Identity Management Overview - Yale University 2009-2012

Overview

Yale's identity and access management project had a wide and varied scope. The goal of the IDM data model was to be flexible enough to accommodate such a scope. Ultimately, the goal was to create an aggregate collection of identity, role, and permission information for everyone affiliated with the university.

Key Challenges

A university environment presents some unique challenges to identity management. The first is that there are multiple sources of identity information, for example (not a complete list):

- Student systems (Banner)
- Faculty/Staff system (Oracle EBS)
- Alumni system (Blackbaud)

A person can (of course) be in multiple systems at once. This required a heuristic based identity matching algorithm to identity matching identities and merge them during aggregation.

Another challenge is that there are multiple role hierarchies within the university, including (but not limited to):

- Financial hierarchy (who reports to whom)
- Faculty hierarchy (teaching faculty below department heads below provosts, etc)

Any given person could be in multiple hierarchies (I have a "manager" but I also teach in the Computer Science Department, so I'm under the CS Department head). Additionally, a given person might be found multiple times in a single hierarchy (I might teach in both the Physics and Math departments).

There were a number of additional esoteric challenges unique to Yale (vs. unique to higher ed institutions) that complicate the matter but probably aren't worth exploring here.

Requirements

There are two sides to IDM at Yale. The IDM system was responsible for pulling identity data from multiple sources of record (inbound) and then providing it to multiple IDM data consumers (outbound).

On the inbound side, it's only worth noting that there are multiple sources of identity, and that a process was created to merge those multiple sources of data into a single identity store. It's a messy process due to possible data quality problems with the sources of identity.

On the outbound side, the identity data was to be consumed in a number of ways, including:

- Providing identity data to partners (both internal and cloud services)
    - Example: nightly update of identity data to Salesforce.com
    - Example: hourly push of data to self-hosted HP Service Manager
- Providing APIs to allow partners to directly use the identity data
    - Example: custom built JBoss login module to set up a SecurityContext based on identity and role/permission data in the identity store
- Providing identity change events to drive async business processes
    - Example: trigger onboarding process to auto-provision required services for the new user

The identity data consists of both identity attributes (name, phone numbers, etc) and role/ permission information. The identity data is pretty straightforward, but the role/permission information was intended to be used in the following scenarios (representative list, but not exhaustive):

- Coarse-grained application authorization - an application might query the IDM system to check that an authenticated user had a simple role before it would let them in. For example, an application that only students are allowed to access might check for a simple Student role.
- Fine-grained application authorization - an application might need to allow a user to manipulate a subset of data based on the user's qualified role membership. For example, an Email Administrator is allowed to manage instances of the Email Service in the Service Provisioning application but cannot view or manage instances of any other type of Service. On the other hand, a Help Desk User can view all services but only perform certain actions on them.
- Fine-grained data access - a business manager has access to payroll data, but only for the employees who are found in a particular node in the financial hierarchy (along with all descendents of that node). For example, Business Manager A is responsible for the Help Desk unit and can see only their payroll information (note: the business manager is not herself a member of the Help Desk group/role). Business Manager B is responsible for the ITS unit, which is an ancestor node of Help Desk unit in the financial hierarchy, so she has access to the same information as Business Manager A and a whole lot more.

Data Model

First here are the definitions of the various components of the data model:

- **Identity** - standard identity attributes such as Name, Phone Numbers, Address, etc
- **System** - attributes to identity a “System” such as Name, Type, Description. This represents a software application that must make API calls to some other application and requires permissions to perform certain actions.
- **Subject** - points to either an Identity or a System, this “represents” one of those and is the entity that is granted membership in Roles. (possibly an unnecessary level of abstraction)
- **Principal** - a login credential such as a username or an x509 certificate. The principal has a pointer to a Subject. This is typically the login credential a user types in to authenticate to an application.
- **Group** - a simple collection of Subjects. This allows the construction of group hierarchies, because Groups can be put into Groups. This is pretty standard LDAP-like stuff.
- **Role** - a collection of Permissions. Subjects and Groups can be assigned a Role. APIs allow querying for Roles and/or Permissions (coarse vs. fine grained access). Ultimately, Permissions (used for fine-grained access to applications) are granted to users when a user is given a Role.
- **Permission** - used for fine-grained access to applications, these are associated with Roles and granted to users who are assigned those Roles.

To complicate matters, when a Subject or Group is given a Role, it can be qualified by a Qualifier. A Qualifier is either a simple string value, or else a node in a pre-defined qualifier hierarchy. So a given Subject might be given the same Role multiple times but with different Qualifiers. The Permissions given out to Subjects based on the Subject’s Role assignments inherit the Qualifier specified in that assignment. In other words, if the user is assigned the SystemAdmin role, qualified by “rhel-web1.internal.yale.edu” then that user might end up with the following Permissions:

SSH(rhel-web1.internal.yale.edu)
Reboot(rhel-web1.internal.yale.edu)
ViewLogs(rhel-web1.internal.yale.edu)

If that same user is assigned the SystemAdmin role again, this time qualified by “rhel-mysql3.internal.yale.edu” then that user might have the following Permissions:

SSH(rhel-web1.internal.yale.edu)
SSH(rhel-mysql3.internal.yale.edu)
Reboot(rhel-web1.internal.yale.edu)

Reboot(rhel-mysql3.internal.yale.edu)
ViewLogs(rhel-web1.internal.yale.edu)
ViewLogs(rhel-mysql3.internal.yale.edu)