Secure Information and Resource Sharing in Cloud Infrastructure as a Service

Yun Zhang
Institute for Cyber Security
Univ of Texas at San Antonio
San Antonio, TX 78249
Amy.u.Zhang@gmail.com

Ram Krishnan
Dept. of Electrical and Computer Engineering
Univ of Texas at San Antonio
San Antonio, TX 78249
Ram.Krishnan@utsa.edu

Ravi Sandhu
Institute for Cyber Security
Univ of Texas at San Antonio
San Antonio, TX 78249
Ravi.Sandhu@utsa.edu

ABSTRACT

Cloud infrastructure as a service (IaaS) refers to virtualized IT resources such as compute, storage and networking, offered as a service by a cloud service provider on demand to its customers (equivalently tenants). IaaS is a fast-maturing cloud service model today where tenants are strictly isolated from each other. With the prominence of IaaS as the next generation model for outsourcing IT infrastructure, we believe there is a need to facilitate secure sharing between tenants for various reasons such joint cyber incident response, catalyzing productivity, etc. In this paper, we investigate various models for information and resource sharing between tenants in an IaaS cloud. The models facilitate a tenant to share its IT resources with other tenants in a controlled manner. One motivation for sharing resources is for cyber incident response. We formally specify operational and administrative models for sharing and discuss enforcement and implementation issues in the widely-deployed OpenStack platform, the *de facto* open-source cloud IaaS software.

Categories and Subject Descriptors

H.1.1 [**Information Systems**]: MODELS AND PRINCIPLESSystems and Information Theory[Information theory; General systems theory]

General Terms

Theory

Keywords

Formal models; IaaS; OpenStack

1. INTRODUCTION AND MOTIVATION

Cloud computing is revolutionizing the way businesses obtain IT resources. Infrastructure as a service (IaaS) is a cloud service model [7] where a cloud service provider (CSP) offers compute, storage and networking resources as a service to its tenants, on demand. By tenant, we refer to an organization that is a customer of a CSP. The need to share information between organizations (commercial and governmental) continues to be an important requirement for various reasons including incident response, improved productivity, collaboration, etc.

Our premise is that as organizations move to cloud, the traditional information sharing activities would also need to move to cloud. Consider a community cyber incident response scenario where organizations that provide critical infrastructure to a community (such as a city, county or a state) share information related to a cyber incident in a controlled manner [4]. Sharing information amongst such organizations can greatly improve the resilience of increasingly cyber-dependent communities in case of co-ordinated cyber attacks [5]. One domain of a community that can benefit from cyber incident sharing is that of the electric power grid (see Wang et al [12] for a theoretical example of cascading a small scale attack to the entire U.S. power grid). A key requirement of effective cyber incident sharing and response is that the community needs a capability to share beyond simple data such as log files and documents. In particular, organizations need to replicate a smaller-scale version of their affected IT infrastructure including infected servers with network architectures, routers and firewall configurations, etc., for effective analysis, response and sharing with other organizations.

We consider two aspects in the context of information and resource sharing as one, to a certain degree, will depend on the other: models and technology. For technology, we focus on IaaS cloud for two reasons: (1) IaaS is one of the most adopted cloud service models today (as compared to platform and software as a service cloud models, referred to as PaaS and SaaS, respectively), and (2) IaaS is also the foundation of cloud with characteristics such as elasticity, self-service, etc. By gaining insights on issues related to sharing at this lower level of abstraction, the research community can develop better models for higher levels of abstractions of cloud such as PaaS and SaaS. Note that in the context of IaaS, the unit of sharing comprises virtual resources such as objects in a storage volume, virtual machines (VM), etc. This aligns well with the requirements of cyber incident sharing discussed earlier since organizations can share virtualized snapshots of their IT resources in a community cloud dedicated for, say, electric grid. For models, we focus on *administrative* and *operational* aspects. Administrative models are concerned about managing what resources are to be shared with which users, setting-up and tearing-down platforms for

WISCS'14, November 3, 2014, Scottsdale, Arizona, USA.

http://dx.doi.org/10.1145/2663876.2663884.

sharing, etc. Examples include a tenant administrator creating a shared secure isolated domain, adding/removing its users and resources to that domain, inviting other tenants to join the domain, etc. Operational models are concerned about controlling what activities users can perform on the shared platform. Examples include, creating new resources in the domain, modifying objects in storage volumes, importing new resources, etc.

To be concrete, we develop these models based on the facilities exposed by OpenStack [2], a widely-adopted open-source cloud IaaS project. OpenStack is a robust IaaS platform for building public, private or hybrid clouds that is developed and maintained by a vibrant community with participation from more than 200 world-leading organizations with a release cycle of 6 months. OpenStack software allows creating an IaaS cloud out of conventional hardware. Amazon Web Services (AWS) [1] is an example of a prominent commercial IaaS provider. We base our models on OpenStack since it is open-source and hence feasible to modify and experiment. Furthermore, real-world platforms such as OpenStack can inform us of the practicality of the models that we develop.

Traditionally, IaaS providers maintain strict separation between tenants for obvious reasons. Thus their virtual resources are strongly isolated. For instance, in OpenStack, a tenant user does not have the capability to access resources outside her domain. By domain, we refer to the administrative boundary of that tenant. In the latest release of OpenStack, each tenant is represented internally as a domain. Tenant users are assigned to a domain. Each domain can contain multiple projects and users are assigned to specific roles in the project. The roles provide the permissions to perform tasks such as creating a VM in a given project.

A critical concern for participating organizations is the level of control that they can maintain over the resources that are shared. In particular, participating organizations will need to ensure that the resources are shared only with users from select other organizations, and can retain the ability to enable and disable the sharing. Thus in order to share resources between tenants, we need to develop administrative and operational models that offer precise control to each tenant on what they are willing to share. As we will see, there are multiple alternatives for developing these models that can support different scenarios. The contribution of this paper is to explore the various models for sharing in IaaS, and present implementation approach in OpenStack.

We proceed as follows. We present some background on OpenStack in section 2. Section 3 gives three perspectives to consider when designing access control models for information and resource sharing (IARS) in IaaS. Then we give detailed description of two typical models. One is to set up a shared space inside one organization that is willing to share information. The other is to set up a shared space which is isolated from any of the organizations that are collaborating together. Section 4 gives a formal definition of the second model described in section 3, which defined an isolated space for IARS in the OpenStack platform. We present enforcement considerations of our model in section 5 and related work in section 6. Finally, we conclude in section 7.

2. BACKGROUND

In this section, we introduce how OpenStack is currently structured. OpenStack provides several services, including compute (Nova), identity (Keystone), block storage (Cinder), object storage (Swift), image (Glance), networking (Neutron) and Dashboard (Horizon), as shown in figure 1. Nova allows users to create their own virtual machines. Glance provides users with images (OS, software, configurations, etc.), which are used to create virtual machines. Swift allows users to store their data as swift objects. Cinder provides block storage which is attached to a virtual machine as a storage volume. Neutron provides users with networking services, where different VMs can be networked using virtual routers. Keystone provides users with security services, such as authentication and authorization. Horizon is the web-based dashboard where users can access all the services through a graphic user interface. In addition, OpenStack provides command line based clients to interface with each of the services.

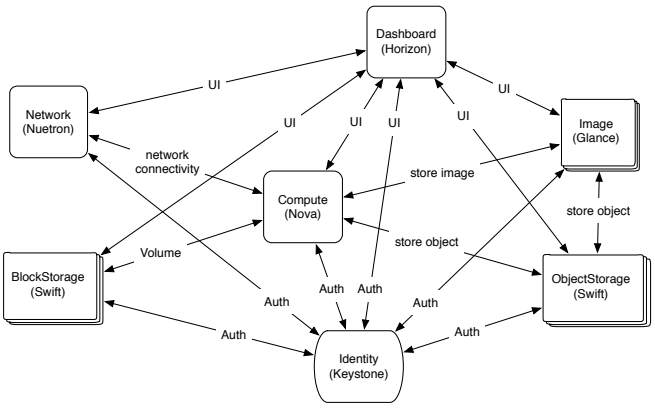

Figure 1: Architecture of OpenStack

In [10], the authors present a core OpenStack Access Control (OSAC) model based on the OpenStack Identity API v3, as shown in figure 2. The OSAC model consists of eight entities: users, groups, projects, domains, roles, services, operations, and tokens. Users represent people who are authenticated to access OpenStack cloud resources while a group is a set of users. Projects define a boundary of cloud resources—a resource container in which users can get access to the services the cloud provides, such as virtual machines, storages, networks, and so on. Domain is a higher level concept that equates to a tenant of the CSP. The projects in a domain represents the administrative boundary of its users and groups. Projects allow tenants to segment their resources and to manage their users' scope of access to those resources. Roles are global, which are used to specify access levels of users to services in specific projects in a given domain. Note that users are assigned to projects with a specific set of roles. An object type and operation pair defines actions which can be performed by end users on cloud services and resources. Users authenticate themselves to Keystone and obtain a token which they then use to access different services. The token contains various information including the domain the user belongs to, and the roles of the user in specific projects in that domain. We elaborate the model and these concepts further below.

Domains and Projects. In OpenStack, a project can only belong to one domain. A user has one home domain but can be assigned to multiple projects, which can be distributed in different domains. That is, the ownership of users and

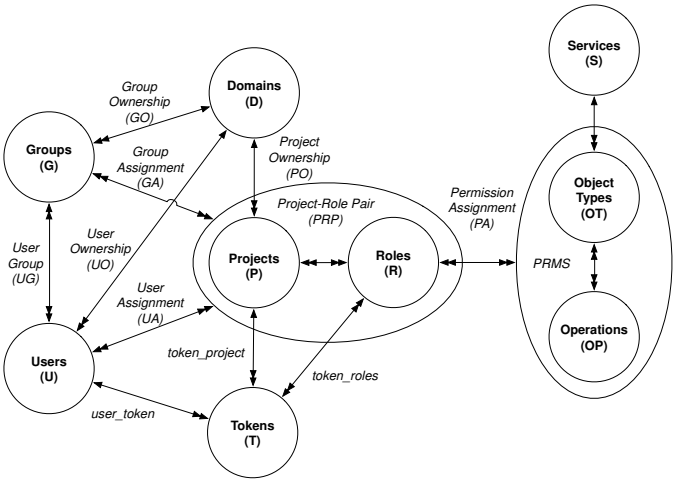

Figure 2: OpenStack Access Control (OSAC) model [10]

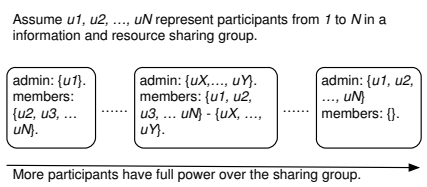

Figure 3: From administrative perspective of modeling

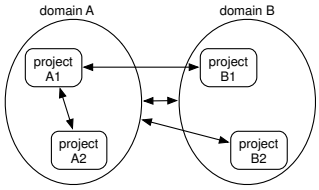

Figure 4: From operational perspective of modeling

projects can be defined by assigning them to a domain. Note that users in a domain are powerless unless they are assigned to a project with a particular role. Typically, domains are created by a CSP for its tenants.

Roles. Role defines the accesses of cloud services and resources the user can have. By assigning a role to a user, one can specify different access rights for the user. For instance, by assigning the *member* role to a user, the user receives all operational permissions over the resources in a project. By assigning the *admin* role to a user, the user receives admin permissions over a project. The accesses defined by roles are enforced by a policy engine in the cloud based on policy files where the roles are defined.

Tokens. There are two types of tokens in OpenStack. One is an unscoped token, which is used for initial authentication to a specific domain. Using the unscoped token, the user can obtain scoped tokens that are project-specific from Keystone. If a user has membership in two domains, the user can obtain two different unscoped tokens and thereby further obtain multiple scoped tokens for projects that belong to those domains. OpenStack clients facilitate this process.

Object Types and Operations. The concept of object types allow specifying different operations for different services. For instance, consider the Nova service. The object type for Nova is VM and operations on VM include start, stop, etc. In contrast, for Swift, the object type is Container and the operations include create, upload object, download object, etc.

Scope and Assumptions: In the model we develop in this paper, we confine our attention to compute (Nova) and object storage (Swift) services in OpenStack. Focussing on the Swift service allows us to investigate information sharing requirements between tenants, while focussing on Nova allows us to investigate resource (VM) sharing requirements between tenants. However, our models equally apply to other services in OpenStack such as Neutron, Cinder, etc. Furthermore, we confine our attention to information and resource sharing between tenants within a single cloud. These issues in the context of multiple CSPs is an interesting research problem in its own right and we plan to pursue this in future work.

3. MODELS FOR SECURE IARS IN IAAS

Informed by the OSAC model, in this section, we discuss various alternatives in designing IARS models in IaaS. Recall that a domain represents a tenant of the CSP and domains can contain multiple projects, where each project is a resource boundary as specified by the roles assigned to the user in that project. Also recall that domain admin is a super user who takes charge of operations inside a domain, including creating new projects, creating and adding users to a project, and so on. With the assumption that each domain represents an organization in a cloud platform, each project inside the domain can represent either an department or a temporary program in that organization. Domain admin roles are assigned to people who have the super power over the organization, such as the capability to manage the departments and initiate a new event in the organization. In the following, we conceptually refer to the sharing platform as a "group". We use the term group informally. Specifically, by group we do not mean the OSAC group unless explicitly stated otherwise. Later, we discuss how exactly we model this group in OpenStack.

We can model IARS from three perspectives: administrative, operational and control. From administrative perspective of modeling, assume we have n participants in the IARS group. We can have n levels of administrative controls, from one participant being in charge of the group to all the participants being in charge of the group, as shown in figure 3. In the model where one participant has absolute control over the collaborating group, this participant has full access to all the information and resources in the shared group, as well as having full power in determining shared group members, and which user can have what level of access over what information and resources inside the shared group. In the other extreme case of IARS administrative control, all the participants have full power over the group, including access to shared information and resources, and management of users in the group. Clearly, there are different degrees of control alternatives over this range as illustrated in figure 3. From the operational perspective of modeling in OpenStack, we have different levels of collaboration among projects and domains: project-to-project collaboration within the same

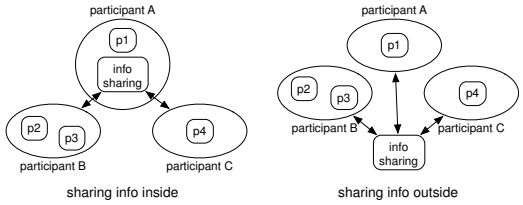

Figure 5: From control perspective of modeling

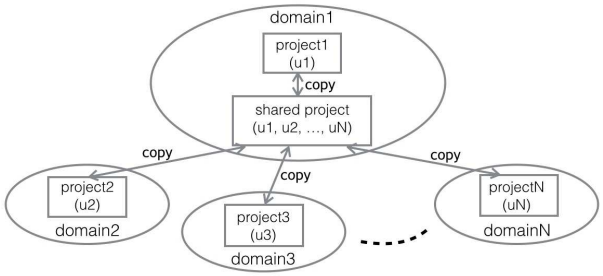

Figure 6: Architecture of Model 1

domain, project-to-project collaboration across different domains, project-to-domain collaboration and domain-to-domain collaboration, as show in figure 4. Project-to-project collaboration involves sharing between several projects either in the same domain or across different domains. Only users who are assigned to these projects can join the collaboration group. Project-to-domain collaboration occurs when a department needs to collaborate with an external organization. This is useful since not all collaboration scenarios need the whole organization to be involved. Project-to-domain perspective minimizes a tenant's exposure to other tenants. In scenarios, where two organizations need large-scale collaboration or merge their resources, domain-to-domain collaboration perspective is a useful construct.

From a source control perspective of modeling, there are two ways to share information and resources among participants. One approach is to host the group inside one of the existing participants' administrative scope. The other approach is to host the sharing group within a third party's administrative scope, where no single participant maintains a superior control on the group. For the first approach, any participant can set up an IARS project in its domain and invite others to the sharing group. For the second approach, participants who are willing to share sensitive information can get together to set up a working domain outside of any of the member participant domains. The relationship of shared space with the member participants is showed in figure 5.

Based on these perspectives and the levels of administrative controls discussed above, we now provide an overview of three model alternatives using OpenStack constructs. Specifically, we use the terms project and domain as specified earlier by the OSAC model. In our discussion below, we refer to data sharing. However, it equally applies to resource (VMs, networks, etc.) sharing as well.

Model 1

In model 1, one of the collaborating participants holds a shared project where all the other collaborating participants are invited to this shared project. We call the participant who initiates the collaboration as the shared project holder, and the rest of participants the shared project members. The shared project is hosted inside the holder's domain and is isolated from the rest of the projects in the domain in order to secure the sensitive information shared by collaboration members. In this model, the shared project holder has the full power over the shared information and resource, as well as the member participants users. The sharing occurs either at a domain-to-domain level or a project-to-domain level. Figure 6 illustrates this approach.

When collaboration begins, the holder creates an empty shared project for IARS in its own domain. The holder invites other organizations to join the shared project as members by adding their users to the shared project. The holder decides which users can be added to the shared project and what permissions can be assigned to those users. A user who is added to the shared project is assigned with a role, which gives the user proper permission inside the shared project. Data can be brought into the shared project by member users from their original projects in their home domains. Due to the ownership of the shared project, the holder can decide what data is allowed to bring in and how the information and resources are shared.

During IARS, member users inside the shared project exchange their data, work on the shared data and finally generate new data, which may be copied back to members' original domains. Members can create, update and delete data based on their roles in the shared project. After the collaboration, the holder is responsible for disbanding the shared project. All the data which is attached to the shared project are deleted, and all the processes and sessions which are executing in the shared project are killed.

Model 2

In model 2, all the collaborating participants together hold a shared project located in an external domain. This domain does not belong to any of the members of the collaborating participants. In order to facilitate IARS among organizations, we introduce the concepts of Secure Isolated Domain (SID) and Secure Isolated Project (SIP) to OpenStack platform. SID is a special domain specifically designed for IARS. SIP is a secure project set up for each IARS team. In this model, each participant of the collaboration have equal power over the shared information and resource. The sharing happens in domain-to-domain level. Thus the participants are domains of the tenants. Figure 7 illustrates this approach.

In the model, we design a SID for every possible combination of organizations (tenants/domains) in the cloud. Within each SID, there can be multiple SIPs. For instance, different collaborations may occur between different users in a group of organization, which leads to different SIPs in the same SID. Note that a SID between a set of organizations will only need to be created if a collaboration activity is ever necessary between those organization.

When the collaboration starts, a group of organizations together create a SIP in a SID. The creation process is complete only after all the members of the group agree to join to the SIP. Each organization has the same access control right and priority inside the SIP. Inside the SIP, each participating organization has a administrative user who decides which other users from his home domain can be added to the SIP and with what permissions. A user who is added

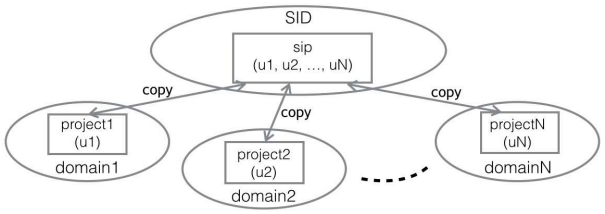

Figure 7: Architecture of Model 2

to the SIP is assigned with a role at joining time. Users inside the SIP can bring data into the SIP from their original projects in their home domains. Users decide what data will be brought in and how the information is shared. Like earlier, users inside the SIP exchange their data, work on the shared data and finally generate new data, which are allowed to be copied back to their respective original domains. Users can create, update and delete data as per their roles. After the collaboration, the set of administrative users are responsible for disbanding the SIP. All the data which are attached to the SIP are deleted, and all the processes and sessions which are running in the SIP are killed.

Model 3 Model 3 is a slight variation of model 2. Similar to model 2, we still utilize SID and SIP concepts to design the model. A set of organization administrative users are responsible for creating, updating and deleting the SIP, and this set of users become the administrative users of the SIP. After the SIP is created, SIP administrative users determine which users inside of their home domain can be added to a SIP or removed from the SIP. The difference is that, instead of multiple SIDs, we design a single SID with multiple SIPs for each collaboration activity between organizations. The idea is that we want to hide the domain level administration of the SIPs which simplifies the implementation of this model.

In this model, we give a default SID to hold all possible SIPs that users can create. The default SID function is transparent to users. Every time a user issues a collaboration activity create request, a SIP is created in the default SID. The system holds the default SID permanently.

Summary: Model 1 is convenient for information and resource sharing in cases of low-assurance requirements on confidentiality of the shared data. It is easy to deploy model 1 in current OpenStack cloud platform. However, since it gives one of the collaborating organizations overwhelming power on the shared project, it can create trust issues between the holder domain and member domains. Moreover, by bring in users outside of its home domain, the holder might be under the risk of exposing itself to other tenants.

Model 2 and model 3 provide all organizations that are involved in IARS an external secure space to cooperate and work together on the shared data. They avoid the disadvantages of model 1, and provide the organizations equal access control over the shared space. Such a capability is valuable in scenarios such as cyber incident response where the data is very sensitive and each participating organization would wish to have equal control on the shared space. Besides, hosting the shared project outside the organization alleviates mutual suspicions that arise in model 1. In the following section, we give detailed design of model 2 for IARS in IaaS. We call it the OpenStack Access Control model with SID extension (OSAC-SID).

4. OSAC-SID MODEL

In the OSAC-SID model, we assume that a user can belong to only one organization, which is consistent with the user and home-domain concept in OpenStack. We also assume that users can be assigned to projects across domains and access those projects separately using appropriate tokens. We extend the OSAC model to include SID and SIP components, as shown in figure 8. For every possible combination of organizations in the cloud, we create a SID to include all SIPs that will be set up among these organizations. For each IARS event, we create a SIP.

Similar to the concept of domains which is designed to add one more layer for administration of projects, SID is a administrative concept to manage SIPs. The SID function is transparent to users. SID and SIP components are isolated from the regular domain and projects components. Unlike the concept of domains, there are no users that belong to a SID. A SID exists only for setting up SIPs. However, since a SID is formed and associated with a group of domains, there are users who will be associated with the SID—but only under the constrains that they are from the group of domains which are associated with the SID.

A SIP provides a secure isolated space for IARS in the cloud. In other words, SIP is another type of resources container in OpenStack, which is restricted only for IARS among domains and projects. It means that users who are assigned to a SIP have similar access capability to request all services cloud provides like users who are assigned to a project. Although for every user, only one home domain can be assigned, users can belong to multiple projects and SIPs. Users can be assigned with multiple roles in same projects/SIPs. We define the attributes of OSAC-SID model in the following part. We inherit some of the attributes of OSAC model discussed earlier. For simplicity, we choose to ignore two entities in OSAC model: group and token, for reasons that group entity is just a set of users, and token is used in the same way as for a domain/project and for a SID/SIP.

Note that a SID is a regular domain in OpenStack with the added restriction that a user who belongs to the SID will have to be a member in one of the regular domains. Similarly, a SIP is a regular project in OpenStack with the added restriction that, for a user to get access to a SIP, she has to be authorized by at least one administrator from any of the participating domains in the SID. Furthermore, the authorization can be revoked by any of those SID administrators.

4.1 Attributes in OSAC-SID

Definition 1. OSAC-SID model has the following attributes.

- SID is a finite sets of special secure isolated domain which holds secure isolated projects(SIPs).
- U, P, SIP, D, C, SO, VM, R, S, OT and OP are finite sets of all existing user ids, project ids(except SIPs), secure isolated project(SIP) ids, domain ids(except SID), swift container ids, swift object ids, nova virtual machine ids, roles, services, object types and operations respectively in OpenStack cloud system.
- UNIV_P, UNIV_D, UNIV_SIP, UNIV_SID, UNIV_C, UNIV_SO and UNIV_VM represent the universe of ids of projects, do-

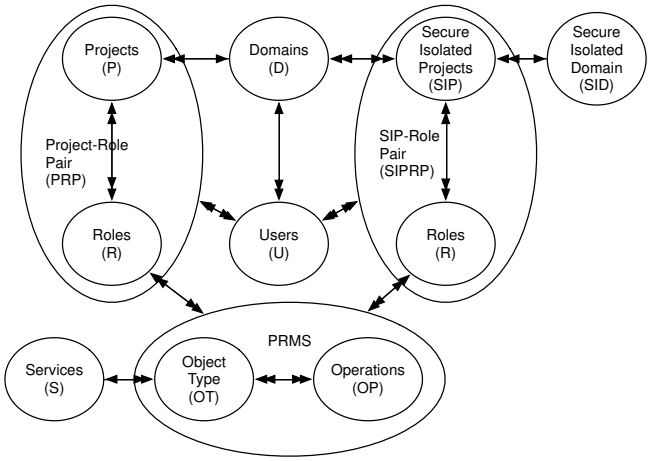

Figure 8: OpenStack Access Control (OSAC) model with SID extension(ignore group and token components)

mains, SIPs, SIDs, swift containers, swift objects, and virtual machines.

- user owner(UO) : U $\rightarrow$ D, a function mapping a user to its owning domain. Equivalently viewed as a manytoone relation UO $\subseteq$ U $\times$ D.
- domain admin(DA) : U $\rightarrow$ {True, False}, a function mapping a user to the resolution whether the user is the admin of its owning domain.
- project owner(PO) : P $\rightarrow$ D, a function mapping a project to its owning domain. Equivalently viewed as a many-to-one relation PO $\subseteq$ P $\times$ D.
- object type owner(OTO) : OT $\rightarrow$ S, a function mapping a object type to its owning service. Equivalently viewed as a many-to-one relation OTO $\subseteq$ OT $\times$ S.
- vm owner(VMO): VM $\rightarrow$ P $\cup$ SIP, a function mapping a virtual machine to its owning project. Equivalently viewed as a many-to-one relation VMO $\subseteq$ VM $\times$ (P $\cup$ SIP).
- container owner(CO): C $\rightarrow$ P $\cup$ SIP, a function mapping a container to its owning project. Equivalently viewed as a many-to-one relation CO $\subseteq$ C $\times$ (P $\cup$ SIP).
- swift object owner(SOO): SO $\rightarrow$ C, a function mapping a swift object to its owning container. Equivalently viewed as a many-to-one relation SOO $\subseteq$ SO $\times$ C.
- vm owner(VMO): VM $\rightarrow$ P $\cup$ SIP, a function mapping a virtual machine to its owning project. Equivalently viewed as a many-to-one relation VMO $\subseteq$ VM $\times$ (P $\cup$ SIP).
- sid owners(SIDO) : SID $\rightarrow 2^D$, a function mapping a SID to the owning domains.
- sip owners(SIPO) : SIP $\rightarrow$ SID, a function mapping a SIP to the owning SID.
- sip admins(SIPA) : U $\rightarrow 2^{SIP}$, a function mapping a user to a set of SIPs where the user is assigned to be admin of the SIP.
- PRP = P $\times$ R, the set of project-role pairs.
- SIPRP = SIP $\times$ R, the set of SIP-role pairs.
- PERMS = OT $\times$ OP, the set of permissions.
- PA $\subseteq$ PERMS $\times$ R, a many-to-many permission to role assignment relation.
- UA $\subseteq$ U $\times$ PRP, a many-to-many user to project-role assignment relation.
- SIPUA $\subseteq$ U $\times$ SIPRP, a many-to-many user to SIP-role assignment relation.

4.2 OSAC-SID Administrative Model

In the OSAC model, there are three levels of administrative roles: *cloud_admin*, *domain_admin*, and *project_admin*. In OpenStack, a project admin is a user who has full access on that project. A domain admin is a super user who is in charge of the domain where the domain admin can add and remove users, create, delete and update projects. A cloud admin is the super user of the entire cloud. In our administrative OSAC-SID model, we avoid a cloud admin to be directly involved with SID/SIP related activities. Also, since the permission of a project admin is limited to his own projects, it is not convenient for a project admin to initiate a SIP establishment, unless the information and resources need to be shared happens to be only in that project. Thus we choose domain admin to initiate an IARS activity since domain admins have the access over all projects inside a domain. In our model, we design domain admins to establish a SID/SIP when a collaboration activity occurs. In the administrative model, for simplicity, we explicitly constrain attribute OP to include only *create* and *delete* operations. In table 1, we give the details of OSAC-SID administrative model.

Administrative model:

SIPCreate: A set of domain administrative users (uSet) form a collaborating group to create a SIP. Each of those domain administrative users is also the administrative user of the SIP. For every domain which is associated with the SIP, there is only one administrative user who can be assigned to the SIP as an administrative user of the SIP. Each SIP has a unique SIP id in the OpenStack platform.

SIPDelete: After IARS is complete in the SIP, the SIP with all data attached to it will be deleted securely. All the processes and objects will be killed in the deleting process. Before the SIP is deleted, it is suggested that all the associated domains should have copied all the resulting objects back to their respective home domains.

SIDCreate: A SID is created automatically when the first SIP is created for a group of collaborating domains. Each SID is associated with a certain number of domains. After the SID is created, it will continue to exist until some SIP exists in that SID. Thus the total number of SIDs in the cloud is $2^{|D|}$.

SIDDelete: After all SIPs inside a SID are deleted, a SID will be deteled.

UserAdd: SIP administrative users can add users from their home domain to the SIP. We refer to these that were users added after the SIP creation as normal users in the SIP. The difference between SIP administrative and normal users in our model is that the SIP administrative users have the right to add and remove normal users while normal users only have operational accesses over the objects in a SIP.

UserRemove: SIP administrative users can remove normal users who belong to the same home domain from the SIP. A SIP administrative user cannot remove a normal user who comes from a different home domain.

CopyObject: A user can copy a Swift object between his home project and a SIP that he belongs to.

Table 1: OSAC-SID Administrative model

Operation	Authorization Requirement	Update
SipCreate(uSet, sip) *(* a set of domain admin users together create a sip *)*	∀ u1, u2 ∈ uSet.((DA(u1)=True ∧ DA(u2)=True ∧ u1 ≠ u2 ∧ UO(u1) ≠ UO(u2))) sip ∈ (UNIV_SIP - SIP)	SIPO(sip) = $\bigcup_{\forall u \in uSet}$ UO(u) SIPU(sip) = uSet ∀ u ∈ uSet.SIPA(u) = SIPA(u) ∪ {sip} SIP' = SIP ∪ {sip}
SipDelete(uSet, sip) *(* delete the sip*)*	∀ u ∈ uSet.((DA(u)=True ∧ sip ∈ SIPA(u))) ∧ SIPO(sip) = $\bigcup_{\forall u \in uSet}$ UO(u) sip ∈ SIP	SIPO(sip) = NULL SIPU(sip) = NULL ∀ u ∈ uSet.SIPA(u) = SIPA(u) - {sip} SIP' = SIP - {sip}
SidCreate(uSet, sid) *(* a set of domain admin users together create a sid *)*	∀ u1, u2 ∈ uSet.((DA(u1)=True ∧ DA(u2)=True ∧ u1 ≠ u2 ∧ UO(u1) ≠ UO(u2))) sid ∈ (UNIV_SID - SID)	SIDO(sid) = $\bigcup_{\forall u \in uSet}$ UO(u) SID' = SID ∪ {sid}
SidDelete(uSet, sid) *(* delete the sid*)*	∀ u ∈ uSet.((DA(u)=True ∧ sid ∈ SIDA(u))) ∧ SIDO(sid) = $\bigcup_{\forall u \in uSet}$ UO(u) sid ∈ SID	SIDO(sid) = NULL SID' = SID - {sid}
UserAdd(admin, r, u, sip) *(* sip admin add a normal user to a sip*)*	sip ∈ SIPA(admin) ∧ DA(admin)=True ∧ UO(admin) ∈ SIDO(sid) ∧ sip ∈ sid ∧ UO(u) = UO(admin) ∧ r ∈ R ∧ sip ∈ SIP ∧ u ∈ U	(u, (sip, r)) ∈ SIPUA ∧ SIPU'(sip) = SIPU(u) ∪ {u}
UserRemove(admin, r, u, sip) *(* sip admin remove a normal user from a sip*)*	sip ∈ SIPA(admin) ∧ DA(admin)=True ∧ UO(admin) ∈ SIDO(sid) ∧ sip ∈ sid ∧ UO(u) = UO(admin) ∧ r ∈ R ∧ sip ∈ SIP ∧ u ∈ U ∧ (u, (sip, r)) ∈ SIPUA	(u, (sip, r)) = NULL ∧ SIPU'(sip) = SIPU(u) - {u}
CopyObject(u, so1, c1, p, d, so2, c2, sip, sid)	so1 ∈ SO ∧ c1 ∈ C ∧ p ∈ P ∪ SIP ∧ d ∈ D ∪ SID ∧ so2 ∈ (UNIV_SO - SO) ∧ c2 ∈ C ∧ sip ∈ P ∪ SIP ∧ sid ∈ D ∪ SID ∧ (so1, c1) ∈ SOO ∧ (c1, p) ∈ CO ∧ (p, d) ∈ PO ∪ SIPO ∧ (c2, sip) ∈ CO ∧ (sip, sid) ∈ PO ∪ SIPO ∧ u ∈ U ∧ (u, (p, r)) ∈ UA ∧ (u, (sip, r)) ∈ SIPUA)	SO' = SO ∪ {so2} SOO' = SOO ∪ {(so2, c2)}

† uSet: a set of domain admin users.

4.3 OSAC-SID Operational Model

In the operational model, we mainly want to show how and what operations a user can issue in our model. We use nova for computing resources and Swift for data storage. For ease of presentation, we only show the core operations related to virtual machines and containers, such as create and delete. Create method allows users to create a new instance of virtual machine or a container in a project or a SIP. Delete method allows users to delete an existing instance of a virtual machine or container in a project or a SIP. For Swift object, we kept main operations from Swift, such as upload object and download object. In table 2, we give the details of operational model.

Operational model:

After a user is assigned to a SIP, the user can issue following operations:

CreateVM/DeleteVM: A user can create/delete a virtual machines in a SIP to which the user is assigned to.

CreateContainer/DeleteContainer: A user can create/delete a container in a SIP to which the user is assigned to. A Swift container holds Swift objects.

CreateObject/DeleteObject: A user can create/delete a Swift object in a container in a SIP to which the user is assigned to.

UploadObject: A user can upload a local file to a Swift object in a container in a SIP to which the user is assigned to.

DownloadObject: A user can download a Swift object to the local drive from a container in a SIP to which the user is assigned to.

5. ENFORCEMENT

In this section, we discuss enforcement and implementation considerations in OpenStack. In order to deploy the model in OpenStack platform, we need to modify Keystone server to include SID and SIP functionality, which has specific features that facilitate IARS. Recall that Keystone facilitates authentication and authorization in OpenStack.

SID functionality include SID creation, updating and deletion, and so on. SID actions are invoked through user's invocation of SIP actions. Each time a SIP creation request is sent to Keystone server, the server will check whether a SID for the set of domains already exists; if not, a new SID with a SIP will be created and associated with these domains. If true, a SIP will be created inside the SID and associated with these domains by inheriting the attributes of the SID. All SIPs that are created between the same set of domains will be set up inside the same SID.

SIP functionality allows users bring in their data and utilize the cloud resources to process the data. It provides users full access to the resource inside a SIP, where a user can create, update and delete an object. With the extension of copy function in Swift, users can copy the objects between their home project and the SIP under restrains. To create a SIP, a set of domain admin users are required to send the request to the Keystone server. To delete a SIP, the same set of admin users are required to agree on the action of delete. Unlike project-user assignment, SIP-user assignment is strictly constrained. A user need to be verified before joining a SIP. Since services are requested using token, after we add SID and SIP functions, we also need to modify the authentication part to include SID and SIP attributes. Following is the outline of OSAC-SID model enforcement in OpenStack.

Table 2: OSAC-SID Operational Model

Operation	Authorization Requirement	Update
Nova:		
CreateVM(vm, sip, u)	vm ∈ (UNIV_VM - VM) ∧ sip ∈ SIP ∧ u ∈ U ∧ ∃ (perms, r) ∈ PA.(perms = (vm, create) ∧ (u, (sip, r)) ∈ SIPUA)	VM' = VM ∪ {vm} VMO' = VMO ∪ {(vm, p)}
DeleteVM(vm, sip, u)	vm ∈ VM ∧ sip ∈ SIP ∧ u ∈ U ∧ ∃ (perms, r) ∈ PA.(perms = (vm, delete) ∧ (u, (sip, r)) ∈ SIPUA)	VM' = VM - {vm} VMO' = VMO - {(vm, p)} vm = NULL
Swift:		
CreateContainer(c, sip, u)	c ∈ (UNIV_C - C) ∧ sip ∈ SIP ∧ u ∈ U ∧ (u, (sip, r)) ∈ SIPUA)	C' = C ∪ {c} CO' = CO ∪ {(c, p)}
DeleteContainer(c, sip, u)	c ∈ C ∧ sip ∈ SIP ∧ u ∈ U ∧ (u, (sip, r)) ∈ SIPUA)	C' = C - {c} CO' = CO - {(c, p)} c = NULL
UploadObject(so, c, sip, u)	so ∈ UNIV_SO ∧ c ∈ C ∧ sip ∈ SIP ∧ u ∈ U ∧ (u, (sip, r)) ∈ SIPUA) if ∃ so' ∈ SO. (so = so'), then so' = so	SO' = SO ∪ {so} SOO' = SOO ∪ {(so, c)}
DownloadObject(so, c, u, p)	so ∈ SO ∧ c ∈ C ∧ sip ∈ SIP ∧ u ∈ U ∧ (u, (sip, r)) ∈ SIPUA)	
DeleteObject(so, c, sip, u)	so ∈ SO ∧ c ∈ C ∧ sip ∈ SIP ∧ u ∈ U ∧ (u, (sip, r)) ∈ SIPUA)	SO' = SO - {so} SOO' = SOO - {(so, c)} so = NULL

5.1 Add SID and SIP Functionality

Assignment: In OpenStack, the user assignment module is part of the Keystone server. To add SID and SIP functionality to Keystone, we need to modify the assignment module. The functionality of SID/SIP include at least create, delete and update user assignment on a SID/SIP. In assignment module of Keystone server, there are three components in charge of handling client requests: router, controller and core methods. To assign a user to a SID or SIP, we need to create these three components to have the attributes of SID and SIP. When a client sends a SID or SIP request to Keystone server, first, the router decides which controller method should match with the client request. After the controller gets the request mapping from router, it then looks for the corresponding core methods. Core methods will communicate with Keystone database and ask for add, delete or update information stored in corresponding tables.

Authentication and authorization with tokens: Similar to user-project-role assignment, a user is assigned to a SIP with a SIP-role pair. After a user is assigned to a SIP, the user has access to the resources in the SIP. That is, the user can get a token from Keystone and use the token to access the SIP. After SID/SIP attributes are added to token, Keystone can issue a token which can scoped to a SIP in a SID. Users can use this scoped token to access a SIP. Cloud services authorize users according to the user credential information stored in a scoped token.

Swift: Swift authorizes users by using a middleware to Keystone identity authorization system. By default, this middleware supports tokens with domain and project attributes, rather than SID and SIP attributes. To enable users to use Swift services, SID and SIP attributes need to be added to Swift authorization middleware.

Policy: New methods in SID and SIP modules have to be added to policy files in OpenStack Servers. In policy file, each method has corresponding rules defined. We need to add all new methods we define in SID and SIP modules to policy files in order to make those methods work under proper access control rules. By default, policy version v2 is used. In order to use domain admin role, we use policy v3 instead of v2. More fine-grained access control is defined in policy v3.

5.2 User Verification Process For SIP

Before a user is added to a SIP, it needs to be verified by a user-SIP verification process. As shown in the model, the SIP creation is initiated by a set of domain administrative users. A SIP is not allowed to be created unless all the requesting users send the same request to Keystone server. Thus, a verification process needs to ensure that all requesting users have sent the same request before a real SIP creation is issued. During IARS activity, if a new user requests to join the SIP, the user need to be verified if he comes from domains which are associated with the SID where the SIP is located.

Users that do not belong to the set of associated domains will be rejected by the server. After the sharing is done, the same set of domain administrative users who created the SIP issue a delete command to disband the SIP. At this point, a similar user-SIP verification process will occur, just like when the creation of the SIP was issued. Figure 9 gives the algorithm used in user-SIP verification process to initiate a request of SIP creation. (We assume user issues a SIP request with SIP name sip1 and a set of users: uset={u1, u2, ..., uN}) In the algorithm, function $CreateSip(sip\_name)$ is a openstack command to create a SIP with SIP name sip_name, and $CreateSid(default\_sid\_name)$ is a openstack command to create a SID with SID name $default\_sid\_name$. Users give sip_name while $default\_sid\_name$ is given by SID functions. From the user perspective, we have a sequence of actions from issuing a sip creation request to accessing a sip. First, user issues a SIP request command, which communicates to Keystone that a set of users wish to create a sip. Keystone assigns a SIP id and returns the id to users. Meantime, Keystone caches the request until all the member users send the same SIP request. Then Keystone creates the actual SIP in the cloud and assigns all the users who requested the sip. Users need to send a second command to request the sip information, which tell the user whether the SIP is ready

```
verification(request):
  sip_name = get_sip_name(request)
  uSet = get_users(request)
  token = get_token(request)
  user = get_user(token)

  /* user who initiate the request must be in uSet */
  if uSet(user) != true
    return false

  /* users in uSet must have domain admin role */
  for u in uSet:
    if role(u) != domain_admin
      return false

  /* only one domain admin user is allow for each domain */
  for u1, u2 in uSet:
    if u1.home_domain = u2.home_domain
      return false

  /* check if the request already exist in database */
  /* connect keystone database and get sipinfo table */
  sipinfo = connect.database(keystone).sipinfo
  if sipinfo.search(request) == false:
    /* write the request to table sipinfo */
    sipinfo.add(request)
    /* mark user has sent the request */
    sipinfo.update(request, user)
  else:
    /* mark user has sent the request */
    sipinfo.update(request, user)

  /* check mark */
  if sipinfo.search(request, mark) == uSet
    if sid(uSet.domains) exists:
      issue CreateSip(sip_name)
    else:
      issue CreateSid(default_sid_name)
      issue CreateSip(sip_name)
```

Figure 9: Algorithm for user-SIP verification process

to access. After the process of creating a SIP is complete, users can ask for Keystone to get a scoped token to access the sip. With the scoped token, users can share information and resource inside the sip. In Figure 10, we give the sequence of commands that a user issues to create and access a sip.

5.3 Database

OpenStack uses MySQL database as the storage of all meta data. Keystone has one database to store all meta data that the server needs. To store meta data for SID/SIP functions, we need to add one table "sipinfo" to hold meta data for all SIP initiation requests sent by users. Every time a SIP creation request is issued, the sipinfo table stores the request as a record in the table. The record will be removed when the SIP/SID is deleted. The record should at least include SIP name, all requesting users, and information for each requesting user.

6. DISCUSSION

There are several reasons why we need to use SIDs and SIPs instead of normal domains and projects for a collaboration group. First, users are not allowed to execute normal domain level operations like create, delete and update. These are done only by the cloud admin. Assume in a community cloud, where each domain represent an organization, a domain admin's permission is constrained inside its domain rather than over the domain level. A SID is a special shared domain outside organizations' domains. By separating SIDs

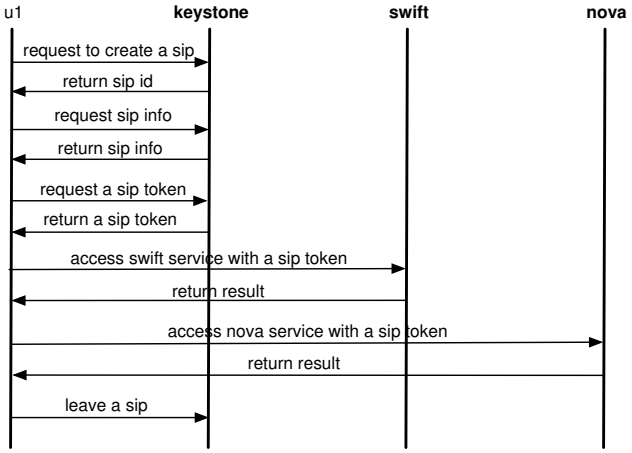

Figure 10: SIP requests sequence

from normal domains, we are able to change permissions to allow a domain admin to have complete control over SIDs. Second, the policy are built upon entities. We cannot build different access control policies over the same entity. If we use the normal domains and projects instead of SIDs and SIPs, we have to follow the policy defined for normal domains and projects. Third, the model we proposed is just a very basic model to start. In future work we are planning to have more fine-grained access control over SIDs and SIPs, which would be quite different than normal domains and projects, so we need to differentiate these two concepts and entities.

As a secure IARS protocol, our initial purpose is to set up an isolated secure space (SID/SIP) for collaborations among organizations. Currently, we haven't had an complete threat model for our protocol. It would be one of our future works. However, because of the approach we set up our protocol, some security issues are guaranteed naturally. For instance, random users won't be able to join the SID/SIP, even the SID admin can only join users from his own domain. If a user inside a SID happens to be compromised, he may have bad influence over some data copy inside the SID, but not the original data from other members' domains. With original data's safety guaranteed, a SID/SIP can be rebuilt any time. Basically, any SID admin cannot create a SIP without being noticed by other SID admins, since all the SID administrative users have equal permission inside a SID. As a SID admin, he can check any SIP inside the SID, regardless of whether he created it or not. To delete important data in a SID, for instance a SIP, the members have to have the agreement either online or offline. If one member failed to send the delete request, it is allowed to send it again. We don't consider the case in which one member refuse to execute the delete operation. In case of incident response, all the members have to have the agreement of doing any operations inside a SID.

7. RELATED WORK

The concept of information sharing is not new to security research community. In traditional systems, various access control and authorization solutions for sharing information

have been proposed. In paper [9] and [8], the author suggested approaches to support collaboration in distributed systems. In other paper like [3], the author proposed coalition models from policy and administrative perspectives. In paper [6], the author proposed a framework for g-SIS, which gives fine-grained access control over the collaboration group from administrative and operational perspectives. The difference between our model with those models are: first, we proposed our model in a IaaS cloud environment rather than distributed systems, where cloud system facilitate the collaboration in terms of unified user and role set and infrastructure for all the tenants; second, we don't give the collaboration group the direct access over the original data and resources in the organization which they did in paper [9] and [8], instead, we transfer copies to the collaboration group; third, we don't use a separate Community Authorization Service(CAS)[8] to manage the access control policies for the collaboration group, instead, we utilize the setting of roles, users and policies of the cloud to facilitate the access control over the collaboration group. Out approach is more like the model proposed in [6], we heritage the group-centric concept and mutated it in a IaaS cloud environment.
In the context of cloud, in [10], the author proposed trust relationships established between tenants to facilitate sharing. It makes outsourcing easy to implement by simply adding trust relationships among tenants in cloud. It is more like the way traditional system deal with collaboration, it give the expedient users direct access over the resources, which in our case we try to avoid. In paper [11], the author introduced a design and implementation of cloud-based assured information sharing system in SaaS. Recently, Microsoft unveiled a new intelligence-sharing platform: Interflow, which is a PaaS clould-based system for sharing attack information among organizations. It is lacking of IaaS cloud-based information and resource sharing both in the literature and industry. Our approach is built in IaaS cloud, and aim to provide a formal model and enforcement guide for implementation in real environment.

8. CONCLUSION AND FUTURE WORK

In this paper, we have developed various models for information and resource sharing in IaaS cloud using OpenStack as a reference platform. For the future work, first, we plan to implement the complete operation sets specified in our administrative and operational models in OpenStack. Second, we plan to harden the implementation to prevent overt information flow. For instance, the VMs in a SIP will need to be restricted in terms of network access so that malicious software are unable to move information out of the SIP in an uncontrolled manner. Finally, we plan to investigate fine-grained access control within a SIP, and collaboration issues that arise when tenants belong to different cloud service providers.

9. ACKNOWLEDGMENTS

The authors would like to acknowledge the support of the LMI Research Institute's Academic Partnership Program. The authors would also like to thank LMI's Systems Development group for their guidance and expertise regarding secure information sharing in the government sector, which helped supplement and guide this project.

10. REFERENCES

[1] http://aws.amazon.com/.
[2] http://openstack.org.
[3] E. Cohen, R. K. Thomas, W. Winsborough, and D. Shands. Models for coalition-based access control (cbac). In *Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 97–106. ACM, 2002.
[4] K. Harrison and G. White. Information sharing requirements and framework needed for community cyber incident detection and response. In *Homeland Security (HST), 2012 IEEE Conference on Technologies for*, pages 463–469, Nov 2012.
[5] K. Harrison and G. B. White. Anonymous and distributed community cyberincident detection. *IEEE Security and Privacy*, 11(5):20–27, 2013.
[6] R. Krishnan, R. Sandhu, J. Niu, and W. Winsborough. Towards a framework for group-centric secure collaboration. In *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*, pages 1–10. IEEE, 2009.
[7] P. Mell and T. Grance. The nist definition of cloud computing. *NIST Special Publication*, pages 800–145, September 2011.
[8] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on*, pages 50–59. IEEE, 2002.
[9] D. Shands, R. Yee, J. Jacobs, and E. J. Sebes. Secure virtual enclaves: Supporting coalition use of distributed application technologies. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, volume 1, pages 335–350. IEEE, 2000.
[10] B. Tang and R. Sandhu. Exteding openstack access control with domain trust. In *In Proceedings 8th International Conference on Network and System Security (NSS 2014)*, October 15-17 2014.
[11] B. Thuraisingham, V. Khadilkar, J. Rachapalli, T. Cadenhead, M. Kantarcioglu, K. Hamlen, L. Khan, and F. Husain. Cloud-centric assured information sharing. In *Intelligence and Security Informatics*, pages 1–26. Springer, 2012.
[12] J.-W. Wang and L.-L. Rong. Cascade-based attack vulnerability on the us power grid. *Safety Science*, 47(10):1332–1336, 2009.