

Brakeman

<http://github.com/presidentbeef/brakeman>

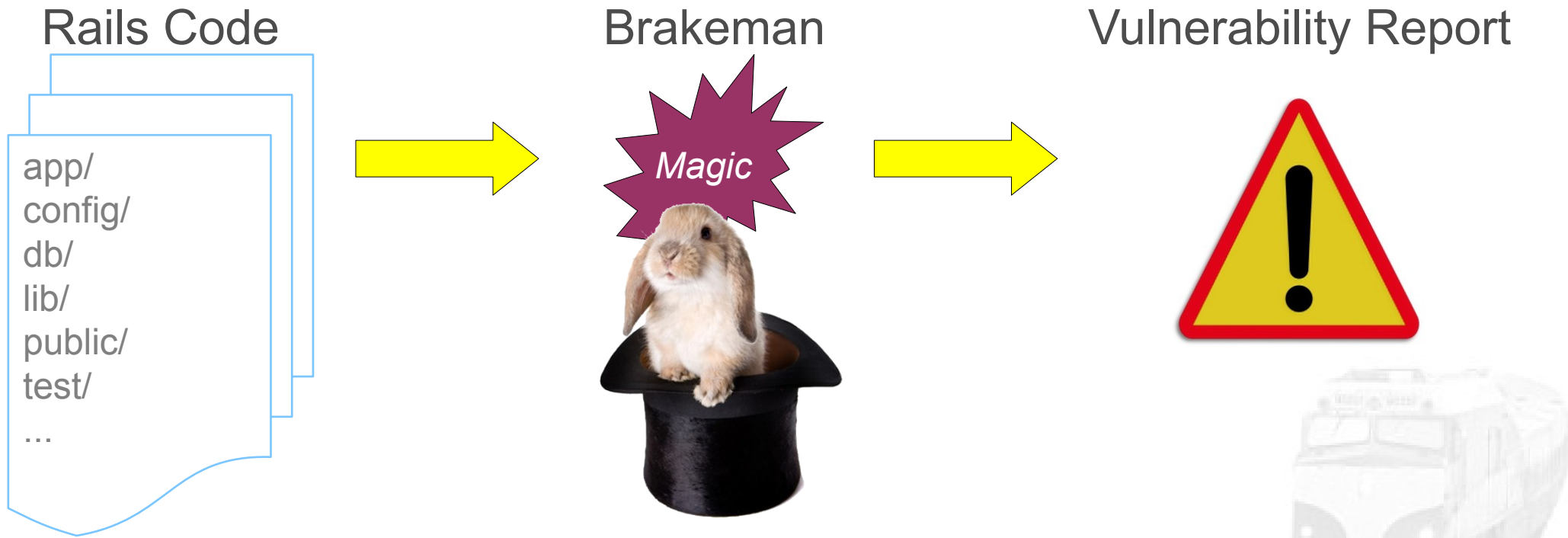


What is Brakeman?

A static analysis vulnerability scanner
for
Ruby on Rails applications



Basic Idea



How to Use Brakeman

```
gem install brakeman
```

```
brakeman my_rails_app
```



Note

- No configuration
- No deployment
- No web server
- No database
- Not even Rails



Example: Cross Site Scripting

`Results for <%= params[:query] %>`



Example: Cross Site Scripting

Confidence	Template	Warning Type	Message
High	search/result (SearchController#result)	Cross Site Scripting	Unescaped parameter value near line 3: params[:query]



Example: Cross Site Scripting

Confidence	Template	Warning Type	Message
High	search/result (SearchController#result)	Cross site Scripting	Unescaped parameter value near line 3: params[:query] <pre>1 <h1>Search Results</h1> 2 <p> 3 Results for <%= params[:query] %> 4 </p></pre>



Going Deeper – Partial

```
<head>  
  <title>  
    <%= title %>  
  </title>  
</head>
```



Going Deeper – View

```
<%= render :partial => 'header',  
  :locals => { :title => @animal[:name] }  
%>
```



Going Deeper – Controller

```
class MammalController < AnimalController
  def cavort
  end
end
```



Going Deeper – Filter

```
class AnimalController < ApplicationController
  before_filter :set_animal, :only => :cavort

  private
  def set_animal
    @animal = params[:animal]
  end
end
```



Going Deeper – Warning

Confidence	Template	Warning Type	Message										
Weak	mammal/_header (Template:mammal/cavort)	Cross Site Scripting	Unescaped parameter value near line 3: params[:animal][:name] <table border="1"><tr><td>1</td><td><head></td></tr><tr><td>2</td><td><title></td></tr><tr><td>3</td><td><%= title %></td></tr><tr><td>4</td><td></title></td></tr><tr><td>5</td><td></head></td></tr></table>	1	<head>	2	<title>	3	<%= title %>	4	</title>	5	</head>
1	<head>												
2	<title>												
3	<%= title %>												
4	</title>												
5	</head>												



(More) Examples of Vulnerabilities which Brakeman Can Detect



SQL Injection

```
Animal.find(:all,  
  :conditions => "name like '%#{params[:q]}%'")
```

Confidence	Class	Method	Warning Type	Message
High	AnimalController	find	SQL Injection	Possible SQL injection near line 5: Animal.find(:all, :conditions => ("name like '%#{params[:q]}%'"))...



Command Injection

```
system "ls #{params[:dir]}"
```

Confidence	Class	Method	Warning Type	Message
High	AnimalController	list_animals	Command Injection	Possible command injection near line 10: system("ls #{params[:dir]}")



File Access

```
send_file "animals/#{params[:file]}"
```

Confidence	Class	Method	Warning Type	Message
High	AnimalController	send_animal	File Access	Parameter value used in file name near line 14: send_file("animals/#{params[:file]}")



Mass Assignment

```
Animal.new(params[:animal])
```

Confidence	Class	Method	Warning Type	Message
High	AnimalController	create	Mass Assignment	Unprotected mass assignment near line 17: Animal.new(params[:animal])



Validation Regexes

```
validates_format_of :animal_name,  
  :with => /^[a-z]+$/i
```

Confidence	Model	Warning Type	Message
High	Animal	Format Validation	Insufficient validation for 'animal_name' using /^[a-z]+\$/i. Use \A and \z as anchors near line 2



Code Evaluation

```
eval(params[:action])
```

Confidence	Class	Method	Warning Type	Message
High	MammalController	act	Dangerous Eval	User input in eval near line 10: eval(params[:action])



And more...

- Unsafe redirects
- Default routes
- Forgery protection
- Session settings
- Dynamic render paths

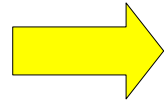


What Brakeman Does

Parse Rails code

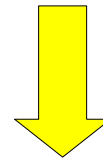
```
<h2><%= l(:label_home)
%></h2><div
class="splitcontentleft">
<%= textilizable
Setting.welcome_text %> <
% if @news.any? %> <div
class="news box">
<h3><
%=l(:label_news_latest)
%></h3> <%=
render :partial =>
'news/news', :collection
=> @news %>
```

via RubyParser



S-Expressions

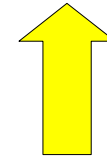
```
s(:call, nil, puts,
(:arglist,
s(:call, nil, 'hello',
s(:arglist))))
```



Wrangle and Mangle



Generate Report



Run Checks



Simple Data Flow

```
name = params[:animal][:name]
```

```
Animals.find(:all,  
  :conditions => "name like '%#{name}%'")
```



Simple Data Flow

```
name = params[:animal][:name]
```

```
Animals.find(:all,  
  :conditions => "name like '%#{params[:animal][:name]}%'")
```



More Complex Data Flow

```
config = { :path => "/tmp/", :prefix => "temp_" }  
file_name = params[:file]  
args = [file_name]  
File.read(config[:path] + config[:prefix] + args[0])
```



More Complex Data Flow

```
config = { :path => "/tmp/", :prefix => "temp_" }  
file_name = params[:file]  
args = [params[:file]]  
File.read("/tmp/temp_" + params[:file])
```



Processing Overview

- Configuration
- Initializers
- Libraries
- Routes
- Views
- Models
- Controllers



Processing Controllers

For each action:

- Process `before_filters`
- Process action
- Propagate variables to view
 - If there is a call to render, process it
 - Otherwise, process default view
- Process view



Then Run Checks

- A "check" is just a Ruby class
- Each check operates independently
- Checks have access to all accumulated information
- Checks emit warnings



Simple Check

```
require 'checks/base_check'

#This check looks for calls to eval, instance_eval, etc. which include
#user input.
class CheckEvaluation < BaseCheck
  Checks.add self

  #Process calls
  def run_check
    calls = tracker.find_call nil, [:eval, :instance_eval, :class_eval, :module_eval]

    calls.each do |call|
      process_result call
    end
  end

  #Warns if result includes user input
  def process_result result
    if include_user_input? result[-1]
      warn :result => result,
          :warning_type => "Dangerous Eval",
          :message => "User input in eval",
          :code => result[-1],
          :confidence => CONFIDENCE[:high]
    end
  end
end
```



Produce Report

- Turn warnings into something readable
- Current formats:
 - Text
 - HTML
 - CSV



Caveats

- Rails 2.x only
- Only works on "typical" applications
- Reported line numbers can be inaccurate
- Confidence levels are only a guess
- Lots of room for improvement

