CSC 4800
Web Application Security

Don’t trust user input
• regular form fields
• hidden form fields
• cookies
• URL parameters
• POST data
• HTTP headers
• Ajax requests

Using Scoped Queries
```ruby
class ContactsController < ApplicationController
  before_filter require_signin
  def show
    @contact = Contact.find params[:id]
  end
end
```
This would be accessed via contacts/show/42
How do we know contact #42 belongs to the current user?

Form Validation
• We have already mentioned that client side form validation is insufficient.
• User can turn off javascript, or create (and submit) his own form.
• Client side validation is a convenience.

Mass Assignment
• Rails allows syntax like `User.create(params[:user])`
• A malicious user could create his own form with extra fields
• Solution: use attr_protected, which prevents bulk assignment
• The flip-side is to use attr_accessible
  – All other attributes will then be protected.
SQL Injection

- see example
- This is bad
  `User.find(:first, :conditions => { :login => '#params[:login]' AND :password => '#params[:password]' })`  
- What if user enters, for password: ' or login='bob' and password != '
- The resulting SQL is:
  `SELECT * FROM users WHERE (login='' and password='' or login='bob' and password != '') LIMIT 1`

SQL Injection

- Solution: simple...never include tainted data directly in SQL statements
- Many safe solutions
  `User.find(:first, :conditions => { :login => params[:login], :password => params[:password] })`
  `User.find(:first, :conditions => [ "login=? AND password=?", params[:login], params[:password] ])`

Session Fixation

- intentionally set a user's session key to a known value
- Some browsers have buggy cookie implementations that allow one site to plant cookies to be delivered to another site. (Cross site cooking)
- Solution – generate a new session ID when a user authenticates.

Cross Site Scripting (XSS)

- Can happen when unescaped user data is included in HTML output.
- Solution is simple...escape user input!
  - In rails, use the h helper.

Hashing passwords

- Why do we hash passwords?
- Passwords still must be sent in the clear from browser to server (unless additional encryption is used)
- But if you are only storing hashed passwords, and an attacker gets to your database, they only have the hashed passwords
- The actual password is never stored anywhere on the system.

Silencing Logs

- By default, at least in development mode, all request parameters are logged.
- Mostly ok, but not for things like credit card info.
- If you are hashing passwords, but the info ends up in the log, you have defeated the advantage of hashing.
- Payment processors prohibit you from storing credit card information.
Silencing Logs

class OrdersController < ApplicationController
  filter_parameter_logging :cc_number,
  :cvv,
  :cd_date

SSL

• Use SSL to transmit sensitive information.

Same-Origin Policy

• This rule is enforced by most browsers.
• JavaScript code may only issue Ajax requests to URLs from the same domain as the original page.
• Actually, combination of domain, port, and protocol.