Moon : un framework pour la gestion des autorisations

Thomas Duval et Dimitri Darthenay
thomas.duval@orange.com
dimitri.darthenay@orange.com

Orange

Résumé. Il existe de nombreux mécanismes pour sécuriser un accès, une donnée ou un réseau. La plupart de ces mécanismes nécessite de s'identifier et de s'authentifier. À partir de cette étape, le système doit être capable d'autoriser l'utilisateur à effectuer des actions sur le système en fonction de certaines règles prédéfinies. Moon est une plateforme permettant de répondre à des requêtes d'autorisation venant de systèmes tiers (typiquement des gestionnaires d'infrastructures virtuelles). Ce système est flexible ; il permet de concevoir, de configurer et d'utiliser un grand nombre de modèles de politique de sécurité de manière fine. La gestion des politiques est aussi centralisée (si nécessaire) et les politiques de sécurité sont modifiables à la volée. Moon est actuellement disponible pour OpenStack et OpenDaylight mais il est théoriquement disponible pour d'autres plateformes. Ce document décrit le fonctionnement de Moon ainsi que son installation et sa configuration de base.

1 Introduction

Dans le monde de la virtualisation, nous avons l'habitude d'utiliser les mêmes modèles de politiques de sécurité, notamment RBAC (Role Based Access Control). Après identification et authentification, l'utilisateur se voit attribuer des droits sur la plateforme en fonction du ou des rôles qu'il a sur cette plateforme. Ces mécanismes d'autorisation sont, en règle générale, invisibles pour l'utilisateur lambda mais ils sont néanmoins indispensables à la sécurité d'un système d'information.

Bien qu'indispensables, ces mécanismes d'autorisation imposent des contraintes potentielles qui peuvent être préjudiciables à la sécurité de nos systèmes. D'une part, les algorithmes utilisés par ces mécanismes sont statiques, on ne peut pas changer de modèle de politique, la plateforme a son modèle implémenté en dur dans son code et il est impossible d'en changer. D'autre part, dans des cas comme OpenStack, la gestion de ces autorisations est décentralisée. Chaque composant gère ses propres autorisations et rien n'est centralisé. Lorsqu'il faut configurer ces autorisations, cela implique de configurer la politique composant par composant

en espérant ne pas générer d'incompatibilités entre composants et cela peut devenir compliqué. Enfin, même si la politique de sécurité convient, l'administrateur ne peut pas systématiquement modifier la granularité de sa gestion d'autorisation. Par exemple, toujours dans le cas d'OpenStack, le nombre d'actions à prendre en compte dans le système de gestion d'autorisation est trop important pour le faire manuellement ; OpenStack gère plusieurs centaines de commandes différentes par composant. Cette granularité dépend fortement du système, de l'administrateur, ... Dans certains, nous allons vouloir une granularité forte (ie. pouvoir gérer chaque action les unes par rapport aux autres), dans d'autres cas, au contraire, nous voudrons rassembler ses actions en des types bien précis (par exemple les actions de type lecture ou écriture, ...).

Comment faire si je veux utiliser un autre modèle que RBAC ? Et si je veux pouvoir changer dynamiquement ma politique sans avoir à redévelopper un moteur de politique et sans avoir à redémarrer mon système de gestion d'autorisations ? Et si je veux centraliser la gestion de mes autorisations pour certains projets et décentraliser dans d'autres ?

2 Historique

Le projet Moon a été développé initialement pour OpenStack car, selon nos besoins, ce dernier possède quelques limitations :

— le control d'accès s'effectue de manière décentralisée, dans chaque composant OpenStack et indépendamment les uns des autres,
— le control d'accès ne permet d'utiliser que la politique de sécurité RBAC (Role Based Access Control)
— l'administration de ce control d'accès n'est pas aisée dans le sens où, un administrateur doit se connecter sur une console et modifier plusieurs fichiers texte (en d'autre terme, le processus d'administration n'est pas user-centric)

À l'époque de l'initialisation du projet et des premiers développements (2015), nous n'avions trouvé aucune autre application ou librairie capable d'offrir certaines caractéristiques qui nous étaient nécessaires, notamment :

— la centralisation de la gestion du control d'accés
— la capacité de maitriser le modèle de politique de sécurité
— la possibilité pour l'administrateur de gérer par une IHM ce control d'accès

Afin de pouvoir intercepter les requêtes d'autorisation d'OpenStack, il fallait adapter le code d'OpenStack (notamment dans KeystoneMiddleware). Après de nombreuses discussions avec la communauté OpenStack

(et notamment celle de Keystone), nous avons pu mettre en place un "hook" dans la librairie Oslo Policy qui est maintenant pérenne.

Depuis, grâce à notre veille technologique, nous avons découvert plusieurs projets qui offrent certaines possibilités. L'un de ces projets est OPA (Open Policy Agent) [2]. OPA propose d'unifier la gestion des politiques de sécurité sur de nombreuses plateformes dont Kubernetes, SSH, certaines API, ... soit grâce à un daemon soit grâce à une librairie. Malheureusement, OPA ne prend pas en charge OpenStack.

3 Description

Moon est un composant *externe* et *indépendant* capable de gérer des requêtes d'autorisation venant de services tiers. L'administrateur définit le modèle de politique de sécurité qu'il souhaite appliquer au projet ou au service demandeur. Ce modèle peut être un simple modèle RBAC ou un modèle MLS (Multi Level Security, les niveaux de sécurité Confidentiel Défense, Secret Défense, ...) ou un modèle arbitraire, type ABAC (Attribute Based Access Control). Moon se base sur un méta-modèle spécifique permettant de modéliser de nombreux modèles de politique de sécurité. Après avoir choisi son modèle de politique de sécurité, l'utilisateur l'applique au projet ou au service puis le configure en fonction des spécificités du projet ou du service. La configuration s'effectue en fonction du service demandeur. Prenons le cas d'OpenStack, ce dernier génère un triptyque à destination des services tiers d'autorisation : le sujet, l'action et l'objet. À ce triptyque s'ajoute le projet depuis lequel l'utilisateur émet la demande. La configuration de Moon va prendre en compte ce trio et va appliquer et lier chaque élément à un ou plusieurs attributs. Par exemple, le modèle va imposer de lier le sujet à un rôle si nous avons un modèle de type RBAC. A contrario, si nous avons un modèle de type MLS, le modèle va imposer de lier le sujet à un niveau de sécurité et l'objet à un autre niveau de sécurité. Ce qui permettra ensuite de mettre en relation ces niveaux de sécurité dans les règles de la politique de sécurité. Par exemple, un sujet de niveau “Confidentiel Défense” ne pourra utiliser qu'un objet de niveau “Confidentiel Défense”. Par contre, un sujet de niveau “Secret Défense” pourra utiliser des objets de niveau “Confidentiel Défense” et “Secret Défense”. À ces éléments venant du service demandeur, nous avons aussi ajouté la possibilité de créer des règles en fonction d'attributs extérieurs comme le mode d'utilisation d'une plateforme. Par exemple, si la plateforme (cible ou pas) est en mode

“build”, certaines actions seront possibles (création de machines virtuelles, gestion réseau, ...) alors qu'en mode “run” ces actions seront impossibles.

Cette plateforme fonctionne actuellement avec OpenStack et OpenDaylight. En théorie, elle pourrait fonctionner sur tout type de plateforme ayant un besoin d'autorisation complexe qui peut être externalisé (ie. où l'on peut intercepter, via la plateforme, les demandes des utilisateurs).

4 Cas d'utilisation

4.1 Cas 1

Votre plateforme utilise un contrôle d'accès spécifique et vous souhaitez utiliser un autre. Par exemple : l'autorisation de plateforme est basée sur des rôles et vous souhaitez avoir des rôles et des groupes.

4.2 Cas 2

L'infrastructure d'autorisation de votre plateforme utilise plusieurs fichiers de configuration et vous souhaitez centraliser votre configuration. Par exemple : OpenStack utilise plusieurs policy.json pour Keystone, Nova, Neutron,... et vous ne souhaitez gérer qu'un seul fichier.

4.3 Cas 3

Votre cadre d'autorisation a une vue macroscopique du processus et vous voulez entrer dans plus de détails. Par exemple : votre plateforme autorise sur des projets globaux et non sur des éléments spécifiques dans ces projets.

4.4 Cas 4

Au contraire, le cadre d'autorisation entre dans des détails trop petits et vous voulez avoir une vue plus macroscopique. par exemple : votre plateforme donne une autorisation pour chaque machine virtuelle et vous souhaitez classer ces machines virtuelles.

5 Architecture

La plateforme Moon est basée sur une architecture maître / esclaves. Le “manager” est le maître, il a pour objectif de gérer l'ensemble des données de la plateforme. Pour modifier les modèles, règles, ... de la

plateforme, l'utilisateur a accès à des API HTTP Rest sur ce manager. Le manager s'appuie sur des esclaves qui ont pour objectif de traiter les requêtes d'autorisations venant des systèmes externes. Ces esclaves sont constitués de 2 éléments ; le Wrapper récupére les requêtes en elle-même et les redirige vers le bon Engine. Ce dernier traite la demande via un moteur spécifique. L'Engine ne contient aucune donnée en dur dans sa mémoire, dès qu'une demande arrive depuis le Wrapper, il vérifie dans sa mémoire en cache si il a toutes les informations nécessaires pour traiter la demande. Dans le cas contraire, il effectue une requête au Manager qui lui fournit les informations juste nécessaires et les met en cache. Il existe un deuxième mode de fonctionnement où l'Engine est seul et isolé du reste de la plateforme. Dans ce cas, la configuration est injectée au démarrage de l'Engine et ne pourra plus être modifiée par la suite. Ceci permet d'avoir un moteur de politiques de sécurité totalement indépendant qui ne nécessite pas une architecture complète pour fonctionner.

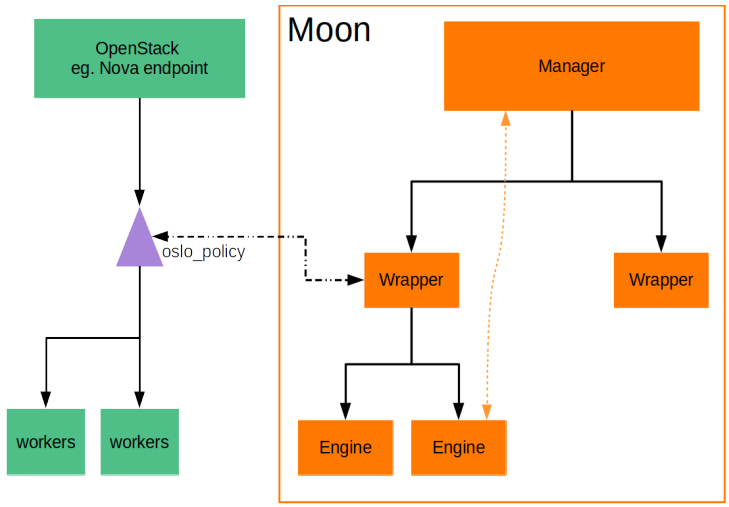

Fig. 1. Architecture de la solution Moon.

6 Politiques de sécurité

Comme exprimé ci-dessus, le méta-modèle utilisé dans Moon pour créer des modèles de politiques de sécurité est suffisamment puissant pour créer des politiques de sécurité ultra personnalisables. Voici quelques

exemples de modèles que nous avons déjà créer. Le méta-modèle étant assez complexe, nous détaillons ici qu'une version simplifiée :

— le système tiers envoie au moteur de politique un triptyque sujet, objet, action,
— le moteur va lier chaque élément du triptyque avec un attribut spécifique,
— ces attributs servent à créer des règles de politiques de sécurité.

6.1 RBAC amélioré

Imaginons qu'un administrateur souhaite catégoriser ses utilisateurs en fonction de leur rôle mais aussi en fonction d'un ou de plusieurs groupes. Par exemple, Alice fait partie des administrateurs au niveau de l'entreprise alors que Bob fait partie des administrateurs mais uniquement au niveau des développeurs et Clara est une utilisatrice standard du système d'information au niveau des ressources humaines. Chacune de ces personnes vont avoir des droits spécifiques sur le système d'information que l'on peut traduire par les règles suivantes :

— Le rôle administrateur du groupe entreprise va pouvoir exécuter toutes les actions dans le système,
— le rôle administrateur du groupe développeur va pouvoir exécuter des actions d'administration uniquement sur le système d'information des développeurs
— le rôle membre du groupe ressources humaines va pouvoir exécuter des actions de lecture seule sur le système d'information des ressources humaines

Cela va donner :

Sujets	Attributs
Alice	rôle : admin
Alice	groupe : entreprise
Bob	rôle : admin
Bob	groupe : développeur
Clara	rôle : membre
Clara	groupe : rh

Objets	Attributs
serveur_adm_ent	serveur_adm_ent
serveur_adm_dev	serveur_adm_dev

Objets	Attributs
serveur_membre_rh	serveur_membre_rh

Actions	Attributs
actions_écriture	actions_écriture
actions_lecture	actions_lecture

Dans certains cas simples, seul un des triptyques a besoin d'être lié à des attributs, c'est le cas ici pour les sujets (liés aux rôles et aux groupes). Dans les autres cas, l'élément (objet ou action) est lié à lui-même. Au niveau des **règles**, nous lions les attributs entre eux :

Sujets	Objets	Actions
rôle : admin, groupe : entreprise	serveur_adm_ent	actions_écriture
rôle : admin, groupe : développeur	serveur_adm_dev	actions_écriture
rôle : membre, groupe : rh	serveur_membre_rh	actions_lecture

6.2 MLS

Imaginons maintenant que tous les serveurs de l'entreprise puissent être tagués "Secret Défense", "Confidentiel Défense", ... Les utilisateurs "Secret Défense" auront accès à tous les serveurs "Secret Défense", "Confidentiel Défense", ... alors que les utilisateurs "Confidentiel Défense" n'auront accès qu'aux serveurs "Confidentiel Défense" et "Standard".

Cela va donner :

Sujets	Attributs
Alice	Secret Défense
Bob	Confidentiel Défense
Clara	Standard

Objets	Attributs
serveur_adm_ent	serveur_adm_ent
serveur_adm_dev	serveur_adm_dev
serveur_membre_rh	serveur_membre_rh

Actions	Attributs
actions_écriture	actions_écriture
actions_lecture	actions_lecture

Au niveau des **règles**, nous lions les attributs entre eux :

Sujets	Objets	Actions
Secret Défense	Secret Défense	actions_écriture
Secret Défense	Secret Défense	actions_lecture
Secret Défense	Confidentiel Défense	actions_écriture
Secret Défense	Confidentiel Défense	actions_lecture
Secret Défense	Standard	actions_écriture
Secret Défense	Standard	actions_lecture
Confidentiel Défense	Confidentiel Défense	actions_écriture
Confidentiel Défense	Confidentiel Défense	actions_lecture
Confidentiel Défense	Standard	actions_écriture
Confidentiel Défense	Standard	actions_lecture
Standard	Standard	actions_écriture
Standard	Standard	actions_lecture

6.3 RBAC + attributs

Imaginons maintenant que nous avons besoin d'un modèle RBAC standard mais que nous avons besoin d'inclure un attribut externe à la plateforme. Cet attribut va permettre de caractériser l'état du réseau, le réseau va donc être catégoriser suivant 2 modes :

— le mode "build" : il est en cours de construction,
— le mode "run" : le réseau est en cours d'utilisation et ne peut pas être modifié.

Cela va donner :

Sujets	Attributs
Alice	administrateur
Bob	membre
Clara	membre

Objets	Attributs
serveurs	serveurs

Actions	Attributs
actions_écriture	actions_écriture
actions_lecture	actions_lecture

Attributs externes	Valeurs
mode	build, run

Au niveau des **règles**, nous lions les attributs entre eux :

Sujets	Objets	Actions
administrateur	serveurs	actions_écriture, build
administrateur	serveurs	actions_lecture, build
administrateur	serveurs	actions_lecture, run
membre	serveurs	actions_lecture, run

Nous pouvons remarquer dans ce dernier exemple que seuls les utilisateurs ayant le rôle d'administrateurs pourront effectuer des actions en lecture/écriture sur le réseau uniquement si l'infrastructure est en mode "build". Quand l'architecture est en mode "run", tous les utilisateurs auront un accès en lecture seule.

Tous ces exemples sont donnés à titre indicatif. Dans une véritable version, il faudrait compléter avec notamment l'ensemble des actions. Sur une plateforme OpenStack, cela peut représenter plusieurs centaines d'actions à intégrer à la plateforme Moon.

7 Restrictions

7.1 Restrictions sur le méta-modèle

Grâce au méta-modèle de Moon, nous pouvons créer de nombreux modèles de politiques de sécurité très différents et facilement paramétrables. Néanmoins, l'actuel méta-modèle ne permet pas d'en réaliser certains. L'une des restrictions concernent la valeur des attributs, ces derniers

doivent être des valeurs entières (par exemple role_adm, role_member, ...). Il n'est pas possible de mettre des valeurs entières ou temporelles et d'émettre une règle avec un intervalle (entre 2 entiers, entre 2 dates, ...). Il sera peut-être possible dans un futur plus ou moins proche de le faire mais à l'heure actuelle, nous n'en avons pas vu la nécessité.

7.2 Restrictions sur la plateforme

Actuellement, la plateforme ne peut que répondre de façon statique à des requêtes d'autorisations. C'est-à-dire qu'elle ne peut pas prendre de décisions complexes comme isoler un réseau en cas de requêtes suspectes. La plateforme ne peut pas non plus mener des actions complémentaires comme, par exemple, créer un nouveau réseau dans le cas de la création d'un serveur ultra sensible. Ce sont des améliorations qui ont été identifiées comme ayant une valeur ajoutée intéressante pour la suite du projet.

7.3 Restrictions sur les informations récupérées depuis les services tiers

Moon récupère des informations depuis les requêtes envoyées par les services tiers (typiquement OpenStack). Dans certains cas, ces derniers ne fournissent pas toutes les données nécessaires pour bien maîtriser la politique pour ce service. Et Moon ne peut évidemment pas deviner l'information de la plateforme dont il dépend. À titre d'exemple, à l'heure actuelle, le composant Nova d'OpenStack fournit la gestion des machines virtuelles dans un cloud OpenStack. Or lors de l'envoi des requêtes à Moon, ces requêtes ne contiennent pas l'identifiant de la machine virtuelle ce qui interdit à Moon de positionner des règles sur des machines spécifiques même si le modèle de Moon le permet. Des discussions sont en cours pouvoir avoir ces informations au sein des requêtes.

8 Installation et configuration

Le projet Moon est développé en Python au sein de plusieurs équipes Orange. Ce projet a commencé sous forme de PoC (Proof of Concept) mais est maintenant géré comme un projet industriel (gestion des versions, intégration continue, ...). Le projet est OpenSource et dans une version majeure 5 sur le repository d'OPNFV ([1,3]). La plateforme possède 2 interfaces, l'une en console et l'autre sous forme d'une IHM Web. L'interface console permet de configurer et de manipuler l'ensemble des données.

L'interface Web ne permet pas actuellement d'avoir accès à toutes les fonctionnalités et il est encore nécessaire de travailler dessus avant de la libérer.

La configuration de la plateforme est parfois complexe. Pour limiter cette complexité, la plateforme propose une fonction d'importation de données. Cette fonction permet d'importer les modèles, les politiques de sécurité ainsi que la configuration de ces dernières.

Les commandes suivantes permettent d'installer la plateforme et de la configurer de façon basique.

```
# Installation
sudo pip install moon_manager
# Installation and configuration edition
sudo moon_manager_setup
sudo vi /etc/moon/moon.yaml
# Database creation and user creation
moon_manager db
moon_manager add_user admin
# Plateforme startup
moon_manager start_manager
# System checks
moon_manager status --human
```

Listing 1. Installation

À partir de cette étape, la plateforme est fonctionnelle, il faut alors créer un esclave (ie le Wrapper de la figure 1). Cet esclave va s'installer sur le même serveur que le maître. Dans un futur proche, les esclaves pourront s'installer sur des systèmes à distance :

```
moon_manager slaves add
moon_manager slaves list
moon_manager status --human
```

Listing 2. Configuration

Comme explicité ci-dessus, la configuration peut être complexe, d'où l'utilisation de la fonction d'importation. Le fichier importé est un fichier YAML qui permet de décrire le ou les modèles utilisés ainsi que la ou les politiques de sécurité ainsi que leur configuration (notamment les règles de la politique) :

```
moon_manager import mon_fichier.yaml
moon_manager status --human
```

Listing 3. Importation

Cette importation va induire la création automatique du composant Engine associé (cf. figure 1). La liaison entre le Wrapper et l'Engine pourra s'effectuer dans le fichier importé ou au travers d'une commande spécifique.

La plateforme Moon est maintenant opérationnelle et peut accepter des requêtes venant de services tiers. Ce service tiers peut être OpenStack. À partir d'un OpenStack fonctionnel, il faut modifier certains fichiers de configuration de l'OpenStack pour que les requêtes d'autorisation arrivent jusqu'à Moon. Ces fichiers sont les fichiers "policy.json" que l'on trouve dans les répertoires `/etc/[keystone,nova,glance,...]`. Ces fichiers sont toujours de la forme suivante :

```
"identity:delete_project": "rule:admin_required"
"identity:list_revoke_events": "rule:service_or_admin"
"identity:revoke_system_grant_for_group": "rule:admin_required"
"identity:get_region": ""
...
```

Listing 4. Oslo Policy

Il suffit de le modifier de la manière suivante :

```
"identity:delete_project": "http://<ip_of_moon>:10000/authz/oslo"
"identity:list_revoke_events": "http://<ip_of_moon>:10000/authz/oslo
    "
"identity:revoke_system_grant_for_group": "http://<ip_of_moon
    >:10000/authz/oslo"
"identity:get_region": "http://<ip_of_moon>:10000/authz/oslo"
...
```

Listing 5. Oslo Policy pour Moon

Le numéro de port est le numéro de port de l'esclave, par défaut 10000. Cette information se trouve via la commande `moon_manager status --human`. Toutes les lignes ne sont pas à modifier systématiquement, tout dépend de la granularité souhaitée. Une fois ces modifications effectuées, il suffit de configurer Moon pour connecter le projet Keystone n°X avec la politique de sécurité n°Y dans Moon :

```
moon_manager pdp add -v <keystone project ID> -s <policy ID> <name
    of the decision point>
```

Listing 6. Importation

9 Utilisation et tests

Le projet Moon n'est, à l'heure actuelle, pas intégré dans une plateforme OpenStack en production. Néanmoins il a été déployé en tant que PoC d'un projet de virtualisation de fonction réseau (NFV) mené par Orange en France. L'objectif était de permettre une gestion des droits d'administration OpenStack dynamique pour restreindre les possibilités d'attaque des fonctions réseaux virtualisées depuis l'infrastructure de virtualisation. Ainsi les comptes d'administration OpenStack ne devaient pas pouvoir interagir avec la machine virtuelle contenant la fonction réseau virtualisée lorsque celle-ci traitait du trafic réseau. Mais bien sûr ces comptes d'administration devaient pouvoir faire leur travail d'administration lorsque la fonction réseau était sortie de production. Ainsi un compte d'administration OpenStack utilisé à des fins malveillantes ne pouvait pas corrompre une fonction réseau en cours d'utilisation. La gestion des autorisations d'un système OpenStack en standard ne permet pas de gérer ce type de problématique. Le PoC a permis de démontrer que la modélisation des politiques de sécurité du moteur Moon permettait de faire cette distinction de mode de façon relativement simple.

La suite du PoC nous permettra notamment de caractériser les capacités de traitement de la plateforme Moon. Actuellement, nos tests montrent que la plateforme peut absorber jusqu'à 150/200 requêtes d'autorisation par seconde en mode complet, c'est-à-dire un nœud maître (Manager) et un nœud esclave (Wrapper + Engine). Comme explicité plus haut, un esclave (Engine) peut être mis en fonctionnement seul (mode standalone) ; dans ce cas, la plateforme peut absorber jusqu'à 300 requêtes par seconde. Ces tests ont été effectués sur une machine virtuelle avec 1 vCPU et 1Go de RAM. Ces différences s'expliquent par le fait qu'en mode complet, les esclaves interrogent régulièrement le maître (Manager) afin de s'assurer qu'il n'y a pas de modification dans la politique de sécurité. En mode standalone, l'esclave détient les données dans une mémoire en cache et ne revient jamais vers le Manager. Les capacités de traitement par rapport à l'ajout de plusieurs esclaves n'ont pas été encore testées. À l'heure actuelle, les esclaves sont installés et exécutés sur le même serveur que le maître, on peut donc faire l'hypothèse que les capacités de traitement des X esclaves en parallèle seront identiques aux capacités citées précédemment, c'est-à-dire 150 à 200 requêtes par seconde pour l'ensemble des esclaves (150/x par esclave). Afin de mieux répartir la charge, les esclaves devront être installés et exécutés sur des serveurs ou des machines virtuelles différentes, cela n'est pas le cas actuellement, mais cela fait parti des objectifs futurs.

Il est difficile d'affirmer que cette capacité de traitement sera suffisante pour un système OpenStack en production. Cela depend de beaucoup de paramètres, du nombre de projets, du nombre de VM par projet, du nombre d'utilisateurs, de la puissance des serveurs physiques, ... Néanmoins, dans le cas extrème où les capacités ne seraient pas suffisantes, les caractéristiques de configuration de la plateforme Moon et la configuration au niveau du module Oslo Policy permettent de minimiser le risque en ne gérant que les actions les plus critiques.

10 Conclusions

Moon est une plateforme permettant de se substituer à la gestion d'autorisation de services tiers. Elle permet une gestion plus fine des sujets, des objets et des actions. La finesse dépend surtout de la finesse des informations fournies par le service tiers, la plateforme Moon ne peut pas créer d'informations là où il n'y en a pas. Grâce à Moon, nous pouvons centraliser toute la gestion des autorisations et rendre dynamique la configuration des politiques de sécurité. La création de modèles de sécurité dans Moon permet d'obtenir des politiques complètement configurables permettant de s'affranchir des modèles rigides type RBAC.

Moon est actuellement testé avec OpenStack mais peut être théoriquement utilisé avec d'autres plateformes. Le développement a été architecturé de manière à pouvoir ajouter des Wrappers spécifiques à d'autres services. Nous sommes en cours d'étude avec Orange France pour intégrer la plateforme Moon dans une de leurs plateformes OpenStack.

Références

1. Code source du projet moon. `https://git.opnfv.org/moon/`. Accessed : 2020-04-16.
2. Policy-based control for cloud native environments. `https://www.openpolicyagent.org/`. Accessed : 2020-04-16.
3. Site web principal du projet opnfv. `https://www.opnfv.org`. Accessed : 2020-04-16.