Authentication in Rails

Aaron Mulder
CTO Chariot Solutions
Philly on Rails, October 2007

CHARIOT SOLUTIONS

Agenda

- The problem
- Plugins in Rails, and the (many) solutions
- acts_as_authenticated
- Generated Code
- Custom Code
- Authentication tests
- Summary and Next Steps
- Q&A

The Problem

Authentication

- How can we write the next great social networking phenomenon without knowing who our users are?
- Want to be able to make site users log in to access certain content (or a customized home page, etc.)
- May want things like:
  - A “remember me” cookie
  - An e-mail they have to click to activate their account
  - An optional login box on the front page

Authorization

- In the easy case, only users in a certain role (e.g. admins, moderators) should be able to access a certain screen
- In the harder case, all logged-in users should be able to access some screen (e.g. edit profile), but only for certain data (your own)
- Still harder: only for people in the same department, or only content generated by you or your buddies, etc.

Why is this a Problem?

- Well... where's the login feature in Rails?
  - I guess it's one of those cases where the core team is waiting for the best solution to emerge?
- You can do it yourself
  - I guess maybe not difficult, but probably more work than it might seem and hard to assure yourself that it's right...
- Or, you can choose from dozens of Rails Plugins that support this to some degree
  - But which is “the best”?

Digression: Plugins

Rails Plugins

- A plugin is an easy-to-install package that extends the Rails environment in some way
  - Libraries
  - Code generators
  - Rake tasks
  - Extensions to existing features (ActiveRecord, Migrations, etc.)
- It lives under vendor/plugins and may include various scripts (install, start, etc.) and its own self-tests

script/plugin

- The tool used to manage plugins
- Can list, install, update, etc.
- There are “sources” (or repositories) for plugins which you can add and remove
  - To list plugins from a source, or install plugins from a source just by using the name of the plugin
- Or each plugin can be installed individually with its own specific URL

script/plugin Usage

```
$ script/plugin
Usage: plugin [OPTIONS] command
COMMANDS
  discover    Discover plugin repositories.
  list        List available plugins.
  install     Install plugin(s) from known repositories or URLs.
  update      Update installed plugins.
  remove      Uninstall plugins.
  source      Add a plugin source repository.
  unsource    Remove a plugin repository.
  sources     List currently configured plugin repositories.
EXAMPLES
  List all available plugins:
    plugin list
...
```

Plugin Sources

- Several repositories have lots of plugins available
  - http://dev.rubyonrails.com/svn/rails/plugins/
  - http://svn.techno-weenie.net/projects/plugins
- If you add a repository you can easily list and install plugins from it just by name
- But can always just install a single plugin by providing a full path to it
  - I don't find “browsing” the plugin repository all that useful – Google has much more info

acts_as_authenticated

Choosing an Authentication Plugin

- There are way too many authentication options:
  - http://wiki.rubyonrails.com/rails/pages/Authentication
- acts_as_authenticated seems functional and useful, so we'll talk about that one today
  - Includes signup, login, remember_me, authorization, quick syntax for designating restricted pages, etc.
- It's a plugin, so we'll install it with the syntax we just saw
- It includes a script/generate generator, which will put all the code we need into the project

Install acts_as_authenticated

```
$ script/plugin install http://svn.techno-
weenie.net/projects/plugins/acts_as_authenticated
+ ./acts_as_authenticated/CHANGELOG
+ ./acts_as_authenticated/README
+ ./acts_as_authenticated/generators/authenticated/USAGE
...
acts_as_authenticated generator
====
This is a basic authentication generator for rails, very much
in the spirit of xal's original Login Generator.

To use:

  ./script/generate authenticated user account
...
```

What Was Installed?

- There was a list at the top of the script/plugin output
- But also, see `vendor/plugins/acts_as_authenticated`
- Mainly everything's hidden in the generator, but we'll shortly see how that goes
  - There's an install script, but it just prints the README during the installation process
- Note that nothing changes in your project until you run the generator(s)

Integrating acts_as_authenticated Part 1

Overall Strategy

- Generate a migration and model for User (a table and a class)
- Generate some common library code to handle authentication tasks (remembering the current user, hooks for redirecting to the login page, etc.)
- Generate a controller for the signup and login processes
- Let you (the developer) hook those in to the application flow as appropriate

Run the generator

- `script/generate authenticated foo bar`
- First argument (foo) is name for user model
- Second argument (bar) is name for controller that will handle signup and login

```
$ script/generate authenticated user account
      create  db/migrate/001_create_users.rb
      create  app/models/user.rb
      create  app/controllers/account_controller.rb
      create  lib/authenticated_system.rb
      create  test/unit/user_test.rb
      create  test/functional/account_controller_test.rb
      ...
```

Migration

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table "users", :force => true do |t|
      t.column :login,                     :string
      t.column :email,                     :string
      t.column :crypted_password,          :string, :limit => 40
      t.column :salt,                      :string, :limit => 40
      t.column :created_at,                :datetime
      t.column :updated_at,                :datetime
      t.column :remember_token,            :string
      t.column :remember_token_expires_at, :datetime
    end
  end

  def self.down
    drop_table "users"
  end
```

User Model

- Too long to show, so...
- Many validations (login required, confirm password, etc.)
- Class methods for login, password hashing, etc.
- Instance methods for dealing with the remember me token
- Before save filter to hash the password before it's stored

Account Controller

- Too long to show, so...
- Some boilerplate code we should move to ApplicationController
- A few methods/screens
  - login
  - logout
  - signup
  - index
- They mostly handle get/post differently

AuthenticatedSystem

- A module we can mix in to ApplicationController
  - Makes all the authentication stuff available to all controllers, but doesn't automatically use it
- Helper methods
  - logged_in?, current_user, access_denied
- Filter methods
  - login_required, login_from_cookie
- Authorization
  - authorized? (need to override this one)

Integrating acts_as_authenticated Part 2

Custom Code

- Move first couple lines from AccountController to ApplicationController
  - This mixes in AuthenticatedSystem to each controller
  - Also adds remember_me login support
- For each protected controller:
  - add `before_filter :login_required`
- For authorization:
  - implement `authorized?`
  - e.g `return true if current_user.role == 'Admin'` or add logic based on `action_name`

A Protected Controller

```
class ProfileController < ApplicationController
  before_filter :login_required

  def authorized?
    puts "Checking #{action_name} for #{current_user.login}"
    return true if current_user.role == 'Admin'
    case action_name
      when "list"
        return true
      when "show", "edit", "update"
        id = params[:id]
        return true unless id.nil? || id.to_i != current_user.id
    end
    false
  end

  ...
```

User Signup

- Handled by AccountController.signup, and the associated view
- Populates and saves a user model, then redirects
  - Or re-renders the signup view in case of error
- You can do whatever you want with this (make enrollment an administrative activity instead of allowing end users to sign up, etc.)

The Login Process

- The before_filter (login_required) saves your target URL and invokes the login process
- Handled by AccountController.login
  - If a POST request, attempts to authenticate using the User.authenticate class method
  - Sets the result (which may be nil) to current_user
  - If the login succeeded (a valid user was set), sets the remember_me cookie if requested
  - Redirects to wherever you were headed or some default landing page

But You May Want To Change...

- Field names in the User (login -> username)
  - Just hunt down all the code that involves it, mainly in the model helper methods and controller login
- The property name current_user (could be current_author or current_customer or whatever)
  - Look in the account controller methods and AuthenticatedSystem methods
- The default landing pages for post-login, post-signup, or access denied
- Maybe add a role field or table for users

Change Post-login Landing Page

- Change the controller/action in this code in the AccountController:

```
def login
  ...
  # if login was successful
  redirect_back_or_default(:controller => '/account',
                           :action => 'index')
  flash[:notice] = "Logged in successfully"
end
```

- Similar change to signup code (same place)

Change Access Denied Handling

- Look at lib/authenticated_system.rb:

```
def access_denied
  respond_to do |accepts|
    accepts.html do
      if logged_in?
        redirect_to :controller => '/account',
                    :action     => 'access_denied'
      else
        store_location
        redirect_to :controller => '/account',
                    :action     => 'login'
      end
    end
    ...
```

And One More Thing

- I didn't like all the account junk in my user model
  - cookie, expiration, password salt, etc.
- I tried moving it out to a child table
  - It was, to quote Steve Miller, a great big hassle
  - I eventually gave up
  - At least with ActiveRecord, you don't have to look at all the properties right there in your model
  - (Except for the page worth of validations
  - I should just move them to the bottom of the file...)

acts_as_authenticated Tests

Unit Tests

- Quite a few written for you
  - Test required login & email, test remember me, test password rehashing, etc.
- Will need to update them if you change property names, add new required fields to the user model, etc.
  - See e.g. UserTest.create_user and the method that call it

Some Unit Test Code

```
def test_should_unset_remember_token
  users(:quentin).remember_me
  assert_not_nil users(:quentin).remember_token
  users(:quentin).forget_me
  assert_nil users(:quentin).remember_token
end

protected
  def create_user(options = {})
    User.create({ :login => 'quire',
                  :email => 'quire@example.com',
                  :password => 'quire',
                  :password_confirmation => 'quire'
                }.merge(options))
  end
```

Functional Tests

- Quite a few written for you
  - Test login and logout cookie behavior, valid and expired cookies, required fields on signup
- Will need to update them if you change property names, add new required fields to the user model, etc.
  - See e.g. AccountControllerTest.create_user and the methods that call it

Some Functional Test Code

```
def test_should_fail_cookie_login
  users(:quentin).remember_me
  @request.cookies["auth_token"] = auth_token('invalid_token')
  get :index
  assert !@controller.send(:logged_in?)
end

protected
  def create_user(options = {})
    post :signup, :user => { :login => 'quire',
                             :email => 'quire@example.com',
                             :password => 'quire',
                             :password_confirmation => 'quire'
                           }.merge(options)
  end
```

Summary and Next Steps

Summary of acts_as_authenticated

- Easy to install plugin
- Code generation puts all the code in your project
- Useful features for login, remember_me, password hashing, authorization, etc.
- May want to customize the generated code a bit, with varying degrees of pain
  - In particular, some kind of role or at least admin flag
- Includes unit and functional tests, which will need to be updated if you change the generated code

Next Steps

- Maybe fancier authorization system
  - Full RBAC with users, groups, roles, etc.?
  - With a DSL for marking up the controller?
  - Could it handle row-level security?
- Maybe a plugin to force HTTPS for the login screen and/or others with sensitive data
- Support reading user ID from HTTP headers for various security proxy servers?
- Support single sign-on tokens?

Q&A

Aaron Mulder
ammulder@chariotsolutions.com
http://chariotsolutions.com/presentations.html

CHARIOT SOLUTIONS