Rails 5 Security Cheat Sheet

by dwapi via cheatography.com/43665/cs/13083/

SQL Injection

Definition:

SQL injection is the placement of malicious code in SQL statements, via web page input.

Example:

Malicious user inputs SQL code as vaue for text input

Threat Level (Medium-High):

ActiveRecord, in most cases, protects against SQL Injection by default, however, there are ways in which it can be used insecurely which can lead to SQL Injection.

Rails Fix:

Avoid using find_by_sql
Do not pass params directly into queries use '?' vars
Explicitly force IDs to_i for queries
Use "Strong Parameters" in Controller

Potentially Dangerous Methods:

Calculations (average, sum, maximum...){{nl}exists?(id)
delete_all / destroy_all
find_by(id)
https://rails-sqli.org/

This is one of the most common web hacking techniques.
Malicious SQL could destroy your database.

Cross-Site Scripting (XSS)

Definition:

XSS is a code injection attack that allows an attacker to execute malicious JavaScript in another user's browser The only way for the attacker to run his malicious JavaScript in the victim's browser is to inject it into one of the pages that the victim downloads from the website.

Persistent XSS

Malicious JS has been saved to DB by attacker. Is executed when victim loads page

Reflected XSS

In a reflected XSS attack, the malicious string is part of the victim's request to the website. The website then includes this malicious string in the response sent back to the user. Think Phishing.

Threat Level (High for Persistent XSS)

Data must be sanitized before being saved to DB

Rails Fix:

Data must be sanitized before being saved to DB

https://excess-xss.com/

Session Hijacking

Definition:

Stealing a user's session ID lets an attacker use the web application in the victim's name.

Example:

Man in the middle sniffs out valid session id

Threat Level (Medium-Low):

Man in the middle attack

Rails Fix:

Expire Sessions
Use https to thwart man-in-the-middle attacks
Call reset_session when logging users in and out to avoid session fixation.
Sanitize user input to avoid XSS.
Use Devise (it will automatically expire sessions on sign in and sign out)

Rails Fix (Enabled by default)

EncryptedCookieStore: Session is encrypted before being stored in cookie (config/secrets.yml)

Session Fixation:

Using a fixed/permament session id. Call reset_session after login/logout to prevent this.

Cross-Site Request Forgery (CSRF) (Built In)

Definition:

An attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.

Example:

Phishing example. User clicks on link to page that looks like legit site. Victim is tricked into submitting a malicious request. It inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf.

Threat Level (LOW):

Modern browsers enforce same-origin policy restrictions on scripts.

Rails Fix (Enabled by default)

Rails protects against CSRF attacks by default by including a token named authenticity_token within HTML responses
That token is also stored in user session and they are compared when Request is made.
To confirm it's enabled verify protect_from_forgery is in ApplicationController

Rails 5 Security Cheat Sheet

by dwapi via cheatography.com/43665/cs/13083/

Cross-Site Request Forgery (CSRF) (Built In) (cont)

Developer Fix:

Use GET and POST properly (CSRF is on POST only)

Develop Responsibly

Filter params saved in logs

config.filter_parameters << :password, :credit_card

Do not Redirect based on a URL in the Request

BAD: http://www.example.com/site/redirect?to=www.attacker.com
redirect_to(params[:to])

Think through file uploads

Check extensions. Beware executeable files

Restrict File Downloads

Make sure users cannot download arbitrary files.
send_file('/var/www/uploads/' + params[:filename])

Brute Force login attacks

Think about using a CAPTCHA