Building a new model for Account Management

Simon Wilkinson <simon@sxw.org.uk>
School of Informatics, University of Edinburgh
Introduction

• Background & Requirements
  • Decentralisation
  • Account types
  • Access and Identity Models
  • Auditing

• Our Implementation
  • Data Model
  • Architecture
  • Technologies

• Conclusions
Watchu talkin’ ‘bout, Willis?

• What do we mean by Account/Identity* Management?

• Layer between corporate database, and your services

• Ensures that changes propagate down (*Identity Lifecycle*)

• Ensures that data in all locations matches the corporate copy

*Insert buzzword of choice
Distribution and decentralisation

• Historically, an organisation would have either
  • A central account database used by every service
  • Every service using its own database
  • (or, a mixture of the above)

• Both are untenable in a distributed, decentralised world
Account Database per Service

• No idea who your ‘users’ are

• Deletion/disabling is impossible

• Management is fragmented

• Delegated services are difficult
Centralised database for all services

- Delegated services are impossible
- Every service must interoperate with the central database
- Central database must know about every service.
- Scalability is poor, especially for systems without a strong centre
Decentralisation

- Imagine a site with hundreds of services
- Bringing up a service at the periphery shouldn’t require action at the centre
- Users have to be able to deploy their own services, which use the central account management system
- Users must be able to manage their own access control, both for these and for centrally managed services
Account spectra

- Traditionally have ‘accounts’ for institutional users
- Gulf separates those from ‘visitors’, and again from web application accounts.
- These boundaries are increasing archaic
- Accounts must be able to slide in position on the spectrum
Role Based Access Control

- Traditionally user access has been controlled via user lists or groups.

- Role based access control adds additional flexibility.

- A user has one or more roles, which describes functions they perform:
  - Staff
  - Student
  - Head of Department
  - System administrator
Entitlements

• Each role confers upon the user a set of *entitlements*

• Entitlements determine what they can do on the system
  • Log in to webmail
  • Log in to the compute cluster
  • Access the finance system
  • Edit the DNS

• For flexibility, we also allow roles to include other roles
  • For example, every *student* is a *person*
Identity Modelling

- Traditionally, all of a users entitlements are associated with a single identity.

- The user authenticates to that identity, and then gets all of the access granted to that identity.

- Doesn’t solve
  - “I want a password that can just access my webmail”
  - “I want a key that will let a process just write to this directory”
  - “I shouldn’t have admin permissions unless necessary”
  - “I should need a smartcard to login as an administrator”
Facets of Identity

• Split identity into multiple facets (or instances)

• Each instance has a subset of the base identity’s entitlements

• Some instances may have additional entitlements

• Let users create instances as required, and distribute entitlements between them
Assurance

- Different levels of access require different levels of identity verification
  - Passwords
  - One-time passwords
  - Hardware tokens

- Important to be able to keep this different levels separate on the system (*can’t all share a Unix UID, if they’re shell sessions*)
Accounts

- Ultimately, an account is a property of a particular system
- Different systems may have different account attributes (uids, gids, shells, etc)
- Not all identities will have accounts, but some will have multiple accounts
Entities

• Machines access services too, and their accounts must be managed.

• Not an Account Management System

• Not an Identity Management System

• An Entity Management System?
Distribution

- A central system doesn’t scale, either physically, or in terms of administrator effort

- Each service must be responsible for managing its local database based on the contents of the central system

- Active, regular synchronisation is vital

- Must be local to the service, rather than pushed from the centre
Auditing

• Important to regularly report on the integrity of the system

• Audit runs identify anomalies and inconsistencies between databases

• Audit reports can highlight software bugs, and operator errors

• Provided as a tool for service administrators
Delegation

• Users should be able to manage their own identity

• Users should be able to own groups and entitlements

• Users should be able to bring up services which use the central system

• A service administrator is just another user
Introducing Prometheus

• The School of Informatics’s new account management system

• Currently named *Prometheus*

• Designed to address all of the previously discussed issues

• Very much a work in progress!
The Entity Data Model

Entity
  Person
  Machine

Identity A
Account A

Identity B

Identity C
Account B
What goes where ...

- Entities contain real-world information, plus overall role data
- Identities contain instance specific data, and authentication details
- Accounts contain OS specific data (uid, gid and the like)
Role and Entitlement data model

• A role contains
  • other roles
  • entitlements
  • negated entitlements

• An entity has both roles and explicit entitlements

• An identity may have any roles and entitlements owned by its parent entity (and must have any negated entries)

• An identity may have additional roles and entitlements
Role and Entitlement Ownership

• In order to achieve delegation we have to have an ownership model for roles and entitlements

• An owner may grant that object to another role or user

• An owner may resign (remove) any object that they’ve been granted, but cannot then restore it.

• Owners may restrict who can use an entitlement

• Owners may delegate these powers to other users
Architecture - abstract

Source Databases

Conduits

Command Line Interface
Web Interface
Scripts

Database

Conduits

Service Databases
Data flow

- Source database may be flat file, RDBMs, LDAP, whatever you like

- Import *conduit* translates from source database to abstract attribute store

- Output *conduit* translates from attribute store to service-specific database

- This can be circular!
Conduit operation

• Conduits can operate over entire data set, or be triggered by entitlements (ie kdcentry gives an instance a principal)

• Conduits can be pushed data or they may pull it. (support for notified pulls is planned)

• Conduits can work with deltas, or the whole database

• Audit runs are supported, and a reporting interface is provided

• Conduits may add arbitrary schema to the attribute store
Architecture - detail

HR
Friends
LCFG

Account Management Tools
Role Editor
Identity Editor

API

Database

API

KDC
LDAP
AFS
Jabber

dbsync
iFriendSync
lcfgsync
afssync
ldapsync
kdcsync
jabbersync
Implementation

• Central database, and communication protocol LDAP based. Server implemented on top of OpenLDAP

• Advantages
  • Common, secure, well specified protocol
  • Hierarchical directory layout suits our data model
  • Straightforward, well documented, common place client API

• Disadvantages
  • Hard to express role and entitlement ownership model
  • Server data model requires external scripts, or plugins
Implementation

- Plan to provide an OpenMetaDir message bus interface to our LDAP repository

- OpenMetaDir is a framework designed for producing account management systems

- Powerful message passing, routing, and schema/ontology definitions

- Likely to be significantly faster than the LDAP system, but require more coding effort in the conduit
Implementation - Conduits

- Conduit simplicity is the primary goal

- Conduits which just want to pull information just need to
  - Register their identity, and triggering entitlement with the server
  - Perform ldapsearches with that identity

```bash
ldapsearch -h prometheus.inf.ed.ac.uk \
  -b "o=Prometheus,dc=inf,dc=ed,dc=ac,dc=uk" \
  (objectClass=prometheusIdentity)
```

- Conduits with more complex requirements can use syncrepl (for updates) and OpenMetaDir (if their needs are specialised)
Implementation - Conduit Simplicity

- Simple conduits require the server have the ability to collapse the entity data model ...
The Web Interface
Web Interface Implementation

• Written using Catalyst - a perl MVC framework

• Uses prometheus API to communicate with LDAP server

• Uses user’s authentication tokens to secure server connection

• All authorization and access control checks performed in the server
Conclusions

• System design that meets all of our requirements

• Implementation continuing as an evolution of our existing system

• Code drops will be available shortly

• Very interested in talking to other sites that might be interested in any of this!
Questions?

This talk: http://www.dice.inf.ed.ac.uk/publications/

Prometheus: http://www.dice.inf.ed.ac.uk/prometheus/

Me: simon@sxw.org.uk