Keystone Security GAP and Threat Identification (Quick Study)

OpenStack Folsom Release

Prepared by:

Ericsson

Table of Contents

1. Basic Information

1.1. Background

The report provides a quick study on security gap and threat identification for OpenStack Identity and Access management - code named Keystone. The report includes GAP analysis of Keystone against key features of an Identity and Access management system. It also includes possible threats against each component/sub-component/interfaces of Keystone. The threats against Keystone subcomponents and interfaces are by no means an exhaustive list to cover all possibilities.

OpenStack [openstack] is an operating system for managing cloud. OpenStack provides a suite of services to manage large pools of compute, storage, and networking resources in a scalable fashion. The services itself expose their management through a set of web APIs. Currently, OpenStack supports two types of APIs: the OS API and the legacy EC2 API.

1.2. Methodology:

The report uses the following steps for GAP and Threat identification.

- Define conceptual model and security objectives of an Identity and Access management system.
- Identification of existing components and interfaces in Keystone.
- Mapping Keystone components and interfaces (Figure 5) against the conceptual model (Figure 1).
- Key features identification of components and interfaces from the conceptual model in Figure 1 (The key feature does not directly related to the security objectives and their requirements. This is something we can to focus on future analysis)
- Find GAPs in Keystone components and interfaces against key features.
- List possible threats for each component and interfaces in Keystone (Threats are not grouped or ranked, a possible future activity).

1.3. Assumptions

- Vanilla Folsom release of Keystone
- The interfaces between end users and Keystone REST API are protected by transport security
- The interfaces between Backend (DB) and Logic layer (service layer) is protected by transport security
- The physical machine in which the keystone is installed has proper security measure to prevent unauthorized access

2. Identity and Access Management

2.1. Generic Architecture

A generic Identity and Access Management (IAM) system consists of an access management service which controls the authentication and authorization; an identity management service for administering the accounts, a provisioning service for propagating accounts and privileges, a data store (directory services) for storing user and policy information.

The core of an IAM system is authentication and authorization. Authentication is the process of verifying credentials of a subject/resource/entity while authorization grants a subject to perform an action on a target resource.

A generic IAM system has the following security objectives [IMPP]:

- Authentication of users and administrators for each resource request
- Access control of resources i.e., only authorized users are allowed to access resources
- Recording of all security operations and transactions
- Account provisioning is only performed for entitled users
- Correct association of account attributes.

IAM System conceptual model

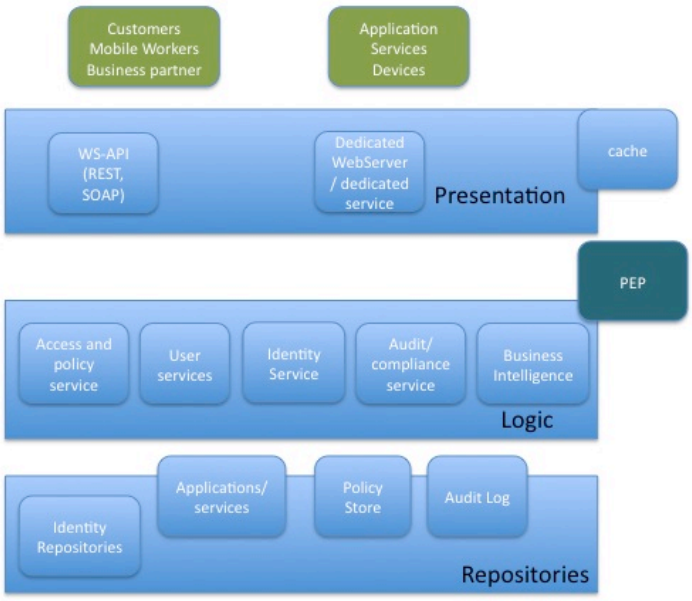

Figure-1: Conceptual model of IAM system (Adapted from SeCaaS implementation guidance [SeCaaS])

The conceptual model of IAM is based on the layered architecture – Presentation, Logic, and Repositories.

2.2. Keystone System Description

Keystone is the centralized Identity and access management (IAM) component of OpenStack. This is designed as a shim layer on top of pluggable data store (SQL, LDAP). It consists of several services (Identity, Token, Catalog, Policy) Ref: http://docs.openstack.org/developer/Keystone/architecture.html. Each service is written as a WSGI application (services can be managed through paste configuration file). Keystone supports two type of auth mechanism: EC2 style and OS style (tokenization). The Folsom releases, also supports signed token and SSL endpoint.

In Keystone, Authentication is centralized. The authorization part is still evolving (currently, supports rule based authorization controlled by each service). Centralized roles are supported but they act as a Meta data for the users/tenants. The current roles are not fully compliant with the RBAC model.

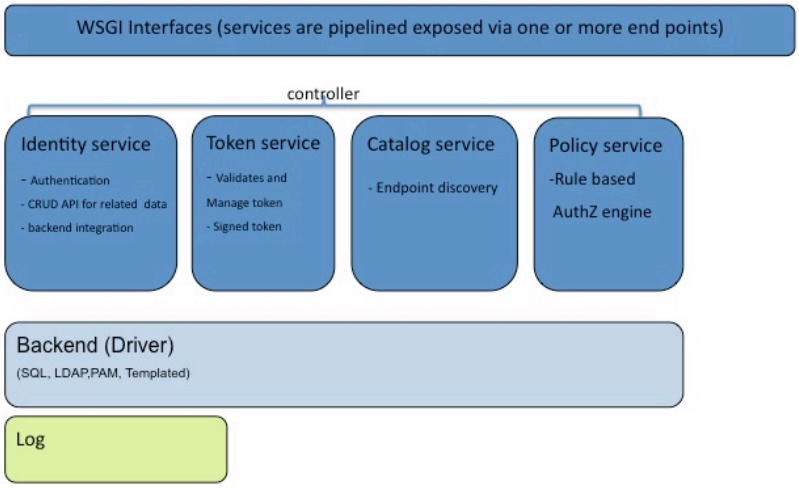

Figure-2: Keystone components and subcomponents

2.2.1. A Dissection of Keystone Middleware

Keystone middleware, a pluggable module, acts as an agent to communicate and validate token between the Keystone server and services within openstack. It enables a common mechanism for centralized authentication and authorization for OpenStack services. Some of the major characteristics of the Keystone middleware include:

- It is designed as a web service gateway interface (WSGI) middleware similar to other the middleware in the OpenStack projects.
- The middleware, itself, acts as a proxy that intercepts HTTP calls, validate the token with Keystone server, and populates (from validated token info) HTTP headers for the request context of downstream WSGI applications.

Operational Flow

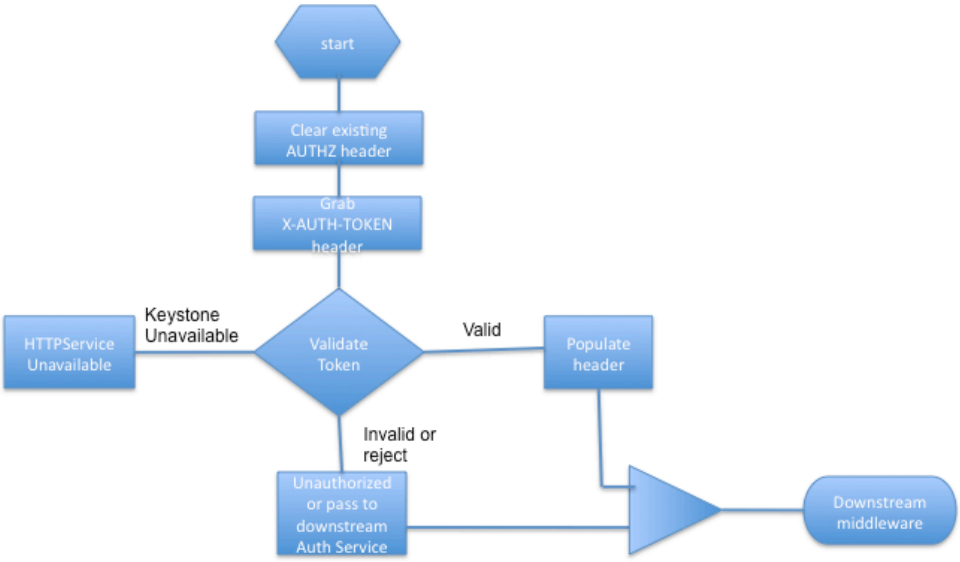

Figure-3: Keystone middleware flow chart

2.2.2. Authorization in Nova

In OpenStack, each service is responsible for authorization of any request authenticated by the Keystone server. Each service makes this decision based on the defined policy (defined rules in policy file) and context information (e.g., usernames, tenants, roles) it receives from a validated request. Although, it supports centralized role (role information is provided by Keystone service) based authorization, the current role is not fully compliant with RBAC model. Some of the missing features include

- Separation of duty among roles
- Role hierarchy
- Role to permission assignment.
- Centralized policy management
- Attribute based access control

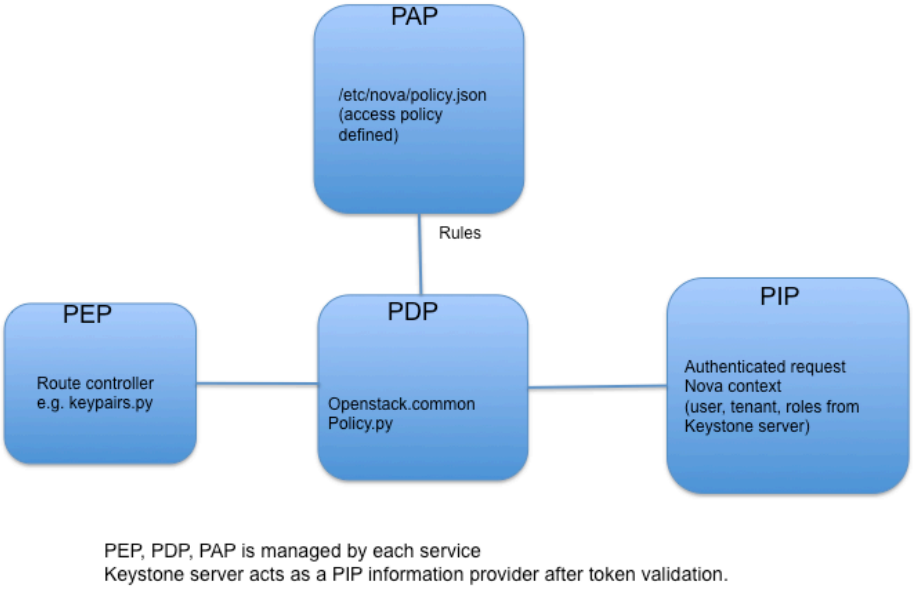

Figure-4: Authorization in Nova (or in any OpenStack Service)

2.2.3. Keystone Logical Model

The logical model of keystone is presented in Figure 2. The logical model shows how keystone fits within the conceptual model.

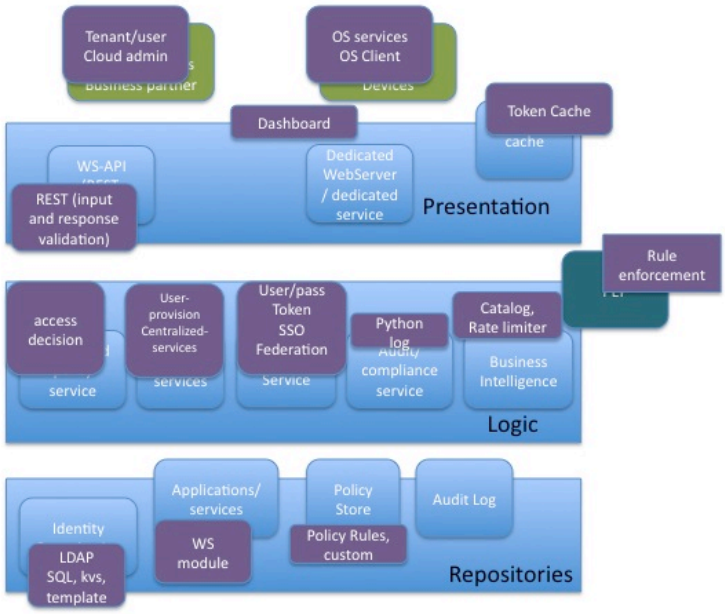

Figure-5: Keystone Logical model (Folsom Release)

2.3. The GAP Points and identified Threats

Entities Interacting with IAM service:

Feature	Keystone	Threats
Customers, Mobile workers, business partners	Tenants/users/ cloud admin can interact with Keystone using direct call to WS-API (additional transport security can be added)	1) Insufficient TLS protection - MITM attack in TLS session, Unbounded TLS session, TLS/PKI trust in certificate, spoofing attack for cert 2) Eavesdropping, Social engineering attack to the end users 3) Attacks on bearer token and unscoped token in the client or in the communication interface. 4) Attacks to the client machine/tools by malicious Web server
Customers, Mobile workers, business partners	Tenants/users/ cloud admin interact with Keystone using GUI interface or a Client tool (Transport security can be added)	1) Insufficient TLS protection - MITM attack in TLS session, Unbounded TLS session, TLS/PKI trust in certificate, spoofing attack for cert 2) Eavesdropping, Social engineering attack to the end users 3) Threats by exposing vulnerabilities of the browser and client tool vulnerabilities 4) Attacks to the client browser by malicious Web server 5) Attacks on bearer token and unscoped token in the client storage or in communication interface. 6) Un-validated redirects and forwards by the browser/user (attacker redirects the URL to an malicious one)
Applications, services (e.g., NOVA, QUANTUM)	Each service maintains a service account with the Keystone server. Through this, each service makes WS API call to verify user token. An alternate option is to use admin token for this purpose.	1) Insufficient TLS protection - MITM attack in TLS session, Unbounded TLS session, TLS/PKI trust in certificate, spoofing attack for cert 2) Insider Threat - Malicious service account in Keystone (by exposing vulnerabilities in service authentication)

Presentation layer:

Feature	Keystone	Threats
Support for WS API Message level security	Keystone exposes REST style WS-API for TOKEN, USER, TENANT, ROLE, ENDPOINT, SERCVICE, EC2-CREDENTIAL operations. Input and response validation is performed for all requests. REST API relies on transport layer security to secure the communication. So, TLS, unlike WS-security does not provide sign/encrypt a particular part of a message during transmission or storage phase. TLS does not provide e2e security.	1) Attacks by exposing the vulnerabilities of Web Server (web servers e.g., apache and bindings mod_wsgi) 2) Input validation Threats - Injection attack, attacks based on lack of secure parser (e.g., lack of secure XML parser – XML injection attack), attacks due to validation of incoming and outgoing content types (e.g., HTTP `Content-Type`, `Accept` header), Clickjack attack (due to `X-Frame-Options`) 3) Threats due to vulnerabilities in separation of permitted operations in exposed ports (admin port vs. public port) and exposure boundary. 4) Attacks by exposing vulnerabilities of the Web server/browser - CSRF and XSS attack, mixing of secure and not secure URL 5) Unbounded composition of transport and message level security, also impedance mismatch
Token caching	Support for token caching in the Keystone middleware agent (e.g., nova). X-Auth-Token caching in the Dashboard.	1) Attacks cached data (replay) - Attacks (replay, malicious access) on bearer token storage in Dashboard and also in keystone middleware.

Logical Layer:

Feature	Keystone	Threats
Access and policy Authentication:	By default, keystone uses token based centralized authentication using username /password. Token can be signed (a public/private key pair is used, the public part is signed by the CA). In addition to that, EC2 style authentication using access/secret key is supported (each request is signed using a secret key).	1) Account hijacking - unconstrained delegation (e.g., bearer token instead of key agreement), unscoped token and credentials 2) Cryptographic weakness (time() and Rand() weakness) 3) Token generator weakness (sequential token – UUID randomness)
Support for One-time Password (OTP) and strong authentication	Not by default, Keystone middleware can be modified to redirect authentication request to a strong authentication backend to support this feature.	4) Insider attack - Privileged persons (e.g., company stuff) access user credentials
Support Single Sign-On (SSO)	SSO through token. The token provides SSO within OpenStack Components.	5) Threats from mal-formatted data arriving from backend.
Support Federated Single Sign-On	Federation through backend integration (data store integration through LDAP – still immature). Front end federation e.g., SAML SSO is not supported by default.	6) Robot attack: Mass password guessing, Geographical attack: attack from country X 7) Availability - Deliberate failed login to lock victim's account
Support for risk based authentication	Not a built in feature	8) Out of channel attack (password reset, identity theft) 9) Protocol weakness attack (e.g., multiple protocols for authentication – attacker selects the weak protocol to attack)
Authentication middleware to verify authentication request towards your Authentication Server	Authtoken middleware is used for token verification towards Keystone server. Successful verification returns user roles.	
Access and policy Authorization	Authorization is performed by individual services (e.g., Nova). However, Keystone does provide role information to assist authorization process. No centralized authorization server.	1. Security mis-configuration - Access rule mis-configuration for users, conflicting access rules, lack of granular control over resources. 2. Data loss and breach due to policy mis-configuration. 3. Insecure direct object reference (unauthorized access to an object by limited authorized user)

Support for Role Based Access Control (RBAC)	Roles are supported in primitive form, roles act as an attribute to the user. No role-to-role assignment, role to permissions assignment.	4. Attack on HTTP methods (e.g., white-listed and blacklisted HTTP methods for each operation). HTTP verb tempering attack
RBAC model support for many-to-many user to role assignment	No	
RBAC model support role hierarchy	No	
RBAC model support for Separation of Duties in user	No	
Policy Decision Point (PDP): Component which evaluates and issues authorization decision	PDP – not centralized, each service is responsible for authorization decision based on user role and service specific policies.	
Policy Administration Point (PAP): administration point of policy	PAP – Not centralized, policies are written as a rule in each service	
Policy Information Point (PIP): Component which stores policy rules and authorization attributes	PIP – Service specific policy store. User, tenant, roles comes from centralized Keystone server.	
Policy Enforcement Point (PEP): Component, which intercepts user's access request to a resource and enforces PDP's decision	PEP – service specific PEP	
ABAC authorization with attribute: A) Subject (e.g. User Id, Role) B) Action (e.g. write, read, execute) C) Resource (e.g. Virtual, machine, user account)	ABAC access control is not supported by default	

D) Environment (time of the day, ay of the week, authentication strength, number of VM allowed to create) Support for policy delegation	 Not by default	
User Services	**Keystone**	**Threats**
User/Tenant management (add, delete, update, list) Authentication credentials (e.g. passwords, PINs, initial secrets, authentication tokens) management Authorization information (e.g. roles, attributes, policy rules) Audit users and their privileges, e.g. delete unauthorized users and minimize unnecessary privileges Automated provisioning, de-provisioning and synchronization of users and user accounts across cloud components and applications Support Bulk (of user and user profile) provisioning	Few manageable attributes (through API) are supported in the default user management system Supported It is possible to set roles for each user and policy rules for each user/role. The rule expression lacks features. Auditing feature is limited This requires additional tools or manual task to migrate accounts from one component to another Additional tools required Not in default setup	1. Security mis-configuration - User, role, attribute management, set rules for non authorized access to resources - who is allowed to perform user management 2. Threats from credential control – weak password or secrets setup by user 3. Threats from attack detection and forensic analysis weakness due to inappropriate amount of audit log. 4. Injection attack (e.g., invalid input) during User data migration and provisioning from a third party store or bulk migration. 5. Correctness of user data/attributes

Support delegated administration, i.e. to provide the ability to delegate administration and operations on a fine grained level User self-service portal e.g. for self-registration, password reset, profile modification, etc. Support for workflow feature that allows tasks to be controlled by a workflow process, e.g. automatically routes requests to the proper approvers. Support for standard APIs based on industry standards	Supported in the default setup Not in the default setup REST based OS API for user management (not same as SCIM or other standards)	
Identity Service	**Keystone**	**Threats**
Federated identities /SSO Password policy enforcement Privileged account management	At the moment, federation through backend integration is supported. No front end federation e.g., SAML / OpenID supported by default. Not in default setup Role based access control. Granular control is missing for privileged accounts as well as for normal accounts	1. Threats from trustworthiness of subject, identity provider and service provider in federation 2. Identity theft and misuse of identity information in Federated environment 3. Compromise of user/tenant account due to weakness in credentials 4. Internal threats - Threats from privilege account
Audit and Business logic	**Keystone**	**Threats**
Log	Log is performed using Python Logger module. The configuration file can define where to place log output. The log feature	1. Threats due to the due to lack of log data (e.g., attack detection and forensic analysis weakness)

Sensitive data log	is basic. No special feature to log sensitive data	2. Threats from non-repudiation of log 3. Threats from sensitive data log (data inference)
Monitoring	By default, monitoring through stats_monitoring filter. It is a WSGI application which provides statics related to WS identity API (e.g., which API is called, how many times, from which machine)	1. Threats from lack of monitoring data and tool
Business Intelligence	**Keystone**	**Threats**
Catalogue	A catalogue of URLs for OpenStack services	Mis-configured /malicious catalogues can range attack from security token/session sniffing, MiTM, and complete control over session. Mis-configured catalogue can cause service unavailability to the user
Rate Limiting	Through rate limiter middleware	Mis-configured rate limiter can cause service unavailability
Support for Lawful interception	No	An unclear process of lawful interception could expose threats to the system

Repositories

Features	**Keystone**	**Threats**
Identity Repositories: Support for a range of identity backend. Sensitive data protection Credential (e.g., password storage, token)	Currently, supported backend File, KVS, SQL and LDAP. No mechanism Encrypted password is stored in the DB	1. Threats from insecure interface/communication between Logic layer and Identity backend (e.g., lack of encrypted communication, access control in the repository) 2. Threats from invalid data arriving from logic layer to the repository (input validation threats) 3. Unauthorized access to data store 4. Lack of protection for storing sensitive data. 5. Lack of protection to store credentials and threats in credential life cycle management

		(e.g., password salting for password, token deletion) 6. Attacks on token state and storage in data store
Policy Store:	Keystone supports JSON based policy language. Policies are written as a rule. Each service stores and defines its own policy.	1. Unauthorized access to policy store 2. Integrity and non-repudiation (identify actor who writes policy) threats on policy data.

3. Conclusion

Keystone, the centralized identity and access management system for OpenStack, provides basic functionalities for authentication and authorization. In our analysis, we have found weaknesses / lack of feature support in many areas including access control mechanisms (e.g., lack of features in RBAC, no ABAC support), authentication attribute provisioning, policy provisioning mechanism, and auditing mechanisms. The threat analysis has identified possible threats against interfaces, components and subcomponents of Keystone. The document itself is generated as part of a quick study not meant to be a complete analysis of Keystone.

References

[openstack] http://www.openstack.org/software/

[SecaaS] Cloud Security Alliance (CSA), SecaaS Implementation Guidance. Category 1// Identity and Access Management, September 2012.

[CSA-Chap12] Cloud Security Alliance (CSA), Chapter 12, Security Guidance for Critical Areas of Focus in Cloud Computing V3.0, 2011

[RiskMatrix] Perterson Gunnar, Introduction to identity management risk metrics, IEEE Security & Privacy, 2006

[OWSAP]https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

[ComonFlaws] Brad Hill, Common Flaws of Distributed Identity and Authentication Systems

[CSA9Threats] The Notorious 9, CSA

[OWSAP] OWSAP TOP 10, https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[CSAGUIDIE] Security Guidance for critical areas of focus in Cloud computing V3.0

[IMPP] http://www.commoncriteriaportal.org/files/ppfiles/pp0024b.pdf