

Rails Security Pitfalls

Jerome Basa
@jldbasa
March 2014

whoami

hire me! ^^;



many to
mention

2014

2.6 years

3 years

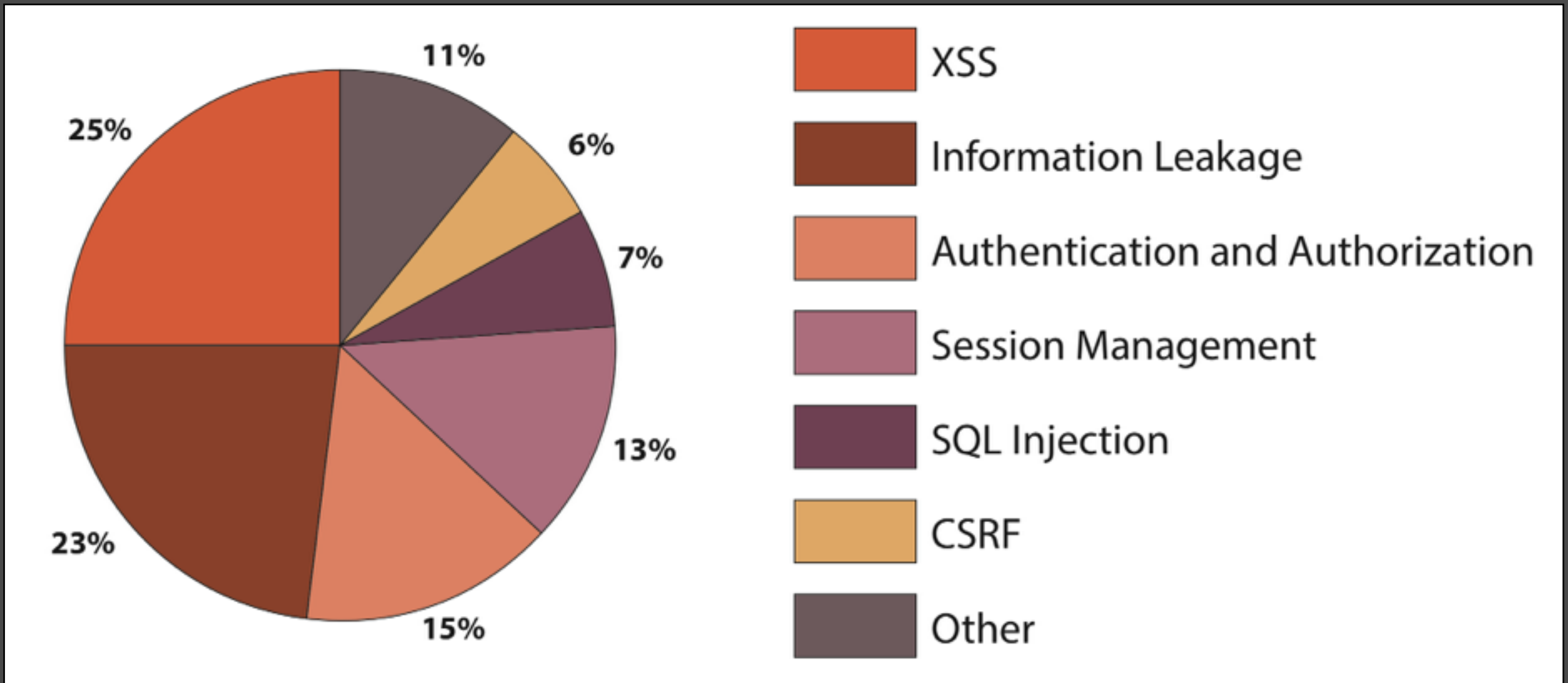
7 years



php

Web Application Security

Web Application Security



source: Cenxic Vulnerability Report 2014

Web Application Security

- “96% of tested applications in 2013 have vulnerabilities” - CENZIC
- developer usually prioritise feature completion rather than security certification
- not all companies hire dedicated security experts

is Rails secure?

Is Rails secure?

relatively secure by default

Attack Type	Rails Countermeasure
SQL Injection	SQL Escape
XSS	HTML Escape
CSRF	Authenticity Token

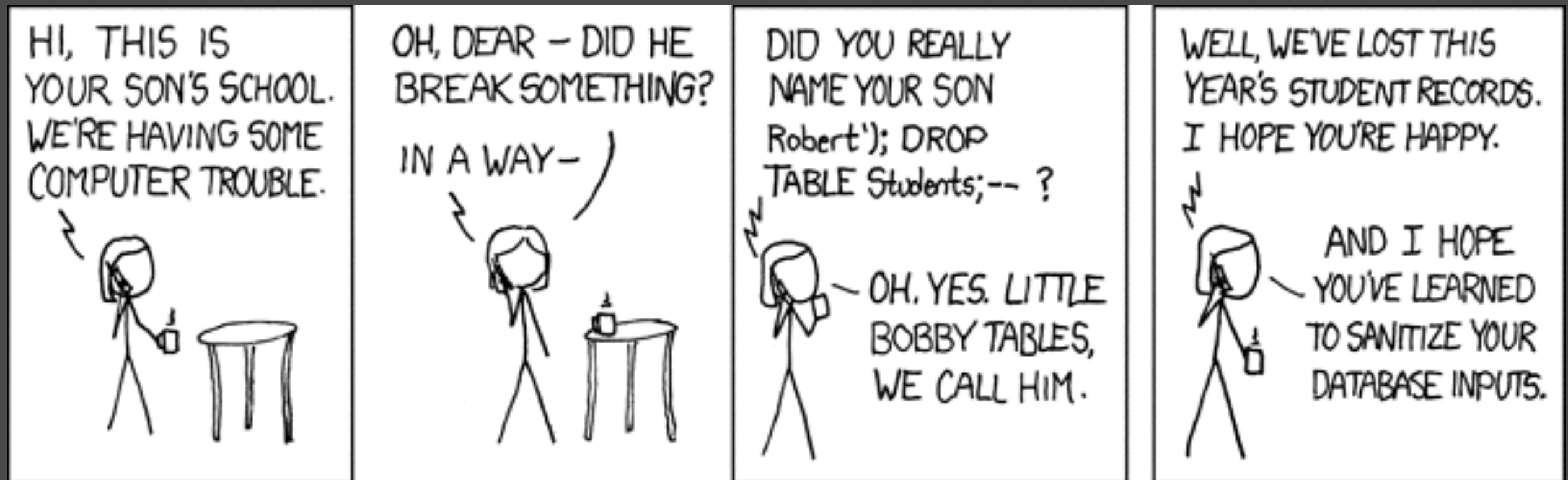
Is Rails secure?

- one framework is not more secure than another
- flawed coding = successful attack

Security Pitfalls

SQL Injection

SQL Injection



exploits of a mom » xkcd.com/327

SQL Injection

```
User.where("username LIKE '#{params[:q]}%'" )
```

```
SELECT  
  `users`.*  
FROM  
  `users` WHERE (username LIKE '%jerome%')
```

SQL Injection

```
params[:q] = "' ) UNION SELECT username, password,1,1,1 FROM  
users --"
```

```
SELECT  
  `users`.*  
FROM  
  `users`  
WHERE (username LIKE '%')  
UNION  
SELECT  
  username,  
  password,  
  1,1,1  
FROM users --%' )
```

SQL Injection

```
params[:q] = "' ) UNION SELECT username, password,1,1,1 FROM  
users --"
```

```
SELECT  
  `users`.*  
FROM  
  `users`  
WHERE (username LIKE '%')  
UNION  
SELECT  
  username,  
  password,  
  1,1,1  
FROM users --%' )
```

SQL Injection

countermeasure

```
User.where("username LIKE ?", "%#{params[:q]}%")
```

Cross-site Scripting (XSS)

XSS

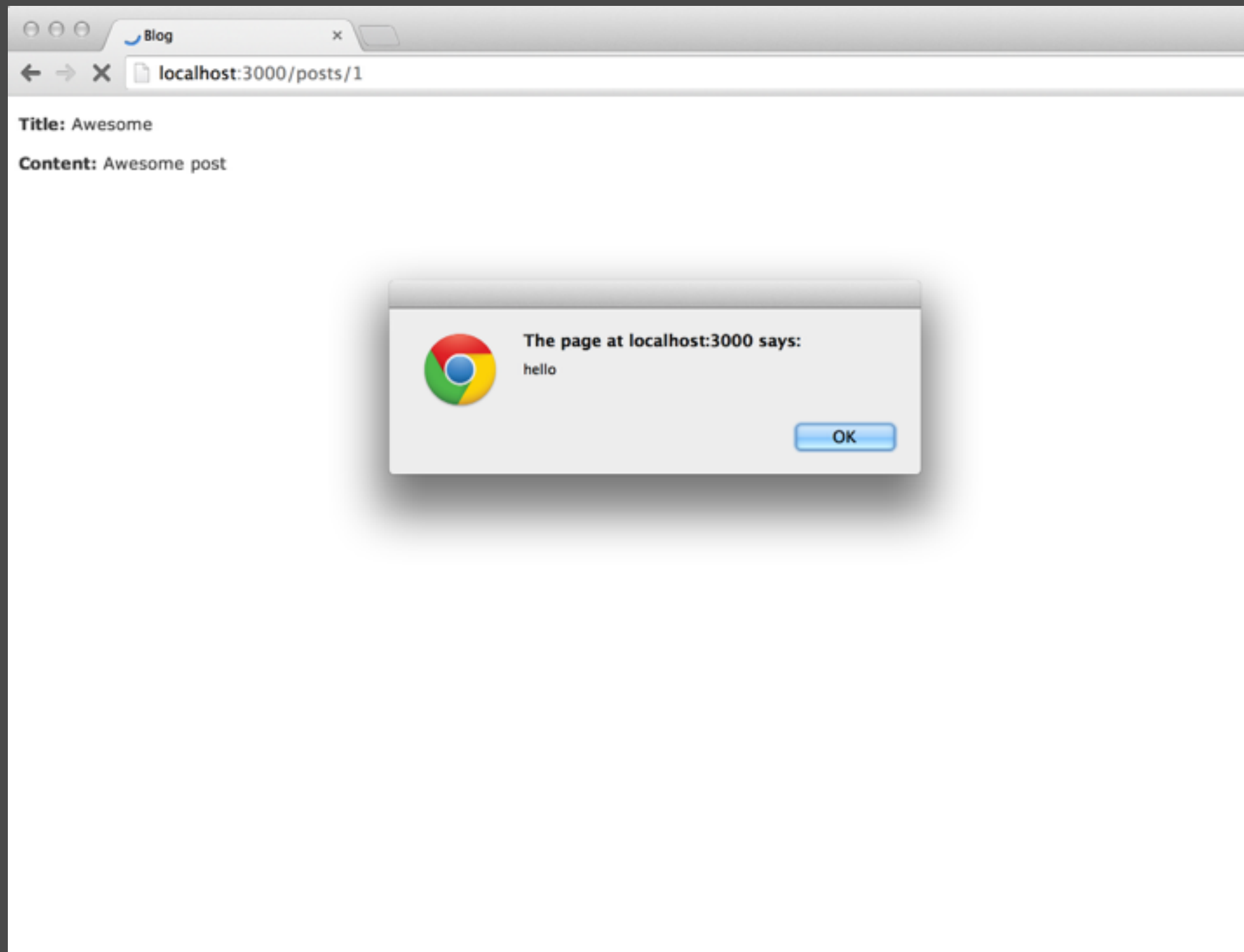
template code

```
<span>  
  <%= raw @post.content %>  
</span>
```

content from user

```
params[:content] = "<script>alert('hello');</script>"
```

XSS



XSS

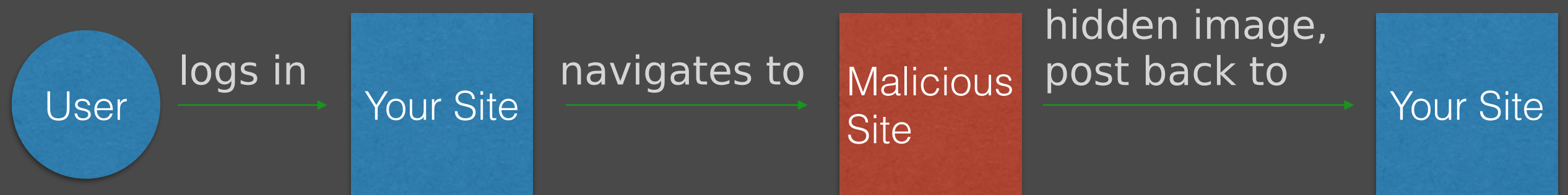
countermeasure

```
<span>
  <%= sanitize(@post.content, tags: %w(a),
    attributes: %w(href)) %>
</span>
```

- sanitize user input; use Rails method such as **sanitize**
- look for: **raw** and **.html_safe**

Cross-site Request Forgery (CSRF)

CSRF



- attacker sends request on victim's behalf
- doesn't depend on XSS

CSRF

countermeasure

- use HTTP (**GET**, **POST**) methods appropriately
- use Rails default CSRF protection

'match' in Routing

- **match** matches all HTTP verb and Rails CSRF protection doesn't apply to GET requests.

```
# Example in config/routes.rb  
match ':controller(/:action(/:id))(.:format)'
```

- route will allow GET method to delete posts

```
match '/posts/delete/:id',  
  :to => "posts#destroy",  
  :as => "delete_post"
```

'match' in Routing

countermeasure

```
# Example in config/routes.rb
# match ':controller(/:action(/:id))(.:format)'

match '/posts/delete/:id',
  :to => "posts#destroy",
  :as => "delete_post",
  :via => :delete
```

- use :via
- use correct HTTP verb in routing e.g. 'get', 'post', etc.

Mass Assignment

Mass Assignment

```
def create
  # ...

  @user = User.new(params[:user])

  # ...
end
```

```
<input type="text" name="user[username]" type="text" />
<input type="text" name="user[email]" type="text" />

<input type="text" name="user[admin]" value="1" type="text" />
```

Mass Assignment

countermeasure

blacklist attributes using **attr_protected**

```
class User < ActiveRecord::Base
  attr_protected :admin

  # ...
end
```

Mass Assignment

whitelist attributes using **attr_accessible**

```
class User < ActiveRecord::Base
  attr_accessible :username, :email
  # ...
end
```

```
config.active_record.whitelist_attributes = true
```

Mass Assignment

use **strong parameters**

```
def create
  # ...
  @user = User.new(params_user)
  # ...
end

private

def params_user
  params.require(:user).permit(
    :username,
    :email)
end
```

Secret Token

Secret Token

config/initializers/secret_token.rb

```
MyApp::Application.config.secret_token = '38d07e4b...'
```

this token is used to sign cookies that the application sets. for more info, read:

http://bit.ly/hack_rails_app_using_secret_token

Secret Token

countermeasure

config/initializers/secret_token.rb

```
MyApp::Application.config.secret_token = ENV['TOKEN']
```

* generate new secret by running **\$ rake secret**

Logging Parameters

Logging Parameters

```
Rails.application.config.filter_parameters += [:password, :ssn]
```

:password & :ssn will be replaced with “[FILTERED]”
logs

Scopes

Scopes

```
class User < ActiveRecord::Base
  has_many :posts
end
```

```
def edit
  @post = Post.find_by id: params[:id]
end
```

Scopes

countermeasure

```
def edit
  @post = current_user.posts.find_by id: params[:id]
end
```

- use authorization gem such as **cancan** or **pundit**
- look for **:edit**, **:update**, **:destroy** methods

Admin

Admin URL

<http://yourapp.com/admin>

“old habits die hard”

Admin URL

recommendation

- whitelist IP address
- separate application
- VPN or intranet access only
- use sub-domain

Conclusion

Conclusion

- keep your application up to date on all layers
- never trust any data from a user
- code review
- use **brakeman** gem - brakemanscanner.org

Conclusion

brakeman - Rails security scanner

```
$ brakeman -o report.html
```

```
+-----+-----+
| Warning Type          | Total |
+-----+-----+
| Cross Site Scripting  | 1     |
| SQL Injection         | 1     |
| Session Setting       | 1     |
+-----+-----+
```

brakeman demo

Further Resources

RoR Security Guide

More at rubyonrails.org: [Overview](#) | [Download](#) | [Deploy](#) | [Code](#) | [Screencasts](#) | [Documentation](#) | [Ecosystem](#) | [Community](#) | [Blog](#)



RailsGuides

[Home](#)[Guides Index](#)[Contribute](#)[Credits](#)

Ruby on Rails Security Guide

This manual describes common security problems in web applications and how to avoid them with Rails.

After reading this guide, you will know:

- ✓ All countermeasures *that are highlighted*.
- ✓ The concept of sessions in Rails, what to put in there and popular attack methods.
- ✓ How just visiting a site can be a security problem (with CSRF).
- ✓ What you have to pay attention to when working with files or providing an administration interface.
- ✓ How to manage users: Logging in and out and attack methods on all layers.
- ✓ And the most popular injection attack methods.

1 Introduction

Web application frameworks are made to help developers build web applications. Some of them also help you with securing the web application. In fact one framework is not more secure than another: If



Chapters

1. [Introduction](#)
2. [Sessions](#)
 - [What are Sessions?](#)
 - [Session id](#)
 - [Session Hijacking](#)
 - [Session Guidelines](#)
 - [Session Storage](#)
 - [Replay Attacks for CookieStore Sessions](#)
 - [Session Fixation](#)
 - [Session Fixation – Countermeasures](#)
 - [Session Expiry](#)
3. [Cross-Site Request Forgery \(CSRF\)](#)
 - [CSRF Countermeasures](#)
4. [Redirection and Files](#)
 - [Redirection](#)
 - [File Uploads](#)
 - [Executable Code in File Uploads](#)

RoR Security Google Groups

The screenshot shows the Google Groups interface for the "Ruby on Rails: Security" group. The page features a search bar at the top, navigation links on the left, and a list of security topics in the main content area.

Google Search for topics

Groups

My groups
Home
Starred

Favorites
Click on a group's star icon to add it to your favorites

Recently viewed
Devise
Ruby on Rails: Sec...
carrierwave
SimpleForm
Symfony users (DL...)

Recent searches
form field same na...

[Privacy](#) - [Terms of Service](#)

Ruby on Rails: Security Shared publicly
30 of many topics (67 unread)

- Denial of Service Vulnerability in Action View when using render :text (CVE-2014-0082)** (1)
By Aaron Patterson - 1 post - 4697 views
- Data Injection Vulnerability in Active Record (CVE-2014-0080)**
By Aaron Patterson - 1 post - 6198 views
- XSS Vulnerability in number_to_currency, number_to_percentage and number_to_human (CVE-2014-0081)**
By Aaron Patterson - 1 post - 9936 views
- [CVE-2013-6416] XSS Vulnerability in simple_format helper** (1)
By Aaron Patterson - 1 post - 1136 views
- [CVE-2013-6414] Denial of Service Vulnerability in Action View** (1)
By Aaron Patterson - 1 post - 762 views
- [CVE-2013-6415] XSS Vulnerability in number_to_currency** (1)
By Aaron Patterson - 1 post - 597 views
- [CVE-2013-6417] Incomplete fix to CVE-2013-0155 (Unsafe Query Generation Risk)** (1)
By Aaron Patterson - 1 post - 687 views
- [CVE-2013-4491] Reflective XSS Vulnerability in Ruby on Rails** (1)
By Aaron Patterson - 1 post - 671 views
- [CVE-2013-1854] Symbol DoS vulnerability in Active Record** (1)
By Aaron Patterson - 1 post - 3434 views
- [CVE-2013-1857] XSS Vulnerability in the `sanitize` helper of Ruby on Rails** (1)
By Aaron Patterson - 1 post - 4461 views
- [CVE-2013-1856] XML Parsing Vulnerability affecting JRuby users** (1)
By Aaron Patterson - 1 post - 2690 views

Railscasts on Security


RAILSCASTS
Ruby on Rails Screencasts

Search Episodes

Browse Episodes

RailsCasts Pro

Applied Filters: Security x



AUTHORIZATION FROM SCRATCH PART 2


EPISODE #386 – Oct 11, 2012 – 38 comments

Authorization from Scratch Part 2 ☆

This finishes the series on building authorization from scratch by refactoring the permission logic into a DSL restricting authorization with attributes, and combining with `strong_parameters` to protect params. (20 minutes)

Watch Episode

Read Episode



AUTHORIZATION FROM SCRATCH PART 1


EPISODE #385 – Oct 07, 2012 – 28 comments

Authorization from Scratch Part 1 ☆

Authorization can be difficult to implement and test because it often involves complex logic that exists throughout the entire app. Here I demonstrate how to test and implement authorization from scratch. (15 minutes)

Watch Episode

Read Episode



BRAKEMAN


EPISODE #358 – Jun 15, 2012 – 13 comments

Brakeman ☆

The Brakeman gem will scan the Ruby code of a Rails application and alert you to common security vulnerabilities. (8 minutes)

Watch Episode

Read Episode



Adding SSL

EPISODE #357 – Jun 08, 2012 – 31 comments

Questions?