Risk Assessment and Mitigation for Access Control mechanisms in OpenStack.

Adam Young

Red Hat Identity Management Team

Jul 15, 2015

Introduction

Risk Assessment and Mitigation for Access Control mechanisms in OpenStack.

Agenda

1. Risk Assessment
   - Use of Tokens
2. Design Considerations
   - Roles and Scopes
   - Use Cases
   - Constraints
3. Dynamic Policy
   - Mission
   - Overall Plan
   - Policy Distribution
   - Roles

Agenda

Threats

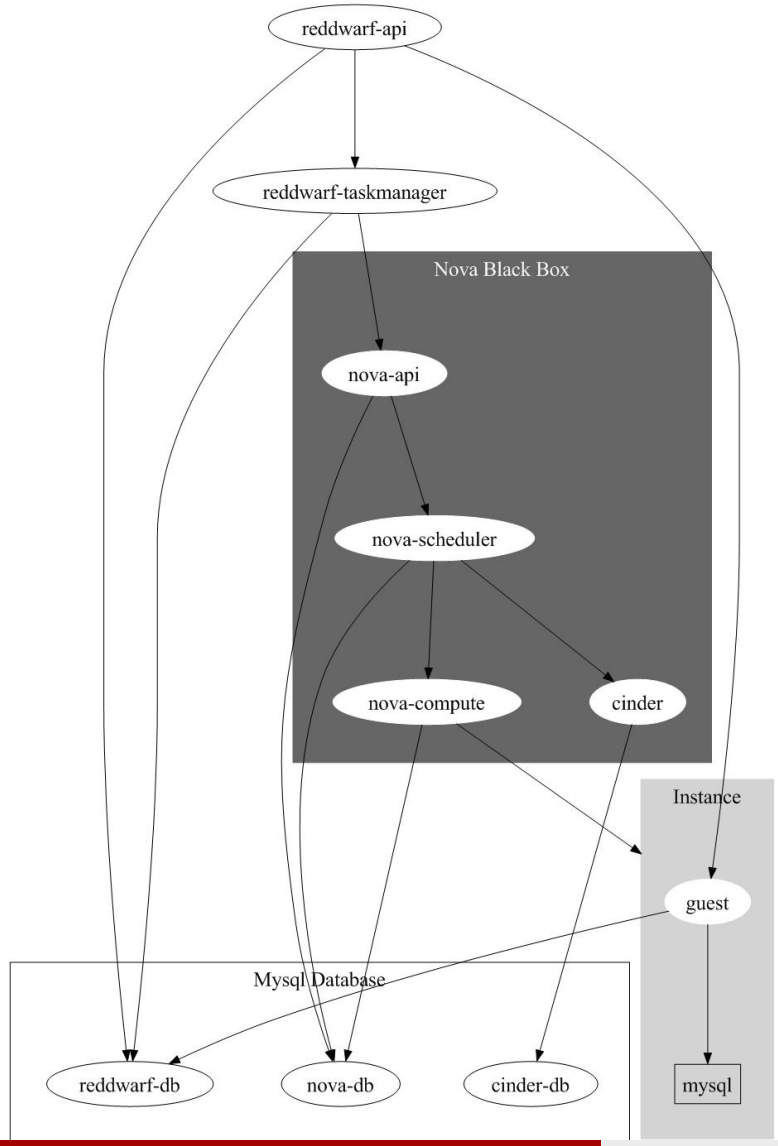
- Hypervisor Compromise: Tokens
  - validates the token
  - Nova Compute Runs on the Hypervisor
  - Tokens included included in requests
  - Rogue VM Harvest Tokens
- Hypervisor Compromise: Trusts
  - Allow for a long term delegation
  - A rogue trust would by-pass the token expiry
  - Without Trusts, passwords will be copied around, which is even worse.
  - Easy to identify but you have to look
- Authenticate via Token when fetching a token
  - Risk
    - Any scope can be converted to another scope
    - Expiration is not extended
  - Why:
    - Web UI does not cache password
  - Mitigate
    - Only allow unscoped to scoped transitions
    - Requires a call to explicitly request a scoped token

Agenda

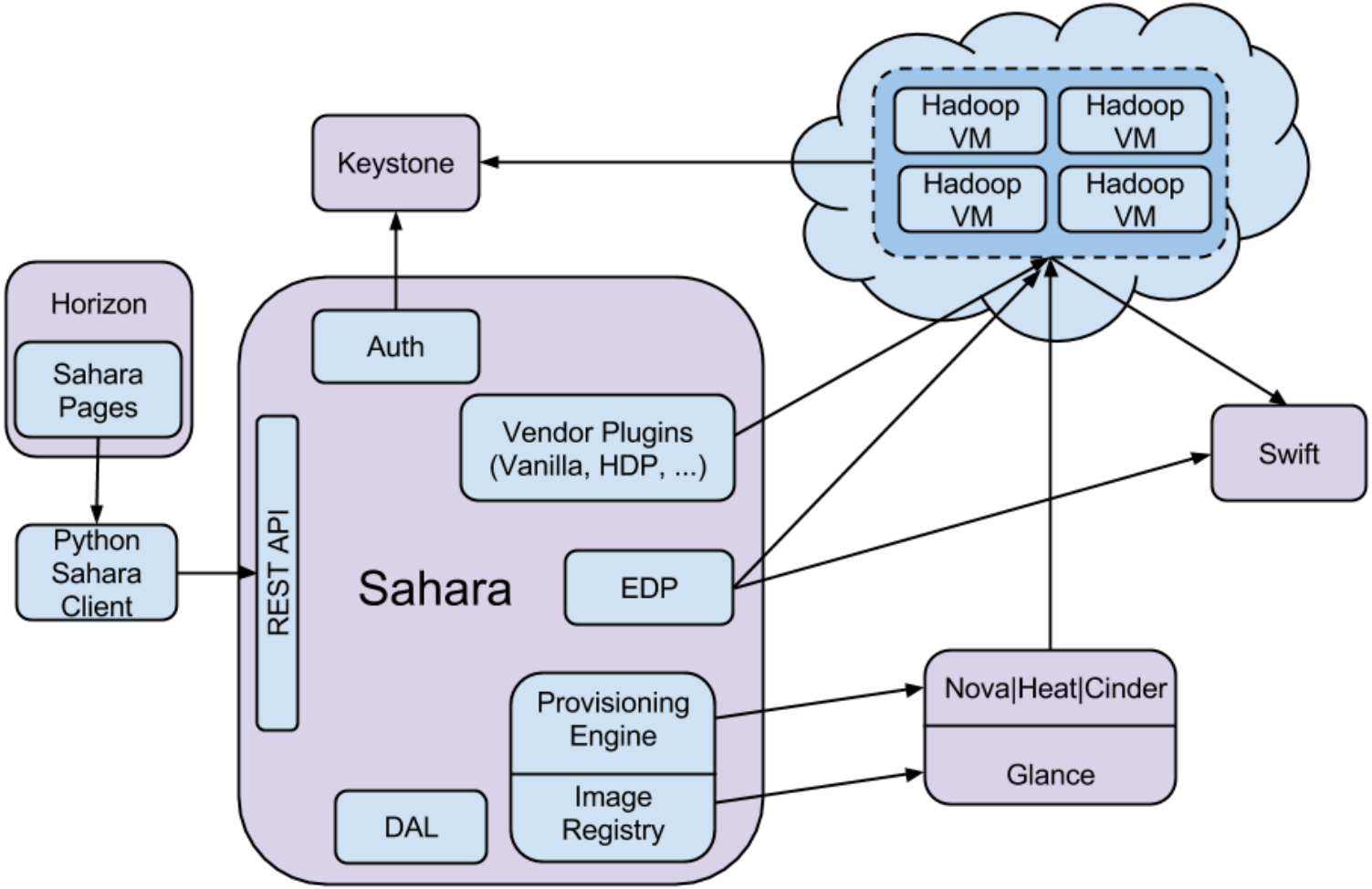
Big Tent Services that User Tokens

- Barbican
- Ceilometer
- Cinder
- Congress
- Cue
- Designate
- Glance
- Heat
- Horizon
- Ironic
- Keystone
- Magneto DB
- Magnum
- Manila
- Mistral
- Murano
- Neutron
- Nova
- Sahara
- Search light
- Solum
- Swift
- TripleO
- Trove
- Zaqar

Trove Uses Tokens

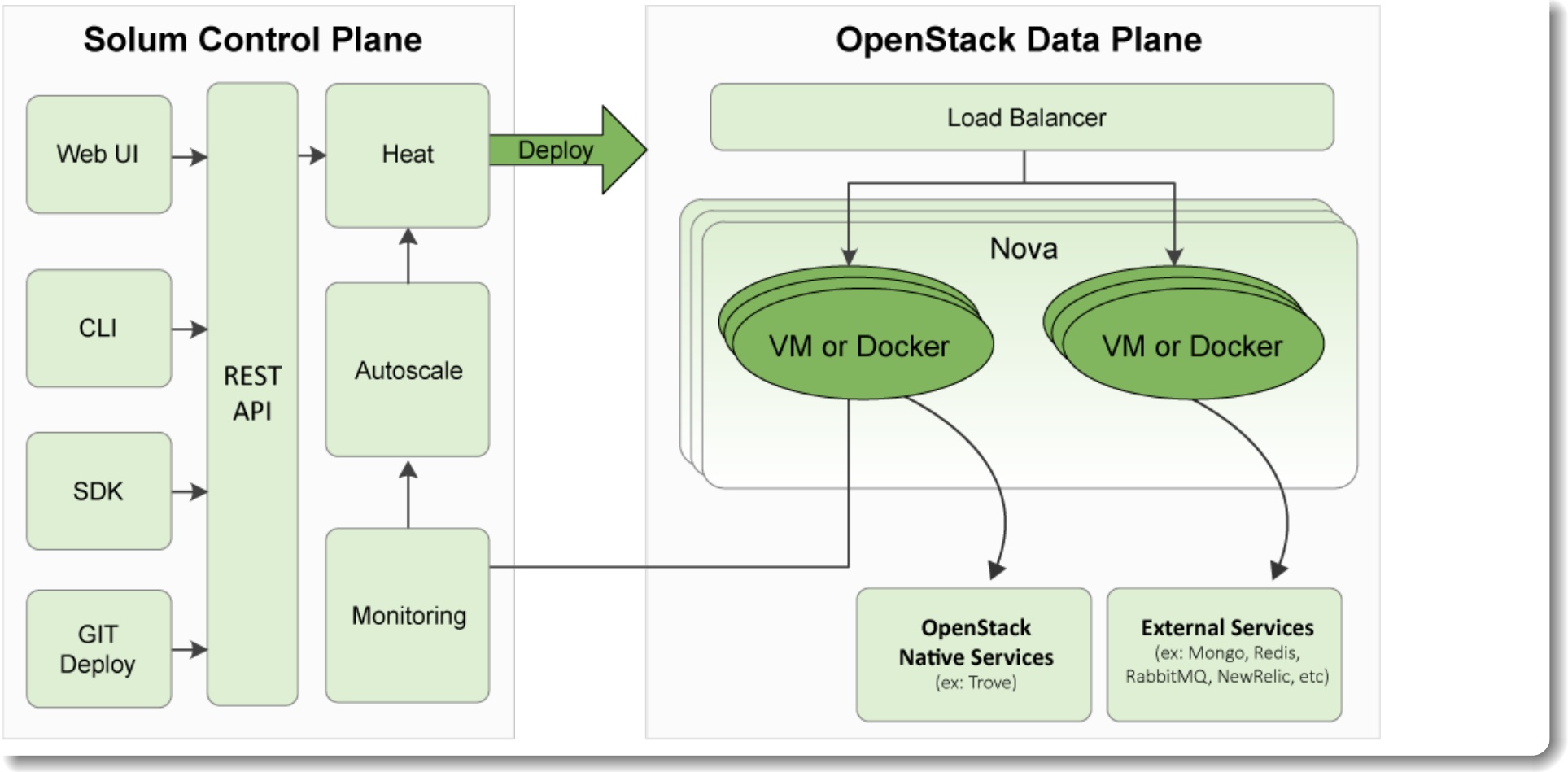


Sahara Uses



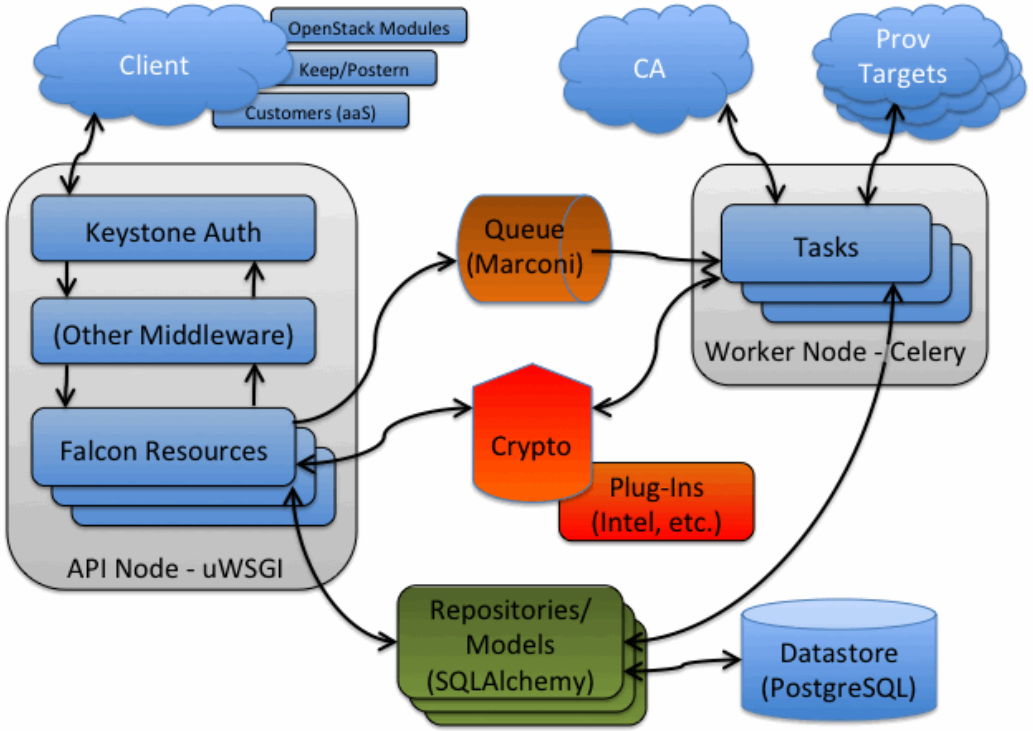
http://docs.openstack.org/developer/sahara/architecture.html

Solum and Heat Use Tokens



http://solum.io/img/chart.png

Barbican Uses Tokens

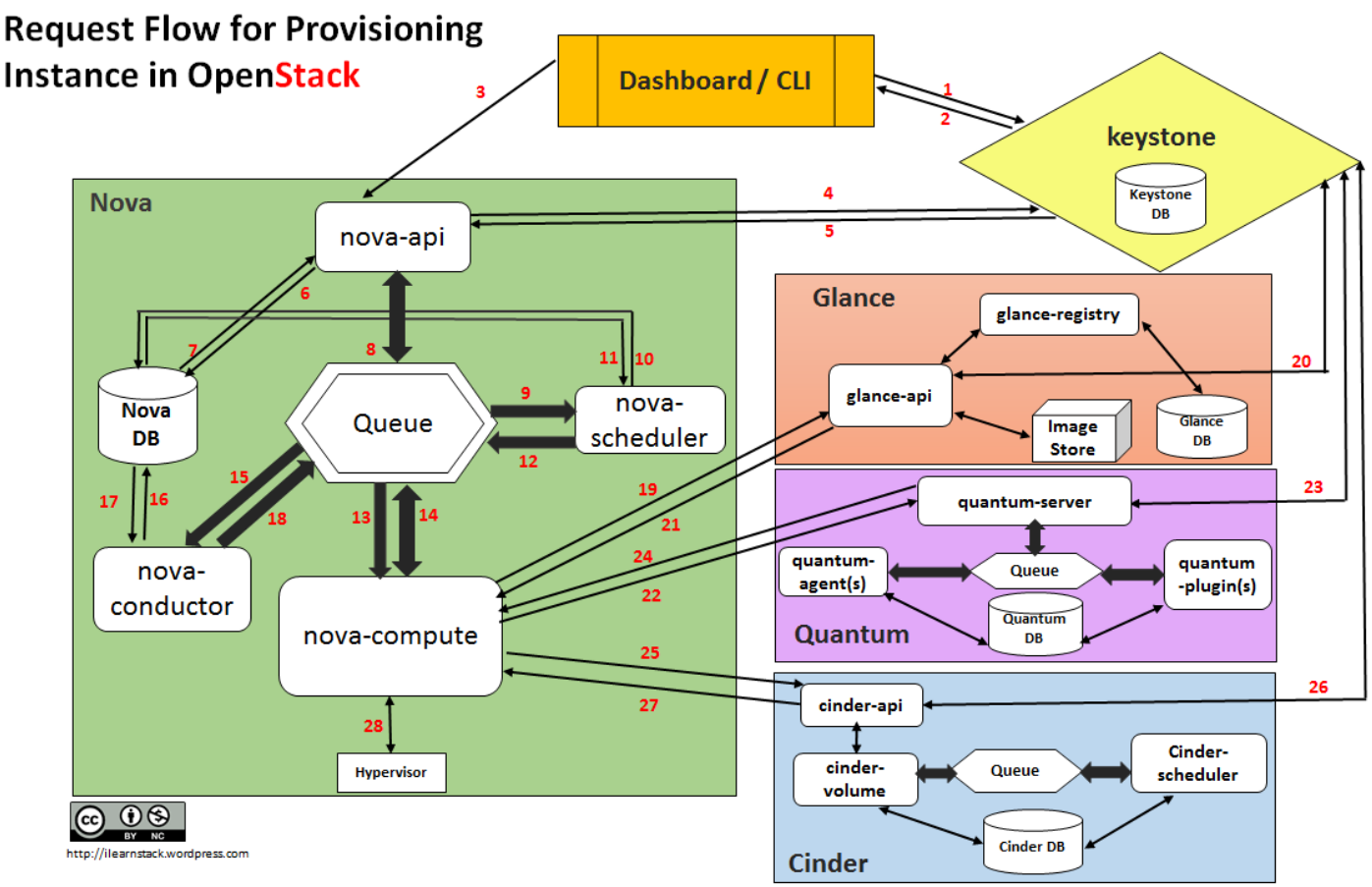


https://github.com/cloudkeep/barbican/wiki/Architecture

Why Are Tokens on Hypervisor

- Boot and Teardown
  - Barbican, Glance, Cinder, Neutron
- Attach (Cinder)
- Periodic Refresh (Neutron)
- Snapshot (Glance)

Why Are Tokens on Hypervisor



http://ilearnstack.com/2013/04/26/

Other Users

Any third party application that drives OpenStack components uses tokens.

Agenda

Status Quo

The current policy enforcement varies greatly between services

Agenda

Scope of a Token

- Most common role check is *Admin* with no scope
- Most policy does not Check Roles

Role vs Scope

Every Rule should have Scope section and Role Section

- Matching Scope should not be customized
  - Engineering decision
  - based on the object structure
- Role Assigned to API should be customizable
  - Chose the roles appropriate to the organization
  - Default to *Admin* if not specified for target

Global Admin

- Most stock policy files have an account to fix things
  - Nova has it in code: nova/tree/nova/policy.py
  - return creds['is_admin'] == self.expected
  - "os_compute_api:os-lock-server:unlock:unlock_override": "rule:admin_api",
  - "admin_api": "is_admin:True",
- Keystone has ADMIN_TOKEN/OS_SERVICE_TOKEN
- Check Role:admin without checking Scope

Where is the Scope

- project_id in the URL
  - Nova list servers
  - http://hostname:8774/v2/¡project_id¿/servers/detail
  - Trove list databases for instance
  - https://hostname/v1.0/¡project_id¿/instances/¡instance_id¿/databases
  - Keystone grant role to user on project
  - PUT http://hostname:35357/projects/¡project_id¿/users/¡user_id¿/roles/¡role_i
  - must confirm with database record
- Use the scope from the token
  - Glance image list
  - http://hostname/v1/images
  - Won't work with a global admin

Where is the Scope (Continued)

- Fetch object from Database
  - Ceilometer Rules mostly have
  - "context_is_project": "project_id:
  - Keystone add user to group (Domain scoped)
  - PUT /groups/¡group_id¿/users/¡user_id¿
- Object is not scoped to a project
  - Keystone User owns Credentials and Trusts
  - Domains own projects
  - Barbican user owns secrets

Neutron Policy Rules

- "shared_*":
  - "field:networks:shared=True",
  - "field:firewalls:shared=True",
  - "field:firewall_policies:shared=True",
  - "field:subnetpools:shared=True",
  - "field:address_scopes:shared=True",
- "get_subnetpool":
  - "rule:admin_or_owner or rule:shared_subnetpools",
- (unscoped) admin role check OR access to the API is global
- Limited to read only.

What is a Role?

- A label for a set of permissions across multiple services
- But we have no way to share role information between services or endpoints with stock policy files

Problems with Role Assignments today

- If I can assign one role, I can assign any role
- Admin role is not scoped to a project
- If I can assign admin, I am admin
- Need to restrict ability to assignment only the roles/scope a user has assigned to them.

Agenda

Scope of a Token

- Most common role check is *Admin* with no scope
- Most policy does not Check Roles
- *_member_* role was added by Keystone to deal with migration from Tenant owning users
- A scoped token used on any service is valid for any *_member_* command anywhere else in the OpenStack Deployment

Member Use Cases

- Boot
  - Heat
  - Sahara
  - Trove
- Write Data
  - Glance
  - Trove
- Read Only
  - Ceilometer
  - Searchlight
- Create Trust
  - Heat
  - Solum

Tenancy Administration Use Cases

- Nova
  - Quotas
  - Security Groups
  - Key Pairs
- Cinder
  - Quotas
- Glance
  - images
- Keystone
  - create (sub)project
  - Assignments
  - Federation Mappings
- Neutron
  - Networks
  - Subnets
  - Extensions

Service Administration Use Cases

- Nova
  - Hypervisors
  - Floating ip associate
  - Cells
- Keystone
  - Roles
  - Service Catalog
  - Policy
  - Identity Providers

Agenda

Things we can't break

- Deployments expect the current stock policy files to work
- Global Admin to the rescue
- Where is the scope?

Agenda

Agenda

1. Risk Assessment
   - Use of Tokens
2. Design Considerations
   - Roles and Scopes
   - Use Cases
   - Constraints

3 Dynamic Policy

- Mission
- Overall Plan
- Policy Distribution
- Roles

Goal

- Minimize the damage that can be done with a stolen token:
- A stolen token can only perform operations within the same class of use cases.

Agenda

Original Dynamic Policy Design

- Graduate oslo policy to a library
- Common code to enforce policy on a token
- Fetch policy from Keystone
- Provide a unified policy file
- Database schema to hold policy rules.
- Hierarchical roles
- Generate rules from hierarchical roles
- Break member up into smaller roles.
- Use the smaller roles in the Policy targets.
- Users can only assign a role that they themselves have on the designated scope.

Dynamic Policy Successes

- Graduate oslo policy to a library complete
- Fetch policy from Keystone no targeted at URLs, not Endpoint Ids
- Kent has Proof of Concept for Database

Dynamic Policy Adjustments

- Unified Policy File may conflict dynamic natures of microversions
- Hierarchical Roles now includes role namespacing
- Kent design is better for querying

Agenda

Why not Use Configuration Management System

- Can be done, but has drawbacks
- Removes Keystone's ability to determine which policy file to fetch
- Most installations treat Policy as static, and will not update even if Keystone updates
- Extracts Policy out of the application management flow
- In the future, project specifc policy will require futher integration

Fetching the right policy file

- policy to endpoint mapping has been defined for a while
- to use it an endpoint needs to know its own id.
- Fallback from most to least explicit:
  1. explicitly define endpoint_id in the config tile
  2. explicitly define URL in the config file, map to ID via service catalog
  3. Use Incoming URL on first request to match the URL in the Service catalog.
- config/request_url.startswith(endpoint.url)

Why fetch by Endpoint

- May not get a unified policy file
- Need to server separate policy for each service
- If customized for a specific server, we need to serve the policy for that server
- While this might be multiple *endpoints*we can treat all endpoints with the same URL as one endpoint for policy reasons
- CMS can calculate the URL prior to registering endpoint
- With endpoint_id: register, retreive ID, inject into config, restart server
- For most cases, middelware could deduce the URL from a request.
- Why not Free form label?
  - Might be useful in the future
  - Would still require mapping to the endpoint

Where do we fetch?

- Oslo.policy is not specific to RBAC
- Fetching from Keystone is not specific to Oslo.policy
- After token validation
- Before Policy is called
- Using Policy for Endpoint binding in Middleware

auth_token vs policy middleware

- auth_token is purely a convenience,
- does not require modifying pipelines
- Separate middleware would be cleaner
- Compromise: auth_token as a series of separateable middleware modules

Enforce policy from middleware

- enforce policy on URLs, not random strings inside the code
- role/scope split
  - only the role portion
  - this makes the most sense for customization
- Scope would have to be enforced after database fetch for many resources
- Middleware can return HTTP 401 with extra knowledge of what role is required
  - Bends the standard, but does not break it

Policy Endpoint Extension

- Existing code
- Associate a policy (by id) with an entry in the service catalog
- Resolved from most specific
  1. Endpoint
  2. Service
  3. Region
  4. Default
- But what do we use for default?

Single Unified Policy File

- **Not stock policy**
  - Kept in its own repo
  - only deployed deliberately
- Common section for common rules
- Specify lowest role necessary
- Single Policy Header
- Move toward a single File
- Hierarchical Roles
- Break member up into smaller roles
- Change the rules for specific API policy enforcement points to know about the new roles.

Agenda

Implied Roles

- Hierarchical is overloaded term but what NIST uses
- Implied roles:
  - Explicit-role IMPLIES Implicit-Role
  - Impicit Roles can IMPLY other Implicit-Roles
  - Admin IMPLIES Member
  - Member IMPLIES both Read-Only and Writeable
- A user is assigned a role at the top of the hierarchy
- A target specifies a role as low on the hierarchy as possible
- A user can then delegate an implicit role instead of explicit

Global Admin

- Better Global Admin
  - Global admin means user is in admin role on admin domain
  - In order to test that outside of Keystone server, we need to be able to query Keystone config remotely Added Benefit: allows a way to query default domain as well
- Alternative to Global Admin
  - Provide a means for a specific user to get a token scoped to any project with any role
  - Is it really any worse?

Role Namespaces

- Roles are lables.
- Currently, all roles are global
- Roles hould map to the services
  - *Admin* versus *Compute.admin, network.admin*
- Implied roles can still work across namespaces
  - Compute.boot implies network.reader
- Can expand the role of Keystone beyond Undercloud:
  - *Wordpress.editor*
  - *Gerrit.approver* implies *Git.committer*
- Segregated by Project Scope as well
  - *Gerrit.approver* on *Openstack/Keystone* vs
  - *Gerrit.approver* on *Openstack/Nova*

Scope All Entities

- Keystone Service level entities be owned by the Admin Domain
  - Services
  - Endpoint
  - Regions
  - Roles
- Lends itself to better delegation in the future
  - Scope Roles under services to Namespace

How Do We Know What Permissions to Delegate?

- Changes are rare enough that we can figure out statically
- When doing a Nova Boot, the Nova client can be smart enough to get the right token.
- All those things that call Nova boot need to be smart enought, too
- As things get more complicated, we can run Tempest against policy in permissive mode and see what would have gotten rejected.
  - This approach works for SELinux and AppArmor

Requesting a Token with a subset of roles

- Request only the role required for the task at hand
- "I need a token for booting a VM"
- Ideally, a token would have only a single role
  - Added Benefit: Fernet-style token implementations smaller

Long lived delegations

- shorter the token lifespan = smaller attack surface
- No risk of timing out on long uploads etc
- Nova could request a read only token for other services during boot.
- Trusts and OAUTH1 should use the same logic
  - OAUTH Consumer becomes user in a custom domain

Unified Delegation

- Role assignment Follows Trust model except we add concept of *Position* or permanent role assignment
  - A user is assigned to a *position*
  - All permanent role assigments happen from *position* to *position*
  - A user can only delegate to a *position* a role that they themself are assigned,either explicitly or implicitly
  - If a *position* goes unfilled, role assignments previously made from that *position* are:
    - inactive until it is filled again, or
    - are approved by a higher authority

Task Ordering

- Each commit should show value stand alone
- Front load the changes that must be approved by the other projects
- Start with dynamic fetch, manual management
- Continue to refine the plan

Questions

Questions?