openstack.

29 April 2019

You can’t make a (Denver) omelette without breaking eggs

Using OpenStack policies for great good

Brian Rosmaita, irc: rosmaita

redhat.

Brian Rosmaita

Senior Software Engineer

Making a Denver omelette ...

Sidewalk marker between 15th and 16th Streets on California Street in Denver

openstack.

29 April 2019

Some remarks about policies

With special reference to Cinder and Glance

Brian Rosmaita, irc: rosmaita

Some remarks about policies

- This presentation is mostly me talking
- You have a great opportunity this afternoon to try this stuff out and get expert help for your questions
  - Access Control Policy Hands On Lab
    - 3:50pm-5:20pm, Meeting Room Level - 4D
    - Harry Rybacki, Adam Young, and Nathan Kinder
    - The focus is on learning and practicing techniques to customize an access control policy for your cloud
    - Should be fun -- you get a pre-configured VM to work with, and away you go

Some remarks about policies

- A lot of this talk is drawn from a HowTo guide I wrote for Cinder during the Stein development cycle
  - https://cinder.openstack.org
    - > (heading) For Operators
    - > (heading) Reference
    - > (link) Cinder Service Configuration
    - > (link) Policy configuration HowTo

Some remarks about policies

- The intended audience for this talk is Operators
  - You need to be in a position to configure an OpenStack cloud to make use of this info
  - Of course, anyone who cranks up Devstack becomes the operator of an OpenStack cloud
- I am not an operator
- The premise of this talk is that if you understand how developers think of users when writing code, it will be much easier for you to configure your own custom policies

Some remarks about policies

- This talk is in the “Security” track
  - You can configure policies to make your cloud more secure
  - It is *remarkably easy* to use policies to make your cloud ***insecure***
  - This is some dangerous stuff we’ll be talking about
- Quick assessment
  - DeMorgan’s Laws
    - “not (A and B)” is logically equivalent to “not A or not B”
    - “not (A or B)” is logically equivalent to “not A and not B”
  - If you don’t remember or haven’t heard of those, be extra careful
  - If you *are* familiar with those, also be extra careful

Being extra careful

- I encourage you to attend the hands-on lab at 3:50pm today where you can experiment with policy changes in a safe environment
- If you are going to make policy configuration changes in a cloud, you ***must*** have a *test plan* in place
  - You can't expect to verify rule logic by simply looking them over
  - The Cinder policy file has more than 75 policy targets (actually, a *lot* more)
    - Did you get 100% correct on all your propositional logic tests in school?
    - If you got less than 100%, was a consequence that a cloud user could, for example, delete other users' resources?

Outline

- Some background and vocabulary
- The Administrative Context
- A very, very, extremely short history of OpenStack policies
- How to find out what the current policies are
- How to modify policies
- A few things about Glance and policies
- Example: Configuring a read-only administrator

Policies

- What they do
  - Allow you to control how users and administrators interact with the various OpenStack services available in your cloud
- How they do this
  - A service defines a set of “policy targets” that the service will respect
  - Each target can be associated with a “policy rule” that must be satisfied when the service enforces the policy
    - Each policy rule can make reference to “roles” defined in Keystone
    - You can create roles and assign them to users with Keystone

Policies

- Here's a Cinder policy

```
"message:get_all": "rule:admin_or_owner"
```

- The general format is
  - The policy target name on the left side, in quotes
  - A colon (:)
  - The policy rule name on the right side, in quotes

Policies

- The **policy targets** are *service-specific*
  - Each service defines the set of targets it will recognize
- The **policy rules** are specific to the policy file you use to configure the policies for that service
  - The targets that appear in the file are defined by the service
  - The rule names that appear in the file are defined by *you* in that file

Vocabulary

- Project
  - An administrative grouping of users into a unit that can own cloud resources
    - This is what used to be called a “tenant”
- Service
  - An OpenStack component that users interact with through an API it provides
    - “Cinder” provides the Block Storage API
    - “Glance” provides the Image API
  - “Cinder” and “Glance” are sometimes referred to as “OpenStack projects”, but I’ll always refer to them as “services” (except when I forget)

User Model

- Developers write code expecting that they'll interact with two kinds of users
  - End users
    - Consume resources and (hopefully) pay the bills
    - Are restricted to acting within a specific project
    - Cannot perform operations on resources not owned by the project (or projects) they are in

User Model

- Developers write code expecting that they'll interact with two kinds of users
  - Administrative users
    - "admins"
    - Can view all resources controlled by the service
      - … and can perform most operations on them
    - Have access to operations that cannot be performed by end users
    - May be able to view resource properties that cannot be seen by end users

Administrative Context

- OpenStack APIs do not in general have a special “admin” API
  - An admin or an end user who want to do a volume-list or a volume-show make the *same* calls
    - … but they will see different responses
- The technical way to speak about this is that one call is being made in an *administrative context* and the other is not
- A user acting in an administrative context can do admin-type stuff
- A user *not* acting in an administrative context can only do normal end-user-type stuff

What an operator can do with policies

- An operator can define what users are granted the privilege of acting in an administrative context
- An operator can specify for specific policy targets which users can perform those actions
- In general
  - An operator can define *who* can make calls in an administrative context
    - You do this with Keystone and the policy configuration file
  - An operator cannot affect *what* can be done in an administrative context
    - This is determined when the code is written

What an operator can do with policies

- Example
  - In Glance the boundaries between projects are strictly enforced
    - Only an admin can view resources across projects
    - There is no way to grant an end-user the ability to see into another project using policies
  - People have asked in IRC how to configure Glance policies so that an end user can only do CRUD on resources in their own project
    - Nothing to configure, that's how it works out of the box!

Key takeaways

- A user can do “admin-type” stuff only if they’re acting in an *administrative context*
  - This includes simply *reading* stuff that only an admin can see
- There are checks throughout the code (not simply at the API layer) to determine whether or not processing is happening in an administrative context, and this affects the result

Key takeaways

- Example: Glance has a policy target named “get_images” that is defined like this in the policy file:

```
"get_images": ""
```

- The empty rule means *anyone* can make the image-list API call. But what’s in the response will differ based on whether the call is made in an administrative context or not
  - You can restrict *who* can make the call, but not what happens when the call is made

Key takeaways

- A lot of operators are interested in having an “observer” or “read-only” administrator who can conduct thorough audits but not modify anything
- To create a read-only administrator, you must allow that person to make calls in an administrative context
  - You can’t simply take an ordinary user and add one or two admin-type powers to that user
  - You must allow such a person to *be* an administrator, and then use policies to restrict them to only performing read-only calls in the policy file

Some remarks about policies

- That’s the theory part of the talk -- now let’s get down to some practical matters
  - what a policy file looks like
  - where you find a policy file
  - Glance is a bit special in how it uses policies, we’ll talk about that
  - Cinder is pretty normal, it’s a good exemplar for the other services
  - Walkthrough of a practical example
    - Configuring a read-only administrator

Extremely short history of OpenStack policies

- The oslo.policy library appeared in Kilo
  - Makes policy configuration more uniform across services
- The original policy files were written in JSON
- YAML support introduced in oslo.policy 1.10.0 (Newton time frame)
  - Wasn't announced until 1.15.0 (Ocata time frame)
  - Key advantage of YAML: comments!
- Queens: the "policies in code" community initiative
  - Makes it possible to run a service without a policy file (sensible defaults are defined in the code)
  - Glance did not participate
    - Glance needs the policy file shipped with Glance to get sensible defaults

No policy file? How do I know the defaults?

- You can generate a policy.yaml file from a checked-out Cinder code repository

```
[master] wha'ppen? tox -e genpolicy

[master] wha'ppen? head etc/cinder/policy.yaml.sample
# Decides what is required for the 'is_admin:True' check to succeed.
#"context_is_admin": "role:admin"

# Create attachment.
# POST  /attachments
#"volume:attachment_create": ""

# Update attachment.
# PUT  /attachments/{attachment_id}
#"volume:attachment_update": "rule:admin_or_owner"
```

Configuring a policy file

- Best practice is to *explicitly* tell the service what policy file it should use
  - For most services, the policy file is optional since Queens, so if it's missing, you won't notice!
- In the service configuration file:

```
[oslo_policy]
policy_file = /path/to/the/policy.yaml
```

The Glance policy file

- The file that ships with Glance is JSON
- Glance uses oslo.policy, and can understand a YAML file
- To convert:

1. `cp policy.json policy.yaml`
2. Remove the ‘{‘, ‘}’
3. Remove all trailing commas
4. The metadefs policies need a space after the colon

```
"add_metadef_object": ""
```

5. Add comments as you change things
6. Tell glance-api.conf that you are using policy.yaml
7. Profit!

Glance warning

- Like all the other services, you can define a rule for `context_is_admin` in the Glance policy file to define the administrative context
- BUT … Glance also has a config option named `admin_role` that overrides the context defined in the policy file and elevates anyone with that role to acting in an administrative context
  - The default value is `admin` but I advise changing this to some role that no one will ever have
    - Keep all administrative context configuration in *one* place

```
[DEFAULT]
admin_role = __not_a_real_role_<uuid>_not_a_real_role__
```

One more Glance thing ...

- Glance has “property protections”, which allow you to put CRUD permissions on individual custom image properties
  - You have the option of defining policy rules for these
  - They must be defined in the *same* policy file that you’re using for your “regular” Glance policies
  - Best practice is to use dedicated rules for this

Configuring property protections

- Part 1: In your glance-api.conf file:

```
[DEFAULT]
property_protection_file = glance.pp
property_protection_rule_format = policies
```

Configuring property protections

- Part 2: In your glance policy file:

```
# used for property protections
"pp:everyone": "@"
"pp:members_only": "role:member and not role:admin"
"pp:special": "role:special_role or role:admin"
```

Configuring property protections

- Part 3: In your property protections file:

```
[^member_.*]
create = pp:members_only
read = pp:members_only
update = pp:members_only
delete = pp:members_only

[.*]
create = pp:everyone
read = pp:everyone
update = pp:everyone
delete = pp:everyone
```

Other than that ...

- Glance is completely normal

Practical application

- Configuring a read-only administrator
  - Even a read-only administrator must be an administrator
    - An administrator is a superuser for a service
    - You have to be really careful!
  - Strategy: introduce a new role, make that role recognized in the administrative context, and then remove any ability for that role to do non-read-only actions
    - Metaphorically: open the floodgates, and then plug up the holes one by one

Step One: Testing

- But we haven't done anything yet!
- What needs to be tested
  - A read-only administrator can see the appropriate stuff
    - You need to define "stuff"
  - A read-only administrator *cannot* do inappropriate stuff
    - Ditto
  - A "regular" administrator's functionality has not changed
    - You need to know what the baseline is
  - A "regular" user's functionality has not changed
    - Ditto
  - Same for any other types of user you have configured via policies

Step Two: Create a new role

- Why a new role?
  - A person with this role will have full administrative powers for any functions we don't explicitly deny -- so if we miss any when we start plugging up the holes, we could have a rogue admin on the loose
  - We want to be able to easily track who the role's been assigned to
  - For Cinder, could use **`cinder:reader-admin`**
    - 'cinder' for the service
    - 'reader' for what this role is intended to do
    - 'admin' so we don't forget that this is serious business

Small digression about roles

- Beginning with Rocky, Keystone creates three roles out of the box:
  - **`member`**
  - **`reader`**
  - **`admin`**
- Beginning with Stein, the Identity Service is configured so that users with the **`reader`** role have read-only access to the Identity API
- This applies *only* to the Identity Service. Having the **`reader`** role doesn't affect any other services unless they have been explicitly configured to deal with it

Step Three: Recognize the new role

- We want someone with this role to operate in the administrative context, so we modify the policy file:

```
"context_is_admin": "role:admin"  or "role:cinder:reader-admin"
```

Step Four: New policy rule

- We need a rule that we can use to plug up the holes we just created:

```
# Default rule for most Admin APIs.  (This rule already exists)
#"admin_api": "is_admin:True or (role:admin and is_admin_project:True)"

# Exclude readers.
"strict_admin_api": "not role:cinder:reader-admin and rule:admin-api"
```

Step Five: Plug up the holes

- Find all occurrences of `rule:admin_api` and decide whether they should be loose (don't change) or strict (use the new rule)
- Before:

```
"volume_extension:services:index": "rule:admin_api"
"volume_extension:quotas:delete": "rule:admin_api"
```

- After:

```
"volume_extension:services:index": "rule:admin_api"
"volume_extension:quotas:delete": "rule:strict_admin_api"
```

Step Six: Find more holes

- Take a look at this default rule:

```
# Delete volume.
# DELETE  /volumes/{volume_id}
#"volume:delete": "rule:admin_or_owner"
```

- It will help to see what the `admin_or_owner` rule does:

```
# Default rule for most non-Admin APIs.
#"admin_or_owner": "is_admin:True or ... or project_id:%(project_id)s"
```

openstack

Step Seven: Another new policy rule

- We need a rule that can plug up these “non-admin” holes:

```
# Default rule for most non-Admin APIs.
#"admin_or_owner": "is_admin:True or ... or project_id:%(project_id)s"

# Exclude readers.
"strict_admin_or_owner": "not role:cinder:reader-admin and rule:admin_or_owner"
```

Step Eight: Plug up the remaining holes

- Find all occurrences of `rule:admin_or_owner` and decide whether they should be loose (don't change) or strict (use the new rule)
- Before:

```
"volume:get": "rule:admin_or_owner"
"volume:delete": "rule:admin_or_owner"
```

- After:

```
"volume:get": "rule:admin_or_owner"
"volume:delete": "rule:strict_admin_or_owner"
```

Step Nine: Testing

- Wait … wasn't that Step One?
- Yes, but now you can use those baselines you ran to make sure you haven't inadvertently affected a "normal" admin or an end user

Access Control Policy Hands On Lab

- This afternoon!
  - 3:50pm-5:20pm
  - Meeting Room Level - 4D

openstack

Q&A

Thank you!

 openstack  @OpenStack  openstack  OpenStackFoundation