Episode 237

Dynamic attr_accessible

Over three years ago episode 26 [watch[1], read[2]] covered the subject of mass assignment and how it can cause security vulnerabilities. Rails 3 is now out and has a lot of security features enabled by default but this is one area that isn't. Rails applications' models still need to have their attributes protected against mass assignment in order to avoid malicious users updating them by sending POST requests to the server. If you aren't familiar with this problem then take a look at episode 26, but in essence whenever you create or update a model in your controllers by using mass assignment you need to use `attr_accessible` in the models to protect the attributes you don'r want to be updated otherwise users can set any attribute and this can lead to huge security problems.

It seems easy enough to add a call to `attr_accessible` in every model but there are two potential problems when doing so. The first problem can occur when you're testing your app. Sometimes you want to mass-assign attributes during testing and having models protected with `attr_accessible` can make this difficult. One solution to this is to use factories, and this was covered in episode 158 [watch[3], read[4]].

The second problem is that `attr_accessible` is not dynamic. The attributes that are specified by it for a given model are hard-coded and if we want to change the attributes that are accessible based on, say, the user's permissions that can be difficult to do. This was the case in Rails 2 but Rails 3 gives us a new way to make dynamic attributes and we'll show you that in this episode.

---

[1] http://railscasts.com/episodes/26-hackers-love-mass-assignment

[2] http://asciicasts.com/episodes/26-hackers-love-mass-assignment

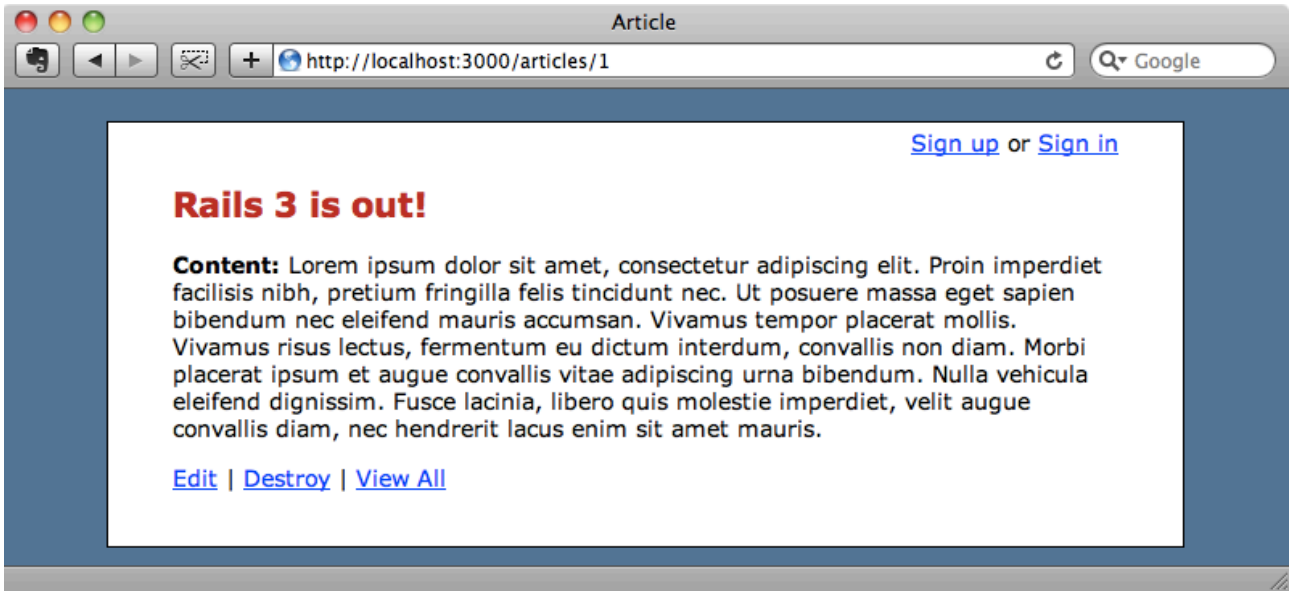
[3] http://railscasts.com/episodes/158-factories-not-fixtures

[4] http://asciicasts.com/episodes/158-factories-not-fixtures

Our Wiki Site

To demonstrate dynamic attributes we'll use a wiki site. This site has has a number of articles and an article can be edited by anyone. On the edit form, along with the name and content fields is a checkbox field that allows a user to mark as an article as 'important'.

When an article is marked as important its title will appear in red.



We're going to modify the application so that only administrators can change the importance of articles; non-admin users should not be able to modify the `important` field. It would be easy enough to modify the form to remove the checkbox so that it is only shown

to administrators, but this wouldn't solve our problem as it would still be possible for users to bypass this form and send a POST request that would modify the `important` field for an article.

The solution to this problem lies in the controller and model layers, specifically the `create` and `update` actions in the `ArticlesController` as it's here that the mass assignment happens. One approach we could take to protect the `important` attribute would be to remove it from the parameter hash unless the current user is an admin.

/app/controllers/articles_controller.rb

```
def update
  params[:article].delete(:important) unless admin?
  @article = Article.find(params[:id])
  if @article.update_attributes(params[:article])
    flash[:notice] = "Successfully updated article."
    redirect_to @article
  else
    render :action => 'edit'
  end
end
```

The trouble with this is that we have to remember to do it for every attribute we want to protect. Also there is no correlation with the `attr_accessible` call in the model. It would be much better if we could instead make `attr_accessible` dynamic.

We'll take a look at the Rails API documentation to see if we can find anything about `attr_accessible`[5] that can help. An interesting thing to note in the documentation is that `attr_accessible` is now included in the `ActiveModel::MassAssignmentSecurity` module, not directly in ActiveRecord. This means that it can be included and used within any class and this makes it much more flexible. At the top of the documentation page is an example of `ActiveModel::MassAssignmentSecurity` used directly within a controller instead of through a model, which is a really good idea. What's really interesting though is a piece of code that shows us how to make `attr_accessible` dynamic by overriding the `mass_assignment_authorizer` method.

```
def mass_assignment_authorizer
  admin ? admin_accessible_attributes : super
end
```

---

[5] http://api.rubyonrails.org/classes/ActiveModel/MassAssignmentSecurity/ClassMethods.html#method-i-attr_accessible

The code above changes the behaviour of the application based on whether the user is an admin or not which is exactly what we want to do. By overriding this method in our models we can change the fields that can be modified by mass assignment depending on any condition.

Our `Article` model currently looks like this:

/app/models/article.rb

```
class Article < ActiveRecord::Base
  attr_accessible :name, :content, :important
end
```

This is a simple enough class with just an `attr_accessible` call with three attributes. The `:important` attribute is the one we want to make dynamic and we can do that by overriding `mass_assignment_authorizer` in the class.

/app/models/article.rb

```
class Article < ActiveRecord::Base
  attr_accessible :name, :content

  private
  def mass_assignment_authorizer
    super + [:important]
  end
end
```

By calling `super` in `mass_assignment_authorizer` we'll get the default behaviour which returns a whitelist sanitizer. You don't need to be familiar with how this works to use it, you just need to know that you can add more attributes to it as we have done above. Once we've added this extra parameter here we can remove it from the list of parameters in `attr_accessible`.

The changes we've mades so far won't alter the behaviour of our application at all but we can make the accessibility of `:important` dynamic now as it's defined in an instance variable rather than at the class level. We'll do this by adding a variable to the class that will contain a list of the attributes we want to be accessible.

/app/models/article.rb

```
class Article < ActiveRecord::Base
  attr_accessible :name, :content
  attr_accessor :accessible

  private
  def mass_assignment_authorizer
    super + (accessible || [])
  end
end
```

Any parameters passed to `accessible` will now be added to the list of accessible attributes and we can use this in our controllers. We'll modify the `update` action so that it adds the `:important` parameter only is the current user is an admin.

/app/controllers/articles_controller.rb

```
def update
  @article = Article.find(params[:id])
  @article.accessible = [:important] if admin?
  if @article.update_attributes(params[:article])
    flash[:notice] = "Successfully updated article."
    redirect_to @article
  else
    render :action => 'edit'
  end
end
```

We can start up our application now to see if these changes work. If we log in to the application with an account that isn't an administrator and edit an article, marking it as important, when we're redirected back to the article's page the headline is black indicating that the `important` field hasn't been changed.

If we make our account an administrator and edit the article again, this time the `important` field is updated and the headline changes colour.

Ideally admins should be able to edit any of the fields so it would be useful if `accessible` supported an `:all` option that would allow us to easily make all of a model's attributes editable. We can do this by modifying `mass_assignment_authorizer`.

/app/models/article.rb

```
def mass_assignment_authorizer
  if accessible == :all
    self.class.protected_attributes
  else
    super + (accessible || [])
  end
end
```

The method now checks to see if `accessible` is equal to `:all`. If it is then we need to return something that will make all of the attributes modifiable. It would be good if we could just return an empty array, but unfortunately the object that is returned by `mass_assignment_authorizer` is a sanitizer object so this won't work. The way we've have worked around this is a little bit of a hack but it works well enough: we return `self.class.protected_attributes`. This is used by the `MassAssignmentSecurity` module to provide a blacklist of attributes that can't be modified. As we're not using `attr_protected` in this class it will allow all attributes which is just what we want here. We can now modify the `ArticlesController` to make all of the `Article`'s attributes accessible by passing `:all`.

/app/controllers/articles_controller.rb

```
def update
  @article = Article.find(params[:id])
  @article.accessible = :all if admin?
  if @article.update_attributes(params[:article])
    flash[:notice] = "Successfully updated article."
    redirect_to @article
  else
    render :action => 'edit'
  end
end
```

If we test this out in the application we'll see that admins can still edit the `important` attribute.

In the controller we also need to apply the `accessible` option to the `create` action. If we just apply it like this then it will not work.

/app/controllers/articles_controller.rb

```
@article = Article.new(params[:article])
@article.accessible = :all if admin?
```

The reason that this doesn't work is that the mass assignment happens in the new call so by the time we've set `accessible` it's too late. We need to separate creating a new `Article` from assigning its attributes and slip the call to `accessible` in between the two.

/app/controllers/articles_controller.rb

```
def create
  @article = Article.new
  @article.accessible = :all if admin?
  @article.attributes = params[:article]
  if @article.save
    flash[:notice] = "Successfully created article."
    redirect_to @article
  else
    render :action => 'new'
  end
end
```

You might want to make this behaviour more abstract and remove the duplication in the code in the two actions but this is dependent on how your permission system works so we'll leave this up to you. One change we will make though is to extract the `mass_assignment_authorizer` method out from the `Article` model so that it can be used in all of the application's models.

We'll move the method into an initializer. In the `/config/initializers` directory we'll create a new file called `accessible_attributes.rb`.

/config/initializers/accessible_attributes.rb

```
class ActiveRecord::Base
  attr_accessible
  attr_accessor :accessible

  private
  def mass_assignment_authorizer
    if accessible == :all
      self.class.protected_attributes
    else
      super + (accessible || [])
    end
  end
end
```

This initializer modifies `ActiveRecord::Base` so the behaviour will be applied to all models. Note that we still call `attr_accessible` but with no arguments. This means that the default behaviour will be that no attributes can be set through mass assignment and that you'll need to add another `attr_accessible` call to each model to make attributes settable. We can now clean up the `Article` model to this:

/app/models/article.rb

```
class Article < ActiveRecord::Base
  attr_accessible :name, :content
end
```

That's it for this episode. We've now made `attr_accessible` completely dynamic and can change its attributes based on a user's permissions. What's good about this approach is that everything is locked down by default and access is only granted where we explicitly specify it in the code. This makes mass assignment security issues a much smaller problem as by default all attributes are secure.