Composition of Access Control Schemes from Practice to Theory

Indrani Ray, Mahesh Tripunitara
Electrical and Computer Engineering
{**iray**, tripunit}@uwaterloo.ca

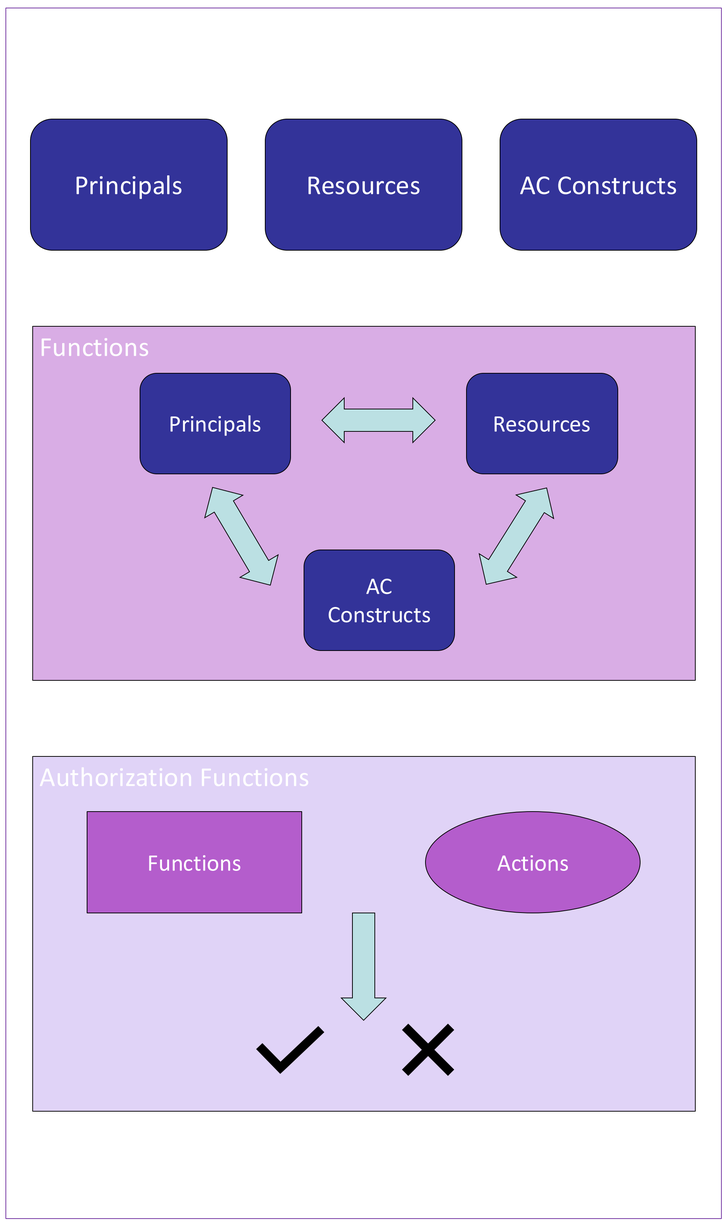
Motivation

- Demand for scalable and secure data management is increasing
- Companies are reliant on enterprise data storage systems
- These systems implement access control in multiple ways
- It is important to study how implementations co-exist to ensure data is protected

Our Approach

- Related work focuses on "policy composition"; researchers develop algebraic frameworks to express access control policies according to proposed syntaxes and introduce operators to reason about their compositions
- We propose the study of state-change systems, i.e., access control schemes instead of studying the composition of static authorization policies
- We take inspiration from state machine composition themes in related fields such as distributed systems.
- We define authorization states, transitions, and compositions based-off of a bottom-up approach studying Google Cloud Storage, Amazon Simple Storage Service, and Oracle Database Express Edition and summarize findings present in all three systems.

Authorization State



State Transition

In each access control scheme, we consider two types of state transitions:

- those that effect changes to the sets defined in the authorization state
- those that affect evaluations of the non-authorization functions on elements of their domains

Example Actions

System	Scheme	Action	State-Change Action
Google Cloud Storage	IAM	list bucket	remove role from user on bucket
	ACL	list objects in bucket	change ACL permission on bucket
	Composed	download object from bucket	create object
Amazon Simple Storage Service	Identity	list buckets	detach user policy
	Resource	list objects in bucket	remove from bucket policy
	Composed	get object	create object
Oracle Database Express Edition	Discretionary	list names of tables	grant table privilege to user
	Role	describe table	revoke role from user
	Composed	insert into table	create role

Composition Methodology

1. Define authorization state of each scheme. Sets of principals, resources, and access control constructs are evident from intrinsic design. Relations are modeled according to empirical observations. Authorization functions are defined from sets of necessary and sufficient conditions principals need to exercise actions on resources. Define state transitions accordingly.

2. Formalize set of constraints to determine when a state from one scheme can compose with a state from another. These constraints are based off-of the implementation of the underlying system in which both schemes co-exist.

3. Define composed state which satisfies above constraints using elements and functions of component states and update authorization functions accordingly. Determine transitions from composed state.

Feedback from Real World

Sample Formalization

An example transition from a composed state in Amazon Simple Storage Service is creating an object. The picture below shows the conditions according to which a user i is authorized to this action.

(
$\exists R_1 \in 2^{BUO}, \phi_1 \in 2^{PUS}$ s.t.
- $b \in R_1$
- s3:PutObject $\in \phi_1$
- $\langle R_1, \text{Allow}, \phi_1 \rangle \in ib(i)$ or $\langle \omega, \text{Allow}, \phi_1 \rangle \in ib(i)$

or $\exists V_1 \in 2^{U}, \phi_2 \in 2^{PUS}$ s.t.
- $u \in V_1$
- s3:PutObject $\in \phi_2$
- $\langle V_1, \text{Allow}, \phi_2 \rangle \in rb(b)$

)
and (
$\nexists R_2 \in 2^{BUO}, \phi_3 \in 2^{PUS}$ with
- $b \in R_2$
- s3:PutObject $\in \phi_2$
- $\langle R_2, \text{Deny}, \phi_3 \rangle \in ib(u)$ or $\langle \omega, \text{Deny}, \phi_3 \rangle \in ib(i)$

)
and (
$\nexists V_2 \in 2^{U}, \phi_4 \in 2^{PUS}$ with
- $u \in V_2$
- s3:PutObject $\in \phi_4$
- $\langle V_2, \text{Deny}, \phi_4 \rangle \in rb(b)$

)

Informally, what is expressed is an exhaustive check of resource-based and identity-based policies to ensure the user is explicitly granted and not denied the `s3:PutObject` permission on the bucket.

Takeaways

- Each system has two schemes, but the initial state in one scheme must be derived from a state in the other.
- In each system, from a composed state, a user may be authorized to exercise some state-changing actions pertaining to one scheme by means of authorization from the other.
- Let S_1 be a state in one scheme, S_2 in the other, and T_1 a transition from S_1. Then, T_1 is also a transition from $S = S_1 \circ S_2$
- Let T be a transition from $S = S_1 \circ S_2$, then T is not necessarily enabled from both component states

Next Steps

Practice

- Companies migrate from one storage platform to another or use multiple in conjunction.
- Defining theoretical frameworks for these circumstances is a next step, especially considering partnerships among platforms studied here.
- An integration of Oracle database services into the AWS environment claims to enable users to transfer files from a database instance into a bucket; our composition will answer whether the user is authorized to do so.

Theory

- Designs of access control systems take inspiration from well-established theoretical models
- Identity and access management systems (in, for example, Google Cloud Storage) are designed off of role-based access control (RBAC) and administrative role-based access control (ARBAC).
- File systems and database management systems take inspiration from access matrix models.
- Our current work-in-progress considers these foundational models, namely ARBAC and the HRU access matrix, we define both schemes, outline states and transitions, and formulate a composition of the two schemes.