Domain-based Virtualized Resource Management in Cloud Computing

Dongwan Shin and Hakan Akkan
Secure Computing Laboratory
Department of Computer Science and Engineering
New Mexico Tech
Socorro, NM 87801
Email: {doshin, hakkan}@nmt.edu

***Abstract*—Cloud computing has drawn much attention in recent years. One of its delivery models, called infrastructure as a service (IaaS), provides users with infrastructure services such as computation and data storage, heavily dependent upon virtualization techniques that offers benefits such as elasticity and cost efficiency. Most of current IaaS service providers have adopted a user-based service model, where users are directly mapped to virtualized resources that they want to use and they are charged based on usage. Hence, user and resource management are centralized and easily administered at the IaaS provider. However, this also results in the lack of support for scalable management of users and resources, organization-level security policy, let alone flexible pricing model. Considering the increasing popularity of cloud computing, there is a strong need for a more scalable and flexible IaaS model, along with a more fine grained access control mechanism. In this paper we propose a domain-based framework for provisioning and managing users and virtualized resources in IaaS to address this issue. Specifically, an additional layer called domain is introduced to the user-based service model, and the domain layer facilitates the de-centralization of user and virtualized resource management in IaaS. The cloud service provider is able to delegate its administrative works to domains, and domains manage their users and virtualized resources allocated from the cloud service provider. Our framework provides benefits such as scalable user/resource management, domain-based advanced policy support, and flexible pricing.**

***Index Terms*—cloud computing; IaaS; role-based;**

I. Introduction

Cloud computing is a new type of computing, which enables convenient, on-demand access to computing resources. Though its definitions, attributes, and characteristics are varied, it is rapidly emerging and diversified, and is being adopted as a new potential infrastructure for enterprise, government, and academic computing. In general, It is considered as a computing model that promotes availability of computing resources, which can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. Typical examples of cloud computing include office applications migrated to the Internet such as Google Docs and enterprise computing & storage service such as Amazon EC2 & S3, Google Apps, Salesforce's Cloud Platform, and Microsoft's Azure [2], [3], [4], [5].

With major companies like Google, Amazon, IBM, and Microsoft all currently at the forefront of the movement toward cloud computing, the federal and state governments have also shown keen interest in cloud computing; for instance, the U.S. Census Bureau is using Salesforce's cloud to manage the activities of about 100,000 partner organizations across the country; the Defense Information Systems Agency (DISA) has a private cloud within its data centers which is providing human resource management services to both U.S. Army and Air Force; and NASA Ames Research Center recently announced the development and deployment of a cloud computing infrastructure called NEBULA [6], to provide high-capacity computing and storage services by using a virtualized and scalable approach to achieve cost and energy efficiencies.

In addition to satisfying various computing needs from different groups of users, cloud computing provides some benefits from the perspective of computer security such as centralized data management (thereby reducing data leakage and providing monitoring benefits), password assurance, and security testing. However, there are still critical challenges in this computing paradigm demanding more advanced mechanisms for protecting data and applications in private, public, and hybrid clouds; for instance, they include cloud data security (confidentiality, integrity, availability) in clouds, data ownership issues, and protection of virtualized resources, to name a few. In this paper, we are motivated to investigate a flexible, decentralized, and policy-driven approach to protecting virtualized resources.

One of the delivery models of cloud computing, called infrastructure as a service (IaaS), provides users with infrastructure services such as computation and data storage, and it is heavily dependent upon virtualization techniques. Specifically, infrastructure resources such as operating systems and networking are provisioned on virtual platforms and provided as an on-demand service to users, and this offers benefits such as elasticity, and cost/energy efficiency. However, most of current IaaS service providers have a user-based service model, where users are directly mapped to virtualized resources that they want to use and they are charged based on usage. Therefore, the administrative jobs of managing users and virtualized resources are centralized and easily performed at the IaaS provider. However, this also results in the lack of support for scalable management of users and resources, organization-level security policy, let alone flexible

COLLABORATECOM 2010, October 9-12, Chicago, USA
Copyright © 2011 ICST
DOI 10.4108/icst.trustcol.2010.3

pricing model. Considering the increasing popularity and wide adoption of cloud computing, there is a strong need for a more scalable and flexible IaaS model, along with a more fine grained access control support.

In this paper we propose a domain-based framework for provisioning and managing users and virtualized resources in IaaS to address the issue. Specifically, an additional layer called domain is introduced to the current user-based service model, and the domain layer facilitates the de-centralization of user and virtualized resource management in IaaS. The cloud service provider is able to delegate its administrative works to domains, and domains manage their users and virtualized resources allocated from the cloud service provider using role-based security policy. Our framework provides benefits such as scalable user/resource management, domain-based advanced policy support, and flexible pricing.

The rest of this paper is organized as follows. Section 2 discusses background and related work. Section 3 describes our approach to domain-based user/resource management, followed by the discussion of our design and implementation in Section 4. Section 5 concludes this paper with a discussion on our future research direction.

II. Background and Related Work

In this section we first discuss the general characteristics of cloud computing and its three different delivery models. Then, we describe the support of role-based access control (RBAC) as a way to ease the administration and management of user privileges in different cloud computing platforms.

A. Cloud Computing and Its Delivery Models

The general characteristics of cloud computing include on-demand service, ubiquitous access, location independence, rapid elasticity, and measured service [1]. To support these characteristics, cloud computing generally consist of three foundational components and three optional, applied components depending on its deployment models. The three foundational components are essentially those that are needed to build a collection of physical/virtualized, distributed servers for providing cloud services. They are 1) hardware and facilities, 2) software kernel, and 3) virtualization, as shown in the lower part of Figure 1. The three applied components characterize and classify the services and applications of cloud computing. They are 1) computation and storage resource, 2) cloud software development platform, and 3) cloud software application, as shown in the upper part of Figure 1. The computation and storage resource component concerns the deployment model called Infrastructure as a Service (IaaS), the cloud software development platform component pertains to another deployment model called Platform as a Service (PaaS), and the cloud software application component is related to the deployment model called Software as a Service (SaaS). Their differences are as follows; first, in the SaaS model, the cloud consumer can use the cloud provider's applications running on a cloud infrastructure which are accessible through a client interface such as a web browser. Typical examples of this type are Google Docs and Salesforce applications [4], [7]; in the PaaS model, the consumer can deploy onto the cloud infrastructure consumer-created applications using programming languages and tools supported by the provider. Some of the examples of this type include Google App Engine and Windows Azure [3], [5]; and, lastly in the IaaS model, the consumer provisions processing, storage, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software. Amazon EC2 & S3 and Eucalyptus are the examples of this type [2], [8]. The successful implementation of cloud infrastructure requires that both foundational and applied components work together seamlessly.

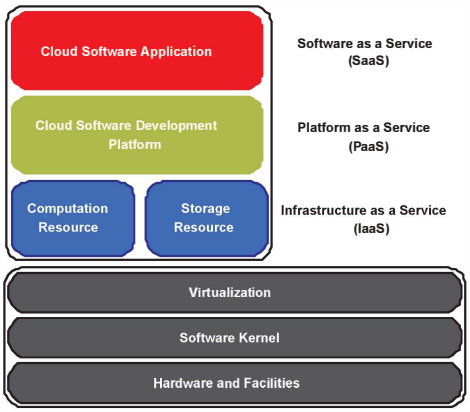

Fig. 1. Cloud computing components and delivery models

B. Role-based Support in Current IaaS Platforms

Virtualization is a powerful and indispensable mechanism for cloud computing, and especially it is true for the IaaS delivery model. Virtualized resources are provisioned and provided to users as a on-demand service. Hence, considering the increasing number of cloud users, a flexible and policy-driven decentralized management and authorization mechanism for protecting virtualized resources is essential for the success of cloud computing in general, and IssS in specific.

Role-based security policy [9], [10] has attracted considerable attention in computer security communities over the last two decades, and it has grown to be a proven solution for managing access control in a simple, flexible, and convenient manner. The basic idea behind role-based access control (RBAC) is to use the intermediary concept called *role* to provide an indirection mechanism between users and permissions. This indirection mechanism helps reduce errors in user/permission management, support advanced features such as constraints and role hierarchy, and allow for convenient user/permission management schemes such as role-based administration and delegation [11], [12], [13], [14]. RBAC has been successfully implemented in many commercial systems including different flavors of operating systems, database systems, enterprise-based web applications. It has been also used and implemented to support the decentralization of access control management [15], [16], [17].

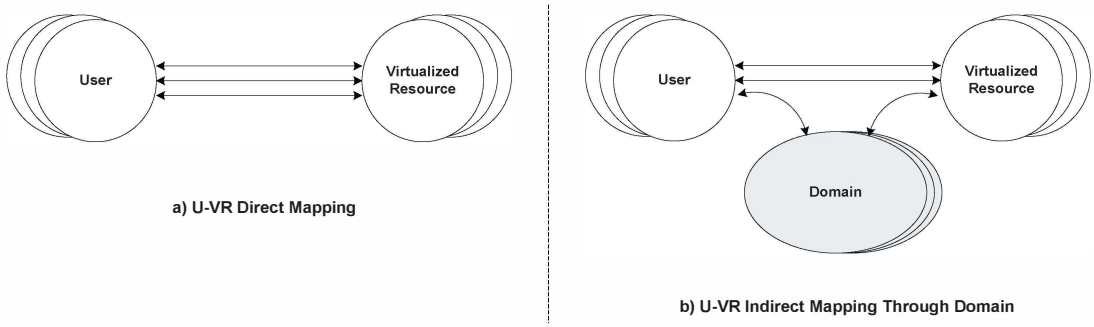

Fig. 2. Our Approach

Unfortunately, most of existing IaaS platforms do not support the notion of grouping or categorizing users, and thus there is no support for RBAC. Each user in the platforms is considered to be independent of others and is provided a root privilege for the virtualized resources that are requested. Therefore, this flat hierarchy rooting from the direct mapping between user and virtualized resources naturally lacks the support for the advanced features that RBAC offers.

Amazon's EC2 platform [2] is the most popular IaaS service. Amazon also provides many accompanying services such as storage (S3, ESB) and database (Amazon SimpleDB, Relational Database Service - RDS). However, there is no support for RBAC in these services. The only type of access control is restriction on operating system (OS) images. A user can upload an OS image and attach an access control list (ACL) to it by specifying which users can use that image. An image can also be made available to the public. Eucalyptus [8] is another popular IaaS platform which was started as an academic research and then converted into an open source project. It has been designed to be an exact clone of the Amazon EC2 in functionality. Hence, it is equipped with the same access control mechanism as EC2 does. However, since its sources are open, it is possible to extend the platform to support different features and our approach is based on this.

NASA's IaaS platform called Nebula [6] was initially based on Eucalyptus but it has been rewritten to add its own engine Nova to address scalability issues with Eucalyptus. Recently the platform was reconfigured to support the use of roles in its access control mechanism [18]. The approach taken is twofold: first the frontal controller was connected to an LDAP server for retrieving user/role information, and second a pass/fail gate was implemented on each API call. Though this approach is similar to our approach, its support for some advanced features such as role hierarchy is not clear and more importantly there is no concept called domain in their approach, which is basically the core part of our approach that allows for domain-based administrative delegation, security, and user/resource management.

Unlike the aforementioned platforms, Windows Azure [5] is a PaaS type of service platform where users are allowed to develop applications on the Azure AppFabric to deploy them on Microsoft's datacenters. The platform also provides storage and automatic scaling/load balancing features. Applications deployed within Azure may belong to either or both of *Web* role and *Worker* role. Depending on the role, the application is allowed to perform different tasks. Hence, roles are assigned to the application, not to the user, in this platform.

III. Our Approach

In order to support a scalable, decentralized, policy-driven scheme for IaaS, we discuss a domain-based framework for managing users and virtualized resource in this section. First, we present the formal definitions of IaaS components along with the introduction to the notion of domain. Then re-definitions of some of role-based policy constructs follow.

A. IaaS Components

The component of IaaS that we are most interested in for our approach is virtualized resources such as virtual machine (VM) types based on different configurations, operating system images, ramdisk images, and networking capabilities including elastic IP addresses. These kinds of virtualized resources can be found very commonly in existing IaaS platforms, with the slightly varying degree of their abstraction.

Definition 1: Let $\mathcal{VR} = \{vr_1, ..., vr_l\}$ denote a set of virtualized resources. A virtualized resource is represented by n-tuple, where n denotes the number of different kinds of virtualized resources serviced in the IaaS platform. For instance, $vr_1 = (VMconfig_1, VMos_2, VMnet_3)$, where $VMconfig_1$ denotes a virtual machine type with 1.7 GB memory, 1 virtual core, 160 GB storage, and 32-bit platform[1]; $VMos_2$ denotes a Linux operating system; and $VMnet_3$ denotes elastic IP addresses.

The cloud user can select and use a subset of virtual resources and he needs to have a unique credential to access them; typically a pair of public and private keys is used for the credential.

Definition 2: Let $\mathcal{U} = \{u_1, ..., u_n\}$ denote a set of cloud users that can be uniquely identified. $\mathcal{UVR} = \mathcal{U} \times \mathcal{VR}$ represents the relation of cloud user-to-virtualized resource, and the function f_{UVR}: $\mathcal{U} \rightarrow 2^{\mathcal{VR}}$ maps a cloud user to a set of virtual resource.

[1]The similar configuration is called a small instance in Amazon EC2 platform.

In addition to virtualized resources and cloud users, we introduce the concept of domain, which can provide an indirection mechanism between the cloud user and virtualized resource. Adding the domain in between the cloud user and virtualized resource can offer various benefits. First, it can provide a means of decentralized, scalable management of the cloud user and virtualized resources through the delegation of administrative jobs. A domain can be delegated the authority to manage the cloud user and virutalized resource associated with the domain. Second, security policies and measures can be applied to the domain level, not the IaaS service provider level. Monitoring and auditing the usage of virtualized resources can be performed at the domain-level. Last, more various subscription types can be introduced based on domain-level contracts. Figure 2 depicts the relationship between the three. We define a domain as a single RBAC domain where role-based user/resource administration can take place.

Definition 3: Let $\mathcal{D} = \{d_1, ..., d_i\}$ denote a set of RBAC domains. $\mathcal{UD} = \mathcal{U} \times \mathcal{D}$ represents the relation of cloud user-to-domain, and the cloud user may or may not be associated with a domain. Similarly, $\mathcal{DVR} = \mathcal{D} \times \mathcal{VR}$ represents the relation of domain-to-virtualized resource, and the virtualized resource may or may not be assigned to a domain. Lastly, $\mathcal{UVR}_{\mathcal{D}}$ represents the ternary relation of cloud user-domain-virtual resource, and the function f_{UVR_D}: $\mathcal{U} \times \mathcal{D} \rightarrow 2^{\mathcal{VR}}$ maps a pair of user and associated domain to a set of virtual resource.

B. RBAC Components

Our approach is based on RBAC due to its advanced features previously discussed. It subsumes existing RBAC components, also following the conventional approach to defining them using the sets, relations, and functions. Note that some of them have been further specified or redefined for our purpose.

Definition 4: The RBAC components supported are as follows.

- $\mathsf{U} = \mathcal{U} \times \mathcal{D}$ represents the set of domain users associated with domains, and $\mathcal{U} \times \phi$ represents the set of cloud users not associated with any domain.
- P represents the set of permissions to use virtualized resources associated domains
- R and S represents the set of roles and sessions, respectively.
- UA, PA, and RH, representing the relation of user-to-role assignment, permission-to-role assignment, and role hierarchy, respectively. RH is partial order on R, written as $\preceq$.
- user: $\mathsf{S} \rightarrow \mathsf{U}$ represents a function mapping each session s_i to the single user.
- roles: $\mathsf{S} \rightarrow 2^R$ represents a function mapping s_i to a set of roles, where $\mathsf{roles} \subseteq \{\mathsf{r} | (\exists \mathsf{r}' \succeq \mathsf{r})[(\mathsf{user}(\mathsf{s}_i), \mathsf{r}') \in \mathsf{UA}]\}$ and s_i has permissions $\bigcup_{\mathsf{r} \in \mathsf{roles}(\mathsf{s}_i)} \{\mathsf{p} | (\exists \mathsf{r}'' \preceq \mathsf{r})[(\mathsf{p}, \mathsf{r}'') \in \mathsf{PA}]\}$.

C. Domain-based Management and Delegation

Our approach supports the domain-based management of users and virtualized resources through the delegation of the administrative functions and allocation of virtualized resources to each domain from the cloud service (IaaS) provider. The user and virtualized resource are associated with the domain, and the domain administrator can manage them based on their organizational security and management policies. It should be noted that this does not mean that our approach does not support the traditional user-based service model. The user can get the infrastructure service through both the direct mapping and indirect mapping as shown in Figure 2.

The cloud service provider can allocate virtualized resources to each domain based on its subscription contract. In addition, the cloud service provider delegates administrative functions related to user management, role management, and permission management. Please refer to [10] for more details on RBAC administrative functions. The access request from the user contains the user identity and requested virtualized resource, and the reference monitor can check if the user is allowed to access the virtualized resource by checking both the direct mapping and indirect mapping.

IV. Design and Implementation

In this section we discuss the design and proof-of-concept implementation of our approach. The Eucalyptus platform has been modified and extended for our purpose. As previously stated, Eucalyptus has no notion of roles for its access control mechanism. Specifically, we designed and implemented objects that represents roles, permissions, and different administrative domains within Eucalyptus; we modified web interface of Eucalyptus to enable the cloud service administrator for managing domains and associated permissions, and to enable domain administrators for managing domain users and roles; and we designed and implemented a reference monitor that checks whether the given request for the creation of virtual machines should be granted or not.

We first discuss the general architecture of Eucalyptus and then present our design and implementation.

A. Eucalyptus Architecture

Eucalyptus is organized into five components, each of which is responsible for operation of a different part of the platform:

1) Cloud Controller (CLC): The main component which governs the system and exposes a query (REST/SOAP) interface for users to communicate with the system. It also leverages a web interface for administration, user registration and retrieving user credentials.
2) Walrus: Amazon S3 equivalent part of Eucalyptus.
3) Cluster Controller (CC): Each cluster has a CC responsible for resource allocation among the nodes within that cluster.
4) Storage Controller (SC): Amazon ESB equivalent part of Eucalyptus. Each cluster has a SC that exports block storage devices over the LAN for VMs to mount them.
5) Node Controller (NC): Responsible for running/terminating virtual machines and networking of them.

In the open source distribution, the source code for CLC includes the code for Walrus and SC components. During the

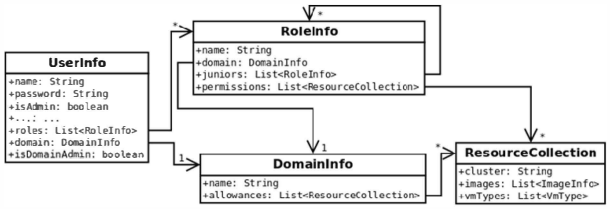

Fig. 3. RBAC Object Design

boot up, CLC marks SC and Walrus components as either local or remote and tells Mule ESB (enterprise service bus) what to do with the remote components. If they are remote, Mule ESB takes care of transportation of messages between remote hosts. The CLC is composed of a number of tightly coupled modules. They are all compiled and packaged into separate jar files and loaded by the same classloader during the boot process of the system. For example, the web administration module is compiled into the package called **eucalyptus-www.jar** and loaded/started by the bootstrapper. We have created a separate module for the RBAC component called **rbac-manager.jar** and put all the code related to roles, domains and the reference monitor in this package.

B. RBAC Component Representation

Since there is a limited number of resources in this platform, we decided to represent permissions over them as collections of resources. Among various virtual resources such as images, VM types, number of public IPs, and allowed ports, we only considered images and VM types for this prototype design and implementation. A resource collection consists of a list of images, a list of VM types, and the name of the cluster on which these images and VM types can be used. Objects that need to be assigned permissions (roles and domains) are associated with resource collection objects, as shown in Figure 3. For example, a user of the role associated with the resource collection object *(cluster="ZoneA",images=[emi-AAAAAA,eri-BBBBBB],vmTypes=[m1.medium])* can only create a virtual machine of type *m1.medium*, only in the cluster named *ZoneA*, and can use either of the images listed. In Eucalyptus, all users are treated the same. By introducing the concept of domains, we can group users and manage access control administration separately from other domains. Each user and role is associated with a domain that makes administration easier. The user to domain mapping information is stored in the **UserInfo** object by adding an extra attribute to the class. The cloud administrator can create domains and modify permissions given to them such as VM types and images. These permissions form the basis for all permissions given to roles within that domain, as shown in Figure 4. The domain administrator then assigns permissions to roles from this set of permissions. Permissions are assigned to domains per computing clusters, and this provides a finer grained access control as to"who can do what on what cluster". The class **RoleInfo** is responsible for representing the roles and the permissions associated with them. The user to role association information is stored also in the **UserInfo** object by adding an extra attribute. Permissions assigned to roles are represented as **ResourceCollection** objects for each cluster. The cloud administrator or domain administrators can select and assign permissions to roles among the permissions that are available to the domain of the role which are assigned by the cloud administrator, as shown in Figure 5.

The reference monitor is involved only in processing a VM creation request. The request message goes through several components in which the validity and acceptability of the parameters are verified. The internal messaging between components in Eucalyptus is done via Mule ESB that also includes the processing of such requests. We have configured the Mule so that a VM creation request message is passed through the reference monitor as the last stage of the validation process. Mule configuration is done in compile-time with XML files in which services and associated endpoints are configured. We also defined a new endpoint called **RolesVerifyWS** and a new service called **RolesVerify** in clc/modules/cloud/src/main/resources/eucalyptus-services.xml and clc/modules/cloud/src/main/resources/eucalyptus-verification.xml, respectively. The reference monitor receives a **VmAllocationInfo** object which contains the necessary information to run VMs such as the user ID, requested machine&kernel&ramdisk images, the VM type etc. It first retrieves the **UserInfo** object associated with the requesting user and retrieves user's roles as a list. Starting from these roles, it initiates a breadth-first-search over the role hierarchy to find all of the image, VM type, and cluster permissions. During the search, if all of the required permissions are found in one or more roles, the search is terminated and the reference monitor allows message to pass through. If the search terminates with consuming all of the role hierarchy and not finding all of the required permissions, an exception is thrown which will prevent the request from further processing and return the exception message to the client.

V. Conclusion and Future Work

In this paper we have discussed a novel approach to managing virtualized resources in cloud computing by introducing the notion of domain and injecting a role-based security policy support into the IaaS service model. Specifically, our approach adds an additional layer of domain to the current user-resource direct mapping mapping, and the cloud service provider delegates its administrative functions to each domain, and the domain administrator further manages users and allocated virtualized resources. This framework provides benefits such as domain optimized resource management, domain-based advanced policy support based on role, domain-based security log analysis, and better pricing model (based on not just user, but group/domain). Finally, we discussed how to design and implement a proof-of-concept prototype by modifying an existing IaaS framework called Eucalyptus.

Our immediate future work includes the investigation on how to support advanced role-based policies such as separation of duty (SoD).

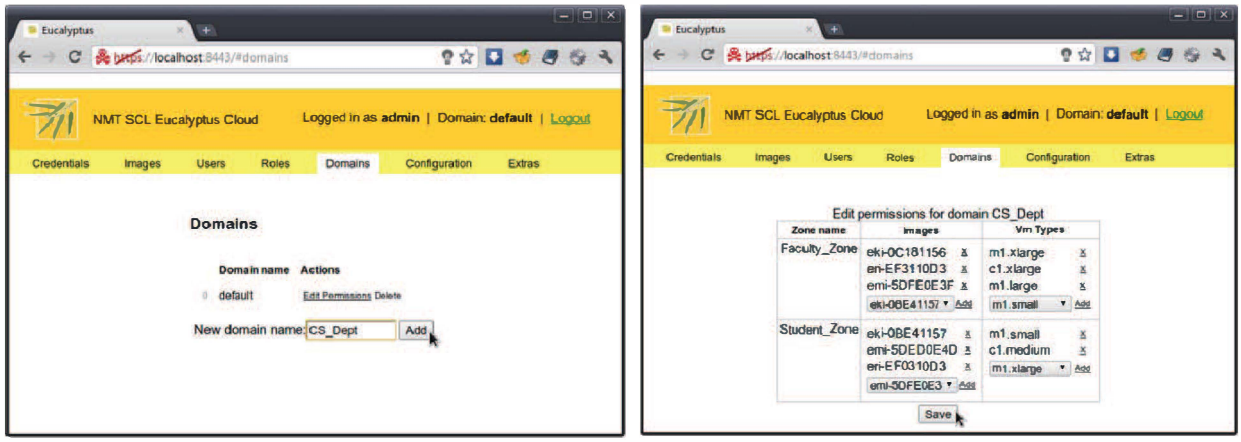

Fig. 4. The web interface for managing domains and permissions

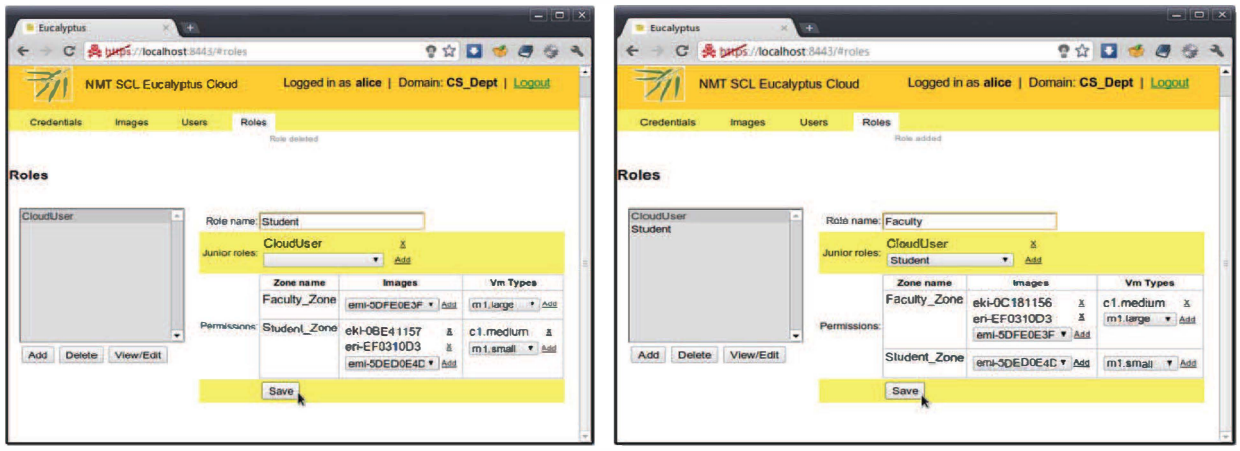

Fig. 5. The web interface for managing roles and permissions

ACKNOWLEDGMENT

This work was supported at the Secure Computing Laboratory at New Mexico Tech by the grant from the National Science Foundation (NSF-IIS-0916875).

REFERENCES

[1] NIST, "Nist working definition of cloud computing," http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, Tech. Rep., 2009.

[2] *Amazon Elastic Compute Cloud and Simple Storage Service*, http://aws.amazon.com.

[3] *Google Apps*, http://www.google.com/a.

[4] *Salesforces the Sales Cloud*, http://www.salesforce.com/crm/sales-force-automation/.

[5] *Windows Azure Platform*, http://www.microsoft.com/azure/default.mspx.

[6] *NEBULA: NASAs Cloud Computing Platform*, http://nebula.nasa.gov/.

[7] *Google Doc*, http://docs.google.com/.

[8] *Eucalyptus Open Source*, http://open.eucalyptus.com/.

[9] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, February 1996.

[10] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli, "Proposed NIST standard for role-based access control," *ACM Transactions on Information and System Security*, vol. 4, no. 3, August 2001.

[11] D. Shin, G.-J. Ahn, S. Cho, and S. Jin, "On modeling system-centric information for role engineering," in *Proceedings of 8th ACM Symposium on Access Control Models and Technologies*, Como, Italy, June 2-3 2003.

[12] G.-J. Ahn and R. Sandhu, "The RSL99 language for role-based separation of duty constraints," in *Proceedings of 4th ACM Workshop on Role-Based Access Control*. Fairfax, VA: ACM, October 28-29 1999, pp. 43–54.

[13] L. Zhang, G.-J. Ahn, and B. Chu, "A rule-based framework for role-based delegation," in *Proceedings of 6th ACM Symposium on Access Control Models and Technologies*, Chantilly, VA, May 3-4 2001, pp. 153–162.

[14] E. S. Barka and R. S. Sandhu, "Framework for role-based delegation models," in *Proceedings of 16th Annual Computer Security Application Conference*, New Orleans, LA, December 2000.

[15] N. Dimmock, A. Belokosztolszki, D. Eyers, J. Bacon, and K. Moody, "Using trust and risk in role-based access control policies," in *Proceedings of 9th ACM Symposium on Access Control Models and Technologies*, Yorktown, NY, June 2004.

[16] ITU, *ITU-T Recommendation X.509. Information Technology: Open Systems Interconnection - The Directory: Public-Key And Attribute Certificate Frameworks*, 2000, iSO/IEC 9594-8.

[17] D. Shin, G.-J. Ahn, and S. Cho, "Role-based EAM using x.509 attribute certificate," in *Proceedings of Sixteenth Annual IFIP WG 11.3 Working Conference on Data and Application Security*, Cambridge, UK, July 29-31 2002.

[18] *RBAC support for Nebula*, http://nebula.nasa.gov/blog/2010/jun/nebulas-implementation-of-role-based-access-contro/.