Toward Group-Based User-Attribute Policies in Azure-like Access Control Systems

Anna Lisa Ferrara[1], Anna Squicciarini[2], Cong Liao[2], Truc L. Nguyen[1]

[1] Computer Science Department,
University of Southampton
{al.ferrara,tnl2g10}@soton.ac.uk
[2] Information Sciences and Technology,
Pennsylvania State University
{acs20,cl13}@psu.edu

Abstract. Cloud resources are increasingly pooled together for collaboration among users from different administrative units. In these settings, separation of duty between resource and identity management is strongly encouraged, as it streamlines organization of resource access in cloud. Yet, this separation may hinder availability and accessibility of resources, negating access to authorized and entitled subjects. In this paper, we present an in-depth analysis of group-reachability in user attribute-based access control. Starting from a concrete instance of an Access Control supported by the Azure platform, we adopt formal verification methods to demonstrate how it is possible to mitigate access availability issues, which may arise as per-attribute criteria groups are deployed.

1 Introduction

The increasing adoption of cloud computing has drawn attention to its security challenges [12, 7]. This has led major cloud providers to incrementally add security features for increased costumers' confidence [1, 12]. Access control, in particular, is acknowledged as one of the fundamental security features required to avoid unauthorized access to sensitive data and protect organizations assets.

Among prominent cloud providers, Microsoft Azure has stood out for its recent focus on security technologies for cloud resources [6]. In particular, Microsoft currently boasts features such as an extensive Security Center (backup features, dependability etc), Automatic Monitoring, and support of Role-based Access Control (RBAC). The deployment of RBAC in Azure has been met by much praise, by both security researchers [23] and by practitioners [2].

The access control mechanism supported by Azure not only provides a suite of easy-to-use ways to implement RBAC, but also is integrated with the native identity management systems supported by Microsoft through Active Directory (referred to as AAD). One of the many peculiar features resulting from the integration is the ability to specify security groups, which can be used for role and privilege assignment. Security groups can be populated manually, on a per-user basis, or through dynamic triggers (aka dynamic groups).

Thus, Azure RBAC roles can be granted to Azure AD security groups with rule-based membership - effectively achieving *user attribute-based access control*. ABAC is deemed more desirable by researchers and practitioners since it provides flexibility and scalability for securely managing access to resources, particularly in collaborative environments.

Yet, implementing user attribute-based access control brings a set of new interesting challenges [1], particularly when dynamic security groups are used and managed by resource managers (and not identity managers).

Developers have no mean to check whether their attribute-based policies for resource (or resource group) access meet intended requirements and administrative units' policies. Identifying the right set of rules for security groups is a complicated and error prone process that should take into account resources' access control policies as well as user-attributes assignment policies.

Accordingly, automatic tools inspecting policy-compliant resource access in cloud - are desirable and needed to assist designers in policy development in order to avoid availability issues. Resource availability and accessibility are among the top security concerns of cloud adopters to date.

In this paper we propose to analyse user attribute-based policies developed under the Azure-like platform. Given dynamic groups implementing user-attribute policies to resources and administrative units policies, we first model the user attribute-based access control system as a state transition system that evolves via administrative actions to modify the user attributes and the user-groups membership; then we study the "group-reachability problem".

The Group-reachability problem. Given a subscription policy with unboundedly many users, a finite set of dynamic groups, a finite set of administrative units' policies, an initial configuration of the user attribute-based access control system, and a target security group *goal*, is there a reachable configuration of the access-control system where some user is assigned group *goal*?

We choose the GURA framework [9] to express user attributes assignments in Azure AD since it enables an elegant and succinct representation of user attributes administrative policies via RBAC. Our contributions include:

1. A proposal to develop user attribute-based access control policies for Azure management via a combination of AD dynamic security groups, GURA policies and the version of RBAC supported by Azure.
2. A proposal to analyse the group-reachability problem via security analysis of Administrative RBAC [3, 16]. First, we show that the reachability problem in Azure-like AC systems is equivalent to the attribute-reachability problem in GURA systems [9]. We then reduce it to the role-reachability problem in ARBAC so that existing tools and techniques can be leveraged to address the group reachability problem.

We note that while we provide a concrete approach to verify user-attribute policies' for Azure-like platforms, our theoretical results on group reachability are agnostic to any implementation, as they could be applied in any user-attribute based access control system that supports dynamic groups.

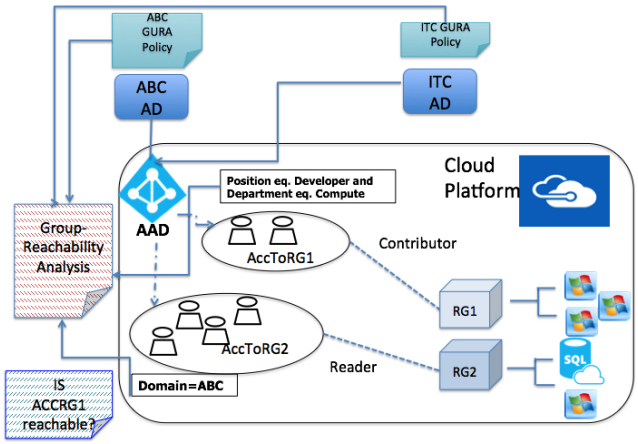

Fig. 1. Example scenario

2 Motivating example

We now present our problem statement through a running example (inspired from discussions of existing cloud adopters). Our goal, through our proposed architecture is to ensure secure enforcement of organizations' access policies while ensuring *group-reachability*, that is to say, verify whether there is a reachable configuration where some group of users can eventually be granted access to a given resource or resource group), given a) the available resources offered by a representative cloud provider such as Azure and b) the administrative security requirements imposed by cloud collaborators and partners.

Consider a large-scale production project, `InsightIT`, involving multiple companies. Assume all such organizations share a production environment through an Azure Subscription (see Figure 1). For simplicity, we focus on two of such collaborative units, ABC and ITC. ABC needs to set up a production environment for their client with PaaS Services like Azure App Service, Azure SQL Database (PaaS), Azure Storage. Assume that ABC employees are organized in 4 departments: storage, compute, networking, security. Each has corresponding responsibilities. ITC is doing development or testing work on some applications originally deployed by ABC. The applications are to be kept confidential to the public and the other partner collaborators, and ITC should only be given access to the environments needed for deployment and collaboration.

In order to enable the ITC Development team and the ABC on a single cloud platform (i.e. Azure Subscription), it is relatively easy (and recommended per Microsoft best practices [14]) to configure these PAAS services into separate containers, referred to as resource groups (see Figure 1). ABC can configure its resources within two resource groups, `RG1` and `RG2`, with `RG2` including the company's Customer Databases and virtualized environment. `RG1` can be used as a container for ITC and ABC collaborative effort. Access to `RG2` and `RG1` must

be controlled carefully to comply with each organization's administrative policy (formalized in the GURA framework as discussed in Section 4). Access is granted to dynamic groups. These groups are specified by resource administrators (or resource owners) according to the project's access needs and the anticipated organizational workflow. In particular, access to resource groups can be granted by assigning designated roles to dynamic groups (discussed in Section 3.1). This enables automated access to resource groups per user-attributes. Let's refer to these dynamic groups as `AccToRG1` and `AccToRG2`, respectively - and assume that membership is granted according to specific per-attribute conditions.

In addition to the resources access requirements, both ABC and ITC maintain distinct internal administrative policies, to be honored during the collaboration and used as a guidance for policy and access decisions. Typically, administrative policies include several conditions- both written (e.g. due to law compliance) and verbal, with no official documentation (reflecting the internal practices of an organization and its culture, e.g. interns are not allowed access to internal servers). For instance, part of ABC administrative policy mandates employees to have rotational periods in various departments. Testers are assigned to the Network department. Further, resources pooled for collaboration with partners may only be accessible to some ABC developers. Further, ABC domain is only granted to permanent workers. On the other hand, ITC, per its internal HR structure, maintains a policy stating that Tester is the official position of their Computing department.

Given a set of administrative policies like the ones above, checking consistency of group-based policies by simple inspection may be difficult. For instance, let's assume the `RG1` owner, in an effort to accommodate both ITC and ABC administrative policies, decides that a user is assigned to a security group to access to `RG1` only if he/she is both part of the Compute department and has a Tester position within the organization. However, this combination of attributes won't enable any ABC user to join the security group, as ABC Testers are mandated membership to Networking department as part of their rotational period. Similarly, ABC employees may be denied access to `RG2` if the rule mandates only the ABC users to join the group, unaware of temporary domains assigned new employees or interns. These issues may be referred to as instances of the *group-reachability* problem.

3 Preliminaries

3.1 Access Control in Azure

Microsoft Azure is one of the dominant cloud IaaS platforms for enterprises. Azures core features include compute, storage, database and networking. An Azure account is referred to as *subscription*, and is overseen by one or more super admins, or owners. A subscription is essentially a container of the owner's Microsoft cloud environment. Azure access control include:
Azure Active Directory (AAD): provides a full suite of identity management capabilities, along with single sign-on (SSO) access to cloud SaaS Applications.

Users information within AAD is stored along with their attributes. Users can be organized in groups. Group owners or administrators can add users one-by-one, or in a criteria-based fashion. In the latter case, groups are referred to as dynamic groups. Users can be granted access to resources through individual role assignment or through group-role assignment.
Resources: are cloud assets which populate the subscriptions (e.g. virtual machines, databases, storage). Resource Groups (RG) are containers to segregate workloads and resources that require different access settings. Access to resources is assigned to RGs or to individual resources.
Roles (R): Users are assigned to a resource group with roles to get permissions to access to cloud resources. Azure supports a large set of built-in roles, and supports easy to add customized roles.

3.2 Generalized User Role Assignment

GURA model [9]: Let U be a finite set of users and $ATTR$ be a finite set of attributes. Each attribute is a function that takes users as input and returns a value from the attribute scope. Attributes can be atomic-valued or set-valued. An atomic-valued attribute will return a single value while a set-valued attribute will return a subset of values within its defined scope. Moreover, let AR be a finite set of administrative roles. Administrators are allowed to change attributes of a user according to some preconditions. A precondition is a logical formula expressed over user attributes that evaluates to **true** or **false**. Formally, administrative rules are tuples in the following relations where C is a set of preconditions:

- for each atomic-valued attribute $att \in ATTR$, permission to assign a particular value to att of a user is specified as: $assign \subseteq AR \times C \times ATTR \times SCOPE_{att}$. The meaning of an assign tuple $(admin, c, att, val) \in assign$ is that a member of the administrative role $admin \in AR$ can give value val to the atomic-valued attribute att of a user whose current user-attributes assignment satisfies precondition c, where $SCOPE_{att}$ is the scope of the attribute $att \in ATTR$;
- permission to add a particular value to the set-valued attribute att of a user is specified as: $add \subseteq AR \times C \times ATTR \times SCOPE_{att}$. The meaning of an add tuple $(admin, c, att, val) \in add$ is that a member of the administrative role $admin \in AR$ can add value val to the set of values for attribute att of a user whose current user-attributes assignment satisfies precondition c.
- permission to revoke the assignment of a value from any atomic-valued or set-valued attribute of a user is specified as: $delete \subseteq AR \times SCOPE_{att}$. The meaning of a tuple $(admin, val) \in delete$ is that value val can be deleted from attribute att of any user.

Management of membership in administrative roles is outside the scope of $GURA$. Thus, we assume that each role in AR always contains some administrator.

GURA Systems: A GURA system is a state transition system that evolves via administrative actions to modify user attributes. Formally, a GURA system

is a tuple $\mathcal{S} = \langle U, AR, ATTR, UATTR, assign, add, delete\rangle$ where $UATTR \subseteq U \times ATTR \times SCOPE_{att}$ is the user-attribute relation.

A *configuration* of $\mathcal{S}$ is any user-attribute assignment relation $UV \subseteq U \times ATTR \times SCOPE_{att}$. A configuration UV is *initial* if $UV = UATTR$.

Given two $\mathcal{S}$ configurations UV and UV', there is a *transition* from UV to UV' with rule $m \in (can\_assign \cup can\_add \cup can\_delete)$, denoted $UV \xrightarrow{m} UV'$, if one of the following holds:
[**assign move**] $m = (admin, c, att, val)$, the user-attribute assignment relation of a user u satisfy precondition c, att is an atomic-valued attribute, and $UV' = (UV \setminus \{(u, att, val') | (u, att, val') \in UV\}) \cup \{(u, att, val)\}$;
[**add move**] $m = (admin, c, att, val)$, the user-attribute assignment relation of a user u satisfy precondition c, att is a set-valued attribute, and $UV' = UV \cup \{(u, att, val)\}$;
[**delete move**] $m = (admin, att, val)$, and $UV' = UV \setminus \{(u, att, val)\}$;
A *run* of $\mathcal{S}$ is any finite sequence of $\mathcal{S}$ transitions $\pi = c_1 \xrightarrow{m_1} c_2 \xrightarrow{m_2} \ldots c_n \xrightarrow{m_n} c_{n+1}$ for some $n \geq 0$, where c_1 is an *initial* configuration of $\mathcal{S}$. An $\mathcal{S}$ configuration c is reachable if c is the last configuration of an $\mathcal{S}$ run.

Definition 1 (Attribute-reachability Problem). *For any pair* $(att, val) \in ATTR \times SCOPE_{att}$, (att, val) *is* reachable *in* $\mathcal{S}$ *if there is an* $\mathcal{S}$ *reachable configuration* UV *such that* $(u, att, val) \in UATTR$, *for some* $u \in U$. *Given a GURA system* $\mathcal{S}$ *over the set of attributes* $ATTR$ *and a target pair* $(\texttt{att}, \texttt{val}) \in UV \times SCOPE_{att}$, *the* attribute-reachability problem *asks whether* $(\texttt{att}, \texttt{val})$ *is reachable in* $\mathcal{S}$.

Restricted $GURA_1$ system: A restricted $GURA_1$ system ($rGURA_1$) is a $GURA$ system where a precondition can be expressed as a conjunction of literals, where each literal is either in positive form ℓ or in negative form $\neg\ell$, for some pair $\ell = (att, val) \in ATTR \times SCOPE_{att}$. A user u satisfies a positive pair (att, val) if $(att(u) = val)$ evaluates to **true**. On the contrary, she satisfies a negative pair if $\neg(att(u) = val)$ evaluates to **true**.

Since Azure AD policies always allow some administrator to revoke an attribute value, we assume that $rGURA_1$ policies contain a delete rule for each attribute-value.

3.3 Administrative Role Based Access Control

An RBAC policy is a tuple $\langle U, R, P, UA, PA\rangle$ where U, R and P are finite sets of *users*, *roles*, and *permissions*, respectively, $UA \subseteq U \times R$ is the *user-role assignment* relation, and $PA \subseteq P \times R$ is the *permission-role assignment* relation. A pair $(u, r) \in UA$ means that user u is a member of role r. Similarly, $(p, r) \in PA$ means that members of role r are granted the permission p.

The ARBAC-URA policy allows to change the user-role assignment UA by means of assignment/revocation rules carried out by administrators which are organized in a set AR of administrative roles.

Administrators are allowed to change roles of a user according to a precondition. A *precondition* is a conjunction of literals, where each literal is either in positive form r or in negative form $\neg r$, for some role r in R. A precondition can be partitioned in two sets denoted Pos and Neg, respectively corresponding to the set of roles that appear in positive and negative form in the precondition.

Permission to assign users to roles is specified as: $can\_assign \subseteq AR \times 2^R \times 2^R \times R$. The meaning of a can-assign tuple $(admin, Pos, Neg, r) \in can\_assign$ is that a member of the administrative role $admin \in AR$ can make a user whose current role memberships satisfies the precondition (Pos, Neg), a member of $r \in R$. In the rest of the paper we assume that $Pos \cap Neg = \emptyset$.

Permission to revoke users from roles is specified as: $can\_revoke \subseteq AR \times R$.

A tuple $(admin, r) \in can\_revoke$ means that a member of the administrative role $admin \in AR$, can revoke the membership of a user from a role $r \in R$.

ARBAC-URA Systems: An ARBAC-URA system is a state transition system that evolves via administrative actions to modify user role assignments. For a formal definition see [4].

Definition 2 (Role-reachability Problem [4]). *For any role $r \in R$, r is* reachable *in an ARBAC-URA system $\mathcal{S}$ if there is an $\mathcal{S}$ reachable configuration UR such that $(u, r) \in UR$, for some $u \in U$. Given an URA system $\mathcal{S}$ over the set of roles R and a role* `goal` $\in R$*, the* role-reachability problem *asks whether* `goal` *is reachable in $\mathcal{S}$.*

4 User Attribute-based Access Control in Azure-like platforms and State Transition System

4.1 User Attribute-based Access Control

We now briefly discuss how to enable User Attribute-Based Access Control in a RBAC deployment such as the one supported by Azure.

We define a practical Azure-like ABAC model. Let's assume a conventional formulation of users U, resources Res and Privilege sets P against resources in Res. Let a subscription denote a single Cloud domain where resources are maintained. Administrative units interface this subscription and corresponding resources by means of one Azure Active Directory. Users U are part of one or more administrative units connected to the AAD.The following elements define a User Attribute-based Access Control (UAA) model for an Azure-like platform.

- ◇ *Users and Attributes.* Each $u \in U$ is described by a unique identifier and a finite set of attributes $ATTR$. Each attribute is a function that takes users as input and returns a value from the attribute scope. Attributes can be atomic-valued or set-valued. An atomic-valued attribute will return a single value while a set-valued attribute will return a subset of values within its defined scope. An example of user is a User of id BobSmith, with attributes domain(BobSmith)=*abc*, Department(BobSmith)=*Network* and Position(BobSmith)=*Tester*.

Rule	Admin	**Pre-condition**	**Attr**	**Value**
`add`	ITC Admin	Department(u) = Network ∧ Position(u)=Developer	Department	Compute
`add`	ITC Admin	Position(u)= Tester	Department	Compute
`assign`	ITC Admin		Position	Tester

Table 1. Examples of administrative rules for ITC

Rule	Admin	**Condition**	**Attr**	**Value**
`assign`	ABC Admin	Department(u) = Network	Position	Tester
`add`	ABC Admin	Position(u)= Developer	Department	Compute
`assign`	ABC Admin		Position	Developer
`assign`	ABC Admin	Position(u)=guest	Domain	.abc
`assign`	ABC Admin	Position(u)=Intern	Domain	.abcBeta

Table 2. Examples of administrative rules for ABC

Group	**GLabel**	**Role and Resource**
`AccToRG1`	Position(u)= Tester and Department(u)= Compute	Contributor RG1
`AccToRG2`	Domain(u)= ABC	Reader RG2

Table 3. Two Dynamic Groups and corresponding conditions

- ◇ *Administrative Units:* Users within a subscription are organized into one or more administrative units. Each administrative unit is responsible for managing attributes assignments for their users. In our framework, we express each administrative unit's policy by means of GURA policies. For instance, in our example of `InsightIT`, portions of the ABC and ITC administrative policies are reported in Table 2 and 1, respectively (delete rules are excluded). We assume that each attribute can be revoked from a user regardless of any precondition. Administrative units are managed by designated administrators according to pre-defined administrative roles.
- ◇ *Dynamic Security Groups*: Users are organized in a finite set G of dynamic security groups- regardless of their original administrative unit. Moreover, *GLabel* is a function that maps a group to a logical formula expressed over user attributes that evaluates to **true** or **false**.

  Our example includes two dynamic groups `AccToRG1` and `AccToRG2`. Examples of conditions of dynamic groups in our `InsightIT` example are reported in Table 3.
- ◇ *Privilege Assignment:* Resource administrators assign privileges to dynamic groups. Privileges are mediated by roles. That is to say, an access privilege is not assigned directly but obtained as part of a role-assignment.

  In our example, Bob Smith may be assigned *Contributor* role to resource group `RG1` if he meets the requirements set for `AccToRG1`. Thus, users' access to resources is mediated by users' membership in dynamic groups.

We assume that resources are statically associated with dynamic groups, thus, in the rest of the paper we refer to an UAA policy as a tuple $\langle G, GLabel, GURAP\rangle$, where $GURAP = \langle U, AR, ATTR, UATTR, assign, add, delete\rangle$ is the $GURA$ policy obtained by combining together all administrative units policies.

4.2 User Attribute-based Systems

A User Attribute-based (UAA) system is a a $GURA$ system where users can be organised in security groups according to their attributes. The system maintains the invariant that a user belongs to a group if and only if her attributes satisfies a given condition associated with the security group.

Formally, a UAA system is a state transition system defined as $\mathcal{S} = \langle U, AR\ G, GLabel, GA, ATTR, UATTR, assign, add, delete\rangle$ where $\langle G, GLabel, GURAP\rangle$ is a UAA policy, $GURAP = \langle U, AR, ATTR, UATTR, assign, add, delete\rangle$ is the underlying $GURA$ system, and $GA \subseteq U \times G$ is the *user-group assignment* relation.

A *configuration* of $\mathcal{S}$ is any pair (UV, UG) where $UV \subseteq U \times ATTR \times SCOPE_{att}$ is a user-attribute assignment relation and $UG \subseteq U \times G$ is a user-group assignment relation. A configuration (UV, UG) is *initial* if $(UV, UG) = (UATTR, GA)$.

Given two $\mathcal{S}$ configurations (UV, UG) and (UV', UG'), there is a *transition* from (UV, UG) to (UV', UG') with rule $m \in (can\_assign \cup can\_add \cup can\_delete)$, denoted $(UV, UG) \xrightarrow{m} (UV', UG')$, if in the underlying $GURA$ system there is a transition $UV \xrightarrow{m} UV'$. Moreover, for each $(u, g) \in UG$ and $(u', g') \in UG'$, u satisfies $GLabel(g)$ and u' satisfies $GLabel(g')$.

A *run* of $\mathcal{S}$ is any finite sequence of $\mathcal{S}$ transitions $\pi = c_1 \xrightarrow{m_1} c_2 \xrightarrow{m_2} \ldots c_n \xrightarrow{m_n} c_{n+1}$ for some $n \geq 0$, where c_1 is an *initial* configuration of $\mathcal{S}$. An $\mathcal{S}$ configuration c is reachable if c is the last configuration of an $\mathcal{S}$ run.

Definition 3 (Group-reachability Problem**).** *For any group $g \in G$, g is* reachable *in $\mathcal{S}$ if there is an $\mathcal{S}$ reachable configuration (UV, UG) such that $(u, g) \in UG$, for some $u \in U$. Given an UAA system $\mathcal{S}$ and a target group $\mathtt{g} \in G$, the* group-reachability problem *asks whether* `g` *is reachable in $\mathcal{S}$.*

For instance, in our `InsightIT` example (Section 2) we can ask whether resource group `RG2` can be reachable by ABC interns or whether group `RG1` can be reachable by an ITC tester.

Restricted UAA (rUAA): A restricted UAA system is a UAA system where the underlying $GURA$ system is a $rGURA_1$ system.

5 Group, Attribute and Role Reachability

In this section, we first show that the group-reachability problem in Azure-like user attribute based access control is equivalent to the attribute-reachability

problem in *GURA*. Then, we show that scalable techniques and tools that have been recently proposed to address the role-reachabilty problem in administrative RBAC can be employed to address the group-reachability problem for an interesting class of instances via a reduction from the attribute-reachability problem in $rGURA_1$ (see Section 3.2) to the ARBAC role-reachability problem. Proofs are omitted for lack of space.

Theorem 1. *The group-reachability problem in UAA is equivalent to the attribute-reachability problem in GURA.*

Corollary 1. *The group-reachability problem in rUAA is equivalent to the attribute-reachability problem in* $rGURA_1$.

We now show a reduction from the attribute-reachability problem in $rGURA_1$ to the role-reachability problem in *ARBAC-URA*.

Definition 4 (From $rGURA_1$ to ARBAC-URA).
Let $\mathcal{S} = \langle U, AR, ATTR, UATTR, assign, add, delete\rangle$ *be a* $rGURA_1$ *system. We construct a corresponding* $ARBAC - URA$ *system* $\mathcal{S} = \langle U, R, AR, UA, can\_assign, can\_revoke\rangle$ *as follows:*

1. $U' = U$, *and* $AR' = AR$;
2. *for each* $(att, val) \in ATTR \times SCOPE_{att}$ *add a role* att_val *to* R;
3. *for each tuple* $(u, att, val) \in UATTR$ *add a tuple* (u, att_val) *to* UA;
4. *for each* $(admin, Pt, Nt, att, val) \in assign$ *add* $(admin, Pos, Neg, att\_val)$ *to* can_assign, *where* $Pos = \{att\_val \in R | (att, val) \in Pt\}$ *and* $Neg = \{att\_val \in R | (att, val) \in Nt\} \cup \{att\_val \in R | att$ *is an atomic-valued attribute and* $(att, val') \in Pt$ *for some* $val' \in SCOPE_{att}, val' \neq val\}$;
5. *for each* $(admin, Pt, Nt, att, val) \in add$ *add* $(admin, Pos, Neg, att\_val)$ *to* can_assign, *where* $Pos = \{att\_val \in R | (att, val) \in Pt\}$ *and* $Neg = \{att\_val \in R | (att, val) \in Nt\} \cup \{att\_val \in R | att$ *is an atomic-valued attribute and* $(att, val') \in Pt$ *for some* $val' \in SCOPE_{att}, val' \neq val\}$;
6. *for each* $(admin, att, val) \in delete$ *add* $(admin, att\_val)$ *to* can_revoke.

Theorem 2. *Let* $\mathcal{S} = \langle U, AR, ATTR, UATTR, assign, add, delete\rangle$ *be a* $rGURA_1$ *system. Let* $\mathcal{S}' = \langle U, R, AR, UA, can\_assign, can\_revoke\rangle$ *be the corresponding ARBAC-URA system of Definition 4. The pair* $(att^*, val^*) \in ATTR \times SCOPE_{att^*}$ *is reachable in* $\mathcal{S}$ *iff the role* $att^*\_val^*$ *is reachable in* $\mathcal{S}'$.

6 Related Work

Our work lies at the crossroad of two related research areas: access control in cloud and formal security analysis of administrative access control models.

Access Control in Cloud. Access control in cloud computing has gained great interest over the recent years [22, 17], with emphasis on attribute based mechanisms [21, 19, 18]. Related work has also focused on secure information sharing in Cloud Computing environment [15]. Sandhu et al. have recently focused on

models for sharing information and resources in various cloud systems, including Azure and Open-Stack [23, 24]. In comparison to this body of work, we tackle the important issue of group-availability, which to our knowledge has not been addressed before. The main concept underlying such models is from Group-Centric Secure Information Sharing (g-SIS) [11], which introduces group-based information and resources sharing, allows sharing among a group of organizations.

Security analysis. Li et al. [13] study role-based policies where role-membership rules may be added or removed by principals. Sasturkar et al. [20] prove reachability to be PSPACE-COMPLETE in ARBAC URA policies. Jayaraman et al proposed Mohawk, a tool able to both finding shallow errors in complex ARBAC policies and proving correcteness [8]. Ferrara et al presented VAC, an automatic and scalable tool for the reachability problem of ARBAC policies with an unbounded number of users [5, 4, 3]. Ranise et al. proposed ASASPXL, which is able to analyse large ARBAC policies [16]. Sandhu et al prove attribute-reachability to be PSPACE-COMPLETE in $rGURA_1$ policies [10].

7 Concluding Remarks

We presented an in-depth analysis of group-reachability in User-based access control systems. Starting from a concrete instance of an access control mechanism supported in the Azure platform, we demonstrated how it is possible to verify well-formedness in user-attribute access control. Well-formedness is analysed by addressing the group-reachability problem, which may arise as per-attribute criteria groups are used. We will provide a real-world architecture in the near future, and test it for scalability with respect to realistic scenarios.

Acknowledgements Portions of Dr Squicciarini's work was funded from National Science Foundation Grant 1453080. Portions of Dr Ferrara's work was supported by the EPSRC Grant no. EP/P022413/1. This research is also partly supported by the Microsoft Azure Internet of Things Research Award.

References

1. Kevin Beaver. What admins should know about Microsoft Azure security and vulnerabilities. http://searchwindowsserver.techtarget.com/tip/What-admins-should-know-about-Microsoft-Azure-security.
2. Biz Tech. Why enterprises that value security trust Microsoft Azure. http://www.biztechmagazine.com/article/2016/10/why-microsoft-azure-essential-enterprises-value-security.
3. Anna Lisa Ferrara, P. Madhusudan, Truc Lam Nguyen, and Gennaro Parlato. Vac - verifier of administrative role-based access control policies. In *CAV*, 2014.
4. Anna Lisa Ferrara, P. Madhusudan, and Gennaro Parlato. Security analysis of role-based access control through program verification. In Stephen Chong, editor, *IEEE Computer Security Foundation*, pages 113–125. IEEE, 2012.
5. Anna Lisa Ferrara, P. Madhusudan, and Gennaro Parlato. Policy analysis for self-administrated role-based access control. In *TACAS*, pages 432–447, 2013.

6. Roberto Freato. *Microsoft Azure Security.* Packt Publishing Ltd, 2015.
7. Inforworld. The dirty dozen: 12 cloud security threats, 2016. http://www.infoworld.com/article/3041078/security/the-dirty-dozen-12-cloud-security-threats.html.
8. Karthick Jayaraman, Mahesh V. Tripunitara, Vijay Ganesh, Martin C. Rinard, and Steve J. Chapin. Mohawk: Abstraction-refinement and bound-estimation for verifying access control policies. *ACM Trans. Inf. Syst. Secur.*, 15(4):18, 2013.
9. Xin Jin, Ram Krishnan, and Ravi Sandhu. A role-based administration model for attributes. In *First International Workshop on Secure and Resilient Architectures and Systems*, pages 7–12. ACM, 2012.
10. Xin Jin, Ram Krishnan, and Ravi Sandhu. Reachability analysis for role-based administration of attributes. In *ACM Workshop on Digital Identity Management*, pages 73–84. ACM, 2013.
11. Ram Krishnan, Ravi Sandhu, Jianwei Niu, and William Winsborough. A conceptual framework for group-centric secure information sharing. In *4th International Symp. on Inf., Computer, and Communications Sec.*, pages 384–387. ACM, 2009.
12. Ronald L Krutz and Russell Dean Vines. *Cloud security: A comprehensive guide to secure cloud computing.* Wiley Publishing, 2010.
13. Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. In *9th ACM SACMAT*, pages 126–135. ACM, 2004.
14. Microsoft Azure Documentation. https://docs.microsoft.com/en-us/azure/active-directory/role-based-access-control-troubleshooting.
15. Meikang Qiu, Keke Gai, Bhavani Thuraisingham, Lixin Tao, and Hui Zhao. Proactive user-centric secure data scheme using attribute-based semantic access controls for mobile clouds in financial industry. *Future Generation Computer Systems*, 2016.
16. Silvio Ranise, Anh Tuan Truong, and Alessandro Armando. Boosting model checking to analyse large ARBAC policies. In *Security and Trust Management - 8th International Workshop, STM*, pages 273–288, 2012.
17. Mariana Raykova, Hang Zhao, and Steven M Bellovin. Privacy enhanced access control for outsourced data sharing. In *International Conference on Financial Cryptography and Data Security*, pages 223–238. Springer, 2012.
18. Khaled Riad and Zhu Yan. Ear-abac: An extended ar-abac access control model for sdn-integrated cloud computing. *environments*, 132(14), 2015.
19. Khaled Riad, Zhu Yan, Hongxin Hu, and Gail-Joon Ahn. Ar-abac: A new attribute based access control model supporting attribute-rules for cloud computing. In *Collaboration and Internet Computing Conference (CIC)*, pages 28–35. IEEE, 2015.
20. Amit Sasturkar, Ping Yang, Scott D. Stoller, and C. R. Ramakrishnan. Policy analysis for administrative role based access control. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19) 2006*, pages 124–138, 2006.
21. Zhiguo Wan, Jun'e Liu, and Robert Deng. HASBE: a hierarchical attribute-based solution for flexible and scalable access control in cloud computing. *IEEE Trans. on information forensics and security*, 7(2):743–754, 2012.
22. Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *IEEE Infocom*, pages 1–9, 2010.
23. Yun Zhang, Farhan Patwa, and Ravi Sandhu. Community-based secure information and resource sharing in Azure Cloud IaaS. In *4th Int. Workshop on Security in Cloud Computing*, pages 82–89. ACM, 2016.
24. Yun Zhang, Farhan Patwa, Ravi Sandhu, and Bo Tang. Hierarchical secure information and resource sharing in openstack community cloud. In *IEEE Int. Conference on Information Reuse and Integration*, pages 419–426. IEEE, 2015.