ECE 495/595 – Web Architectures/Cloud Computing

Module 5, Lecture 2: Web Application Security – Authentication

Professor G.L. Heileman

University of New Mexico

Web Application Security

- In the previous lecture we considered the basic notions behind user authentication in web applications.
- In this lecture we'll consider how to build your own user authentication into a Rails application. This will involve adding user authentication to the blog application, and will include a consideration of the various ways you can manage session information within a rails application.
- Next we'll discuss some of the off-the-shelf authentication packages that can be incorporated into a Rails application.
- Which approach you take depends upon the requirements of your application, how much customization you need to perform, etc. — whatever the case, building authentication from scratch will help you to better understand authentication packages.

Authentication from Scratch – Adding Users

- The first thing we need to do is get rid of the HTTP-based authentication that is currently in application. I.e., delete the following line in `posts\_controller.rb`:

```
# app/controllers/posts_controller.rb
http_basic_authenticate_with :name => 'admin',
   :password =>'secret', :except =>[:index, :show]
```

  Next, we need to add support for users in the blog application. We won't generate a scaffold, because we don't need all of the functionality. Users will login using their email address (to differentiate users) and a password, so run the following:

```
$ rails g model user email:string
      password_hash:string password_salt:string
```

  In order to create new users, we'll include the new action in the the users_controller:

```
$ rails g controller users new
$ rake db:migrate
```

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Sign Up

The form for signing up, `app/views/users/new.html.erb`:

```
<h1>Sign Up</h1>
<%= form_for @user do |f| %>
  <% if @user.errors.any?  %>
    <div class="error_messages">
      <h2>Form is invalid</h2>
      <ul>
        <% for message in @user.errors.full_messages %>
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>
  <p> <%= f.label :email %><br />
      <%= f.text_field :email %> </p>
  <p> <%= f.label :password %><br />
      <%= f.password_field :password %> </p>
  <p> <%= f.label :password_confirmation %><br />
      <%= f.password_field :password_confirmation %> </p>
  <p class="button"><%= f.submit "Sign Up" %></p>
<% end %>
```

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Sign Up

- Notice the if statement that will be executed if there are any validation errors associated with the model.
- Right now, if you start up the server, and navigate to `http://localhost:3000/users/new`, you'll get an error because we have not handed the form a user object to work with. In addition the `create` action that will be called when the form is submitted has not been defined. We take care of both of these actions on the next slide.
- One more thing we should fix. When we generated the users controller, it added the following to `config/routes.rb`:

  ```
  get "users/new"
  ```

  That's why the URL above works. But this doesn't look very professional. To fix this, replace the line above with;

  ```
  get "sign_up" => "users#new", :as => "sign_up"
  root :to => "posts#index"
  resources :users
  ```

  This gives us the URL `http://localhost:3000/sign_up`

Authentication from Scratch – Sign Up

- Edit users_controller.rb, defining the `new` and `create` actions:

```
class UsersController < ApplicationController
def new
  @user = User.new
end

def create
  @user = User.new(params[:user])
  if @user.save
    redirect_to root_url, :notice => "Successfully signed up!"
  else
    render "new"
  end
end
end
```

- Now, when you navigate to

  `http://localhost:3000/sign_up`

  you should see the form. Notice how the password and password_confirmation fields don't display the text typed into them.

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Sign Up

- What happens when you try to sign up a new user? You get an error:

  `Can't mass-assign protected attributes: password, password_confirmation`

  Why? Well, think back to how we generated the model, there are no password or password confirmation attributes in our user model.

- We want the model itself to have a password attribute, but we don't want the password to be saved in the database (remember, we're going to store a hash of the password instead). To accomplish this add the following to `app/models/user.rb`:

```
class User < ActiveRecord::Base
  attr_accessible :email, :password_hash, :password_salt,
      :password, :password_confirmation
  attr_accessor :password
  validates :email, :presence => true,
      :uniqueness => {:case_sensitive => false}
  validates :password, :confirmation => true
end
```

- Note: I took the time to add some validations too!

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Sign Up

- Now, when you navigate to
  `http://localhost:3000/sign_up`
  you should be able to sign up new users. You should also check that the validations are working.
- Sign some up, and check the database (using SQLite Manager), the email is saved to the database, but the `password_hash` and `password_salt` are empty.
- In order to make the root route work, you need to remove the file `./public/index.html`.
- We'll add the gem `bcrypt` and use it's one-way hash function. In your Gemfile add the line:
  ```
  gem 'bcrypt-ruby', :require => 'bcrypt'
  ```
  and run
  ```
  $ bundle install
  ```
- Now add a before filter to the `User` model that takes care of encrypting the password (next slide).

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Sign Up

```
class User < ActiveRecord::Base
 # accessor declarations and validations

  before_save :encrypt_password

  def encrypt_password
    if password.present?
      self.password_salt = BCrypt::Engine.generate_salt
      self.password_hash = BCrypt::Engine.hash_secret(password,
          password_salt)
    end
  end

end
```

Authentication from Scratch – Log In

- When you navigate to `http://localhost:3000/sign_up`, you should be able to sign up new users, and you should see the encrypted password and password salt stored in the database as well.
- We're halfway there — users are now able to sign up, but they can't log in yet. To take care of this, we'll need to write the code that manages user sessions.
- Let's start by creating a controller for sessions:

  ```
  $ rails g controller sessions new
  ```

  Once again, we need to supply a form to work with the `new` action in the sessions controller. This is the form that will be used for signing in.
- Because there is no session model, we will use the `form_tag` method, rather than the `form_for` method. We simply want the form to POST to the `sessions_path`, which is the session controller's `create` action.

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Log In

- In /app/views/sessions/new.html.erb, write:

```
<h1>Log in</h1>
<%= form_tag sessions_path do %>
  <p> <%= label_tag :email %><br />
      <%= text_field_tag :email, params[:email] %> </p>
  <p> <%= label_tag :password %><br />
      <%= password_field_tag :password %> </p>
  <p class="button"><%= submit_tag "Log In" %></p>
<% end %>
```

- We should also modify the routes. In `config/routes.rb` get rid of:

```
get "sessions/new"
```

and add:

```
get "log_in" => "sessions#new", :as => "log_in"
get "log_out" => "sessions#destroy", :as => "log_out"
resources :sessions
```

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Authenticating

We need to authenticate the user when they try to log in. To do this, add the following method in `/app/models/user.rb`:

```
def self.authenticate(email, password)
   user = find_by_email(email)
   if user && user.passwordhash == BCrypt::Engine.hash_secret(
       password, user.password_salt)
      user
   else
      false
   end
end
```

- This method will be invoked through a User object whenever a new session needs to be created.
- The user-supplied email is used to find a user object. Then the user-supplied password, along with the stored password salt for the user, are run through the same one-way hash function used to create the stored password hash. If the computed password hash matches the stored password hash, the user is authenticated.

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Log In

The create method in `app/controllers/sessions_controller.rb`:

```
def create
    @user = User.new(params[:user])
    if @user.save
      redirect_to root_url, :notice => "Successfully signed up!"
    else
      render "new"
    end
end
```

- If a user is returned from the authentication method (authentication successful), then the user's id is stored in the Rails session hash.
- If a user object is not returned (authentication fails), then a flash message is created, and the application redirects to the new method (which has been mapped to log_in) in the sessions controller.

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Log Out

- We also need to handle logging out. Add a `destroy` action to `app/controllers/sessions_controller.rb`:

```
def destroy
  session[:user_id] = nil
  redirect_to root_url, :notice => "Logged out!"
end
```

- We previously set up the `log_out` route to use this action.
- After destroying the session, the application routes to the `root_url`.

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Log In

- To display the flash messages, add the following to `app/views/layouts/application.html.erb`, in the body, just before the `<%= yield %>` statement.

```
<% flash.each do |name, msg| %>
  <%= content_tag :div, msg, :id => "#{name}" %>
<% end %>
```

- If you'd like to use the ActiveRecord session store instead, you simply need to do the following

```
$ rake db:sessions:create
$ rake db:migrate
```

This creates a table for sessions in the database.
Then, you need to tell Rails to use it via a setting in `config/initializers/session_store.rg`:

```
$ MyApplication::Application.config.session_store
                                    :active_record_store
```

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Adding Links

- Finally, let's add some links to the application that let a user sign-up and log in/out from any page in our blog application.
- Place the following before the `<%= yield %>` statement. in `app/views/layouts/application.html.erb`:

```
<div id="user_nav">
  <% if current_user %>
      Logged in as <%= current_user.email %>
      <%= link_to "Log out", log_out_path %>
  <% else %>
      <%= link_to "Sign up", sign_up_path %> or
      <%= link_to "Log in", log_in_path %>
  <% end %>
</div>
```

- Because this div was placed in the `application.html.erb` file, this HTML will be executed prior to executing any other view code.
- If there's a `current_user`, one set of links will be displayed, and if there's not, a different set of links is displayed.

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Adding Links

- However, the `current_user` is not defined yet. To do that, modify `app/controllers/application_controller.rb` to look like:

```
class ApplicationController < ActionController::Base
  protect_from_forgery

  private

  def current_user
    @current_user ||= User.find(session[:user_id]) if
        session[:user_id]
  end

  helper_method :current_user
end
```

- `current_user` is now a helper method that will be made available to every controller.

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Summary

- Right now, we're sending user credentials in the clear. To fix this, add the `force_sll` class method to `user_controller.rb` and `session_controller.rb`.
- We should probably also add some additional validations. E.g., you might consider
  - Making sure the email provided looks like an email address.
  - Ensuring that passwords are strong – right now, you can supply a password at sign-up that is a single character.
- Lastly, in real applications, we need to handle the situation of users forgetting their username or password, logging user activity, etc.
- Related to this notion, why are we saying "Invalid email or password" when user enters either an invalid email (on not in our database) or password?
  Why don't we tell them which one is invalid?

THE UNIVERSITY of NEW MEXICO

Authentication from Scratch – Summary

- That's it, we now have basic user authentication integrated into our blog application.
- However, we haven't used it for anything yet. Specifically, we'd like to use this authentication to restrict users from accessing certain resources — this is referred to as access control.
- In fact, we'd like to assign roles to users, e,g., administrator, moderator, user, and then restrict access according to the users role — this is referred to as role-based access control. We'll cover that in the next lecture.
- In addition, since we now have a good idea as to how authentication works, we'll add access control using on top of an existing Rails authentication framework — this is the way you will normally bring authentication/access control into your applications.

THE UNIVERSITY of NEW MEXICO