levels simplifies log analysis. It is desirable to have logging guidelines in place and to enforce them through code review and verification tools.

A logging best practice is to avoid logging unnecessary sensitive information outside of a trust boundary. It is known that OpenStack logs contain sensitive information (e.g. passwords and authorization tokens). The OpenStack community is working toward implementing the best practice.

Where to keep logs is configurable by specifying the file name and base directory in the relevant configuration files. The default is stdout. In general, each module keeps a separate log file. To ease management of multiple log files, OpenStack supports sending them to a centralized logging server based on *syslog*. Centralized logging, however, is disabled by default. When centralized logging is enabled, module-specific facilities may be further configured to distinguish logs from different modules. The *syslog* protocol in use is configurable and it may be as specified in IETF RFC 3164 [i.14] or IETF RFC 5424 [i.15]. In the long run, only IETF RFC 5424 [i.15] will be supported. The actual *syslog* implementation is outside of OpenStack and is platform dependent. A best practice for centralized logging is to use a secure, reliable transport for transferring logs. OpenStack also relies on external utilities for general log management, such as access control and protection of logs.

10.2 Event notification

Events, similar to logs, can serve as a basis for purposes such as accounting, security monitoring, troubleshooting, and auditing. Events may reflect state changes or access to services. OpenStack supports event notifications through an API built on the AMQP-based messaging service provided by the Oslo module. Other modules emit and consume notifications through the API. Multiple backend drivers are supported for the actual transport of notifications, including RabbitMQ, Qpid, and ZeroMQ. The driver to use is configurable. The default is RabbitMQ (controlled through the RPC backend driver parameter).

An event notification is uniquely identifiable and conveys information such as who publishes the event, what the event is about (e.g. the involved operation), when the event took place, and the priority of the event. The set of priority levels in use mirrors that of logging levels. Depending on the modules, the priority level may be configurable. This is the case for Nova, Trove (the database service), Sahara (the data processing service), Heat, and Ceilometer (the telemetry service).

OpenStack also supports generation of event records according to the Cloud Auditing Data Federation (CADF) [i.18] schema from the DMTF (http://www.dmtf.org/standards/cadf) through the pyCADF library. Having standard event records allows federation and aggregation of auditing data from different cloud infrastructures. The pyCADF library also provides a WSGI middleware filter that can be used to audit API requests and responses in OpenStack.

In Keystone, a notification is sent when a resource of a type in a hard-coded set is created, updated, or deleted successfully. In case of failure, only normal exception handling is in effect. The set of resource types includes project, role, user, trust, and policy, to name a few. The priority level is not configurable but hard coded to INFO.

For transporting notifications the configured RPC backend driver is used by default. On CADF, Keystone to date supports generating compliant records for authentication and authorization events (including federated authentication and role assignment).

11 Compute isolation

The compute hosts in an availability zone can be further organized in terms of aggregates. The compute hosts in the same aggregate share a set of attributes (such as a tenant or a hardware capability) defined by administrators. The shared attributes become the metadata associated with the host aggregate. It is possible for a compute host to belong to more than one aggregate. A generic application of host aggregates is to support compute allocation based on custom attributes and it is embodied in the `AggregateInstanceExtraSpecsFilter` filter. This generic filter has the flexibility to support varied use case. For example, compute hosts can be segregated according to security zones and a host in an aggregate can be allocated based on the associated zone. Another use case is to have tenant-specific aggregates for multi-tenancy hardening. For whatever reasons, Nova has implemented a filter (`AggregateInstanceExtraSpecsFilter`) that is specific to the use case too. The filter allows a tenant to place a virtual machine only on a host aggregate associated with the tenant. Another specific (and seemingly redundant) filter supported is `AggregateTypeAffinityFilter`. The filter allows a virtual machine to be placed on a host aggregate associated with a particular type.

Besides host aggregates, Nova also supports a set of filters applicable to compute isolation. In particular, the filter `IsolatedHostsFilter` can limit instantiation of a special set of images to a special set of hosts. Whether to do so is configurable. The configuration parameters `isolated_images` and `isolated_hosts` specify the respective sets. To further control whether the special set of hosts can run images outside of the special set, the flag `restrict_isolated_hosts_to_isolated_images` is provided. By default, the flag is enabled to enforce strict isolation.

Other filters are as follows:

- `DifferentHostFilter`, which allows a new virtual machine to be placed on a host only if there are no virtual machines on a specific list residing there.
- `SameHostFilter`, which allows a new virtual machine to be placed on the same host as another virtual machine on a specific list.
- `ServerGroupAffinityFilter`, which allows a virtual machine of a specific group to be placed on the same host as other virtual machines in the same group.
- `ServerGroupAntiAffinityFilter`, which allows every virtual machine of a specific group to be placed on a different host.
- `TrustedFilter`, which allows a virtual machine to be placed on a host that supports remote attestation.
- `TypeAffinityFilter`, which allows a virtual machine of a certain type to be placed on a host only if no virtual machines of a different type reside there.

The use of these filters is controlled by the parameter `scheduler_default_filters` in `nova.conf`. The parameter specifies the list of filters in effect when a request does not specify a filter explicitly. By default, it includes the following filters:

- RetryFilter
- AvailabilityZoneFilter
- RamFilter
- ComputeFilter
- ComputeCapabilitiesFilter
- ImagePropertiesFilter
- ServerGroupAntiAffinityFilter

- ServerGroupAffinityFilter

12 Guidance on the use of OpenStack in NFV

OpenStack services are configurable through a myriad of configuration files. These files often contain sensitive information, such as admin passwords and encryption keys. Their access shall be reserved for a strict set of privileged users. The best practices for strong access control, file integrity protection, and monitoring of access activities shall be implemented.

The default configurations of OpenStack are not always optimal for security. For example, by default, access to messaging queues is over TCP, remote console access is over TCP, SHA-1 is used as the digest algorithm for signatures, volume encryption is disabled, and the method for wiping old volumes is "overwriting with zeros." In an NFV environment, OpenStack configurations will need to be hardened. The hardening specifics are deployment-dependent. Some common recommendations for configuration hardening are as follows:

- TLS should be used wherever applicable, including access to API endpoints, internal messaging queues, database servers, and virtual machine consoles. To use TLS properly, both clients and servers shall be configured to user public key certificates, forbid protocol versions earlier than TLS v1.2 (including the SSL predecessors), and disallow insecure cipher suites. TLS requires the availability of a PKI that generates public key certificates as specified in IETF RFC 5280 [i.16]. In general, the best practices for managing certificates and keys (especially those for signing) need to be followed. A key challenge to PKI deployment is tracking of the revocation status of certificates to ensure their validity. The open source project Anchor aims to eliminate the need for certificate revocation. It is developing an ephemeral PKI service that supports automated verification of certificate signing requests and signing of short-lived (namely ephemeral) certificates. Anchor is an option for further study.

- Advanced scheduler filters in Nova that can further enforce compute isolation shall be enabled. Such filters, among other things, allow selection of compute hosts according to tenant, application type, security zone, and trust.

- Neutron rather than Nova shall be used for providing networking services. Neutron together with external plug-ins provides more robust support for network segregation through load balancer as a service, firewall as a service, and VPN as a service.

- The internal communication between OpenStack components shall be on an isolated management network not rea\chable from external networks. Similarly, networks for inter-instance communication and for external networks reachability shall be separate and distinguishable in terms of security policies.

- Encryption of Cinder volumes and ephemeral volumes should be enabled.

- The sanitization option of overwriting existing volumes three times with "random" data shall be used, if the volumes are unencrypted.

- Centralized logging based on *syslog* shall be enabled.

OpenStack consists of loosely-coupled modules. There was no master plan to address security consistently across the different modules. The use of Keystone to provide centralized identity and access management is a step in the right direction. As the gatekeeper to other OpenStack services, it is essential that Keystone be configured strictly so that both the storage and transport for tokens are protected based on best practices. In addition, it is highly recommended to configure Keystone to use an external system that supports strong user authentication; passwords are the only method for initial user authentication that Keystone supports natively. Furthermore, Keystone logs needs to be monitored closely. Naturally, the best practices for log management need to be followed as well.

OpenStack is in the process of deprecating its native support for rate limiting of API requests. External measures shall be used to limit the rate of requests to public API endpoints.

The OpenStack Security Guide [i.12] provides additional guidelines on deploying and operating OpenStack-based infrastructures of varied security requirements.

It goes without saying that as any other software, OpenStack in use shall be kept-up-to-date, with security patches applied in a timely fashion.

13 Recommended OpenStack enhancements in support of NFV

NFV applications usually have high performance requirements. SR-IOV is a technology that can be used to boost network I/O performance in NFV. It achieves performance gains by allowing a guest virtual machine to have a direct data path to the network controller. To date, data transmission over such a path is not part of what can be monitored by the existing OpenStack inbuilt agents. Additional mechanisms will need to be developed to address the gap.

NOTE: An alternative to SR-IOV is vSwitch acceleration. But certain accelerated-vSwitch implementations might be vulnerable to DoS attacks through direct memory access.

Encryption is a main enabler for confidentiality protection. For encryption to be effective, encryption keys need to be properly managed and protected. To date, encryption keys used in OpenStack are stored in the clear, which is not a security practice. The project Barbican on key management helps address the gap. With Barbican, encryption keys can be encrypted with Key Encryptions Keys on a per tenant basis. Similarly, Key Encryptions Keys can be encrypted with Master Key Encryption Keys on a per tenant basis. As such, secure storage and management of Master Key Encryption Keys is critical and best practices call for the use of dedicated hardware-based systems. The OpenStack community needs to further the Barbican effort to stabilize the code base for production user and to address the integration with hardware security modules or other hardware-based solutions, such as isolated secure execution environments. Note that the isolated secure execution environment is generic and has no connection to the Trusted Execution Environment or Secure Execution specified by the GlobalPlatform organization.

Secure deletion provides assurance that deleted data in shared storage cannot be recovered. Support for secure deletion is preliminary to date in OpenStack. Swift (which, in turn, is used by services such as Glance and Trove) does not support secure deletion of objects. Nova and Cinder provides limited support for logical volume sanitization. Two methods are supported. Neither is strong enough. Moreover, the more secure method based on `shred` assumes that data are written *in place*. This method is ineffective for cases where this assumption is invalid, such as backend solid-state drives where write operations are distributed evenly to avoid localized wearing. Yet solid state drives support higher I/O throughput than traditional hard disk drives. They are expected to be in wide use to meet NFV performance requirements. Hence, secure deletion methods that are stronger in general as well as secure deletion methods that are suitable for solid state storage are in order. The Self Encrypting Drive (SED) technology [i.13] is designed to provide full or partial cryptographic erasure of storage and may be applicable here. It is worth noting that secure deletion of data can be reduced to secure deletion of encryption keys if data are encrypted.

Legal and regulatory conditions may restrict where certain applications can run, where user data can be stored, and whether certain user data can flow across national or organizational borders. Lawful interception is an example. Those restrictions give rise to the need for supporting trusted geo-locations in NFV and might call for specialized hardware in support of trusted computing to provide the level of trust required. In the case of lawful interception, there is a further challenge to shield the highly confidential workloads on a host from root at the host level. Active study is underway to address the challenge. A hardware solution based on trusted computing alone might be insufficient.

Annex A (informative): Authors & contributors

The following people have contributed to the present document:

Rapporteur:

Ms. Huilan, Lu, Alcatel-Lucent

Other contributors:

Mr. Igor, Faynberg, Alcatel-Lucent

Mr. Nathan Kinder, Red Hat

Mr. Bob, Moskowitz, Verizon

Mr. T., Prabhu, NEC

Mr. Kapil Sood, Intel

Annex B (informative): Bibliography

ETSI GS NFV-SEC 003: "Network Functions Virtualisation (NFV); NFV Security; Security and Trust Guidance", clause 4.4.3.2.3.

History

Document history		
V1.1.1	August 2015	Publication