RAILS

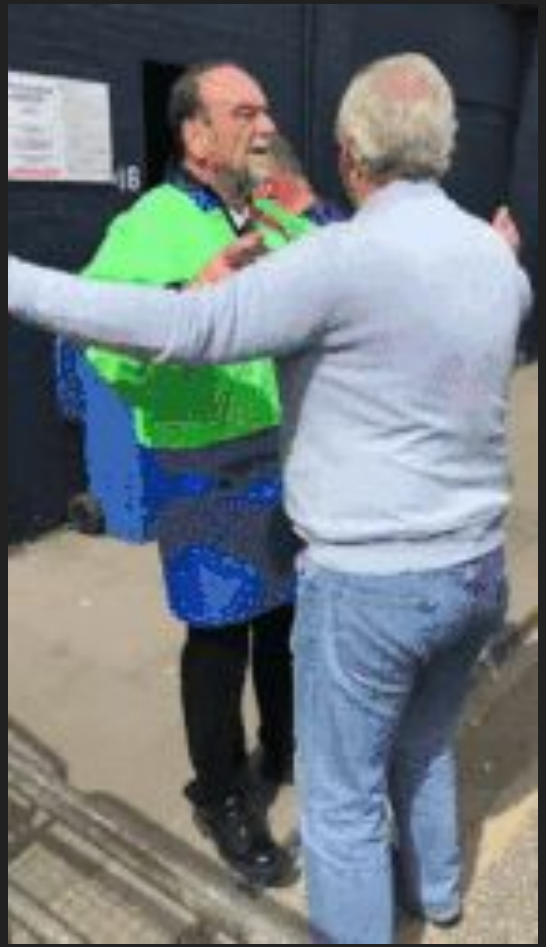
¿Quién soy?

- Ruby on Rails developer +/- 8 años de exp
- “Fulestá” por amor al arte
- Trabajo en Ideaware
- NO soy experto en ciberseguridad
- Cine de Tarantino 👌🏽

Basado +/- en lo que escribí en el artículo del mismo título

https://bit.ly/rails-te-protege

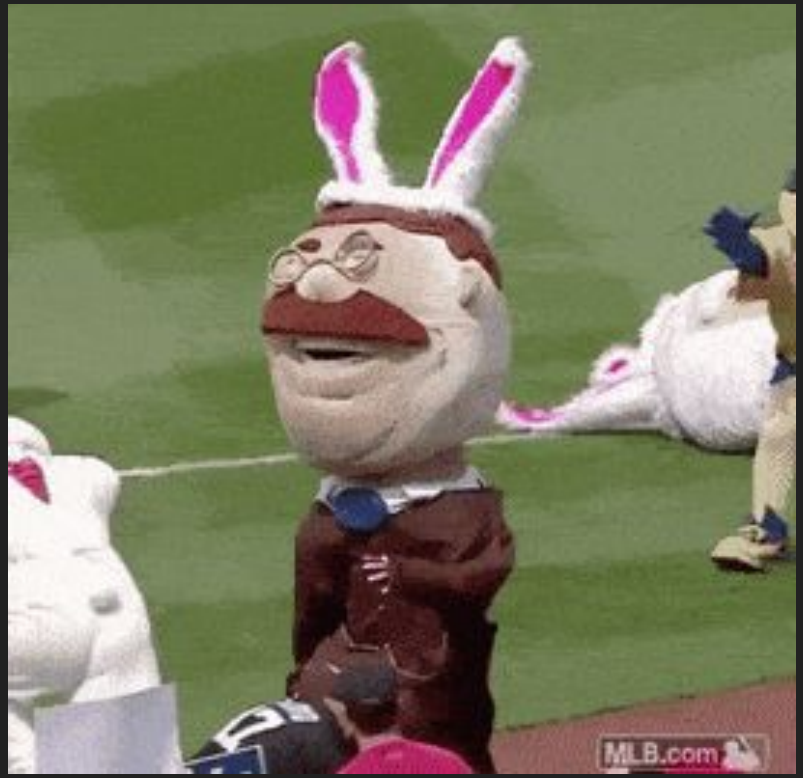
Seguridad en la Web



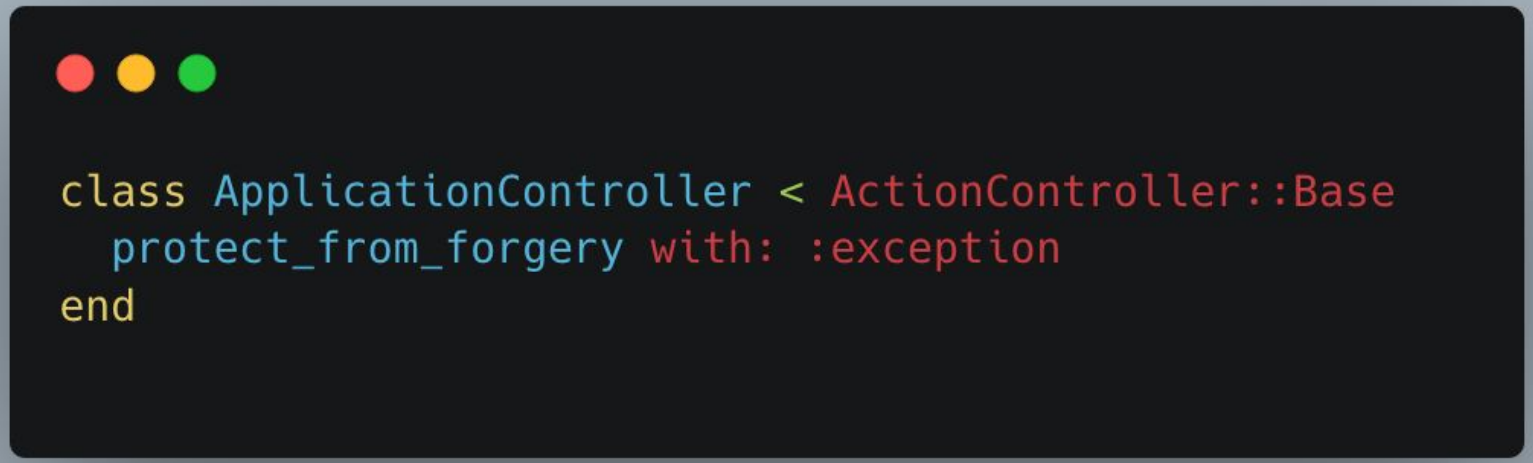
Fuentes Fiables

- OWASP -> https://owasp.org/
- Guías oficiales del framework que usas
- Mozilla Developer Network (sección HTTP)

Cross-Site Request Forgery !!

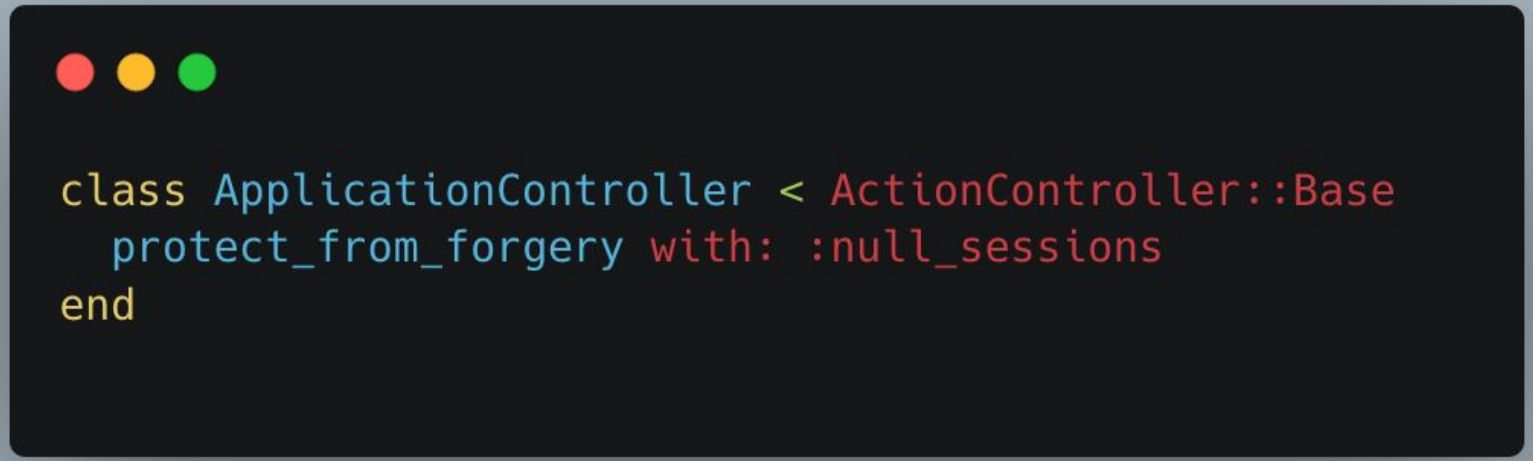


- Usa peticiones HTTP como se debe:
  - GET para solo lectura
  - POST, PUT, DELETE para modificar datos
- Revisa que ofrece tu framework
  - ¿protección por defecto o hay que traer librerías?


class ApplicationController < ActionController::Base
protect_from_forgery with: :exception
end

En Formularios y Peticiones Ajax

```
<head>
  <%= csrf_meta_tags %>
</head>

# Produce:

<meta name='csrf-token' content='THE-TOKEN'>

###

# Insertado en formularios

<input name="authenticity_token" type="hidden" value="xxxxxxxxxxx" />
```


class ApplicationController < ActionController::Base
protect_from_forgery with: :null_sessions
end

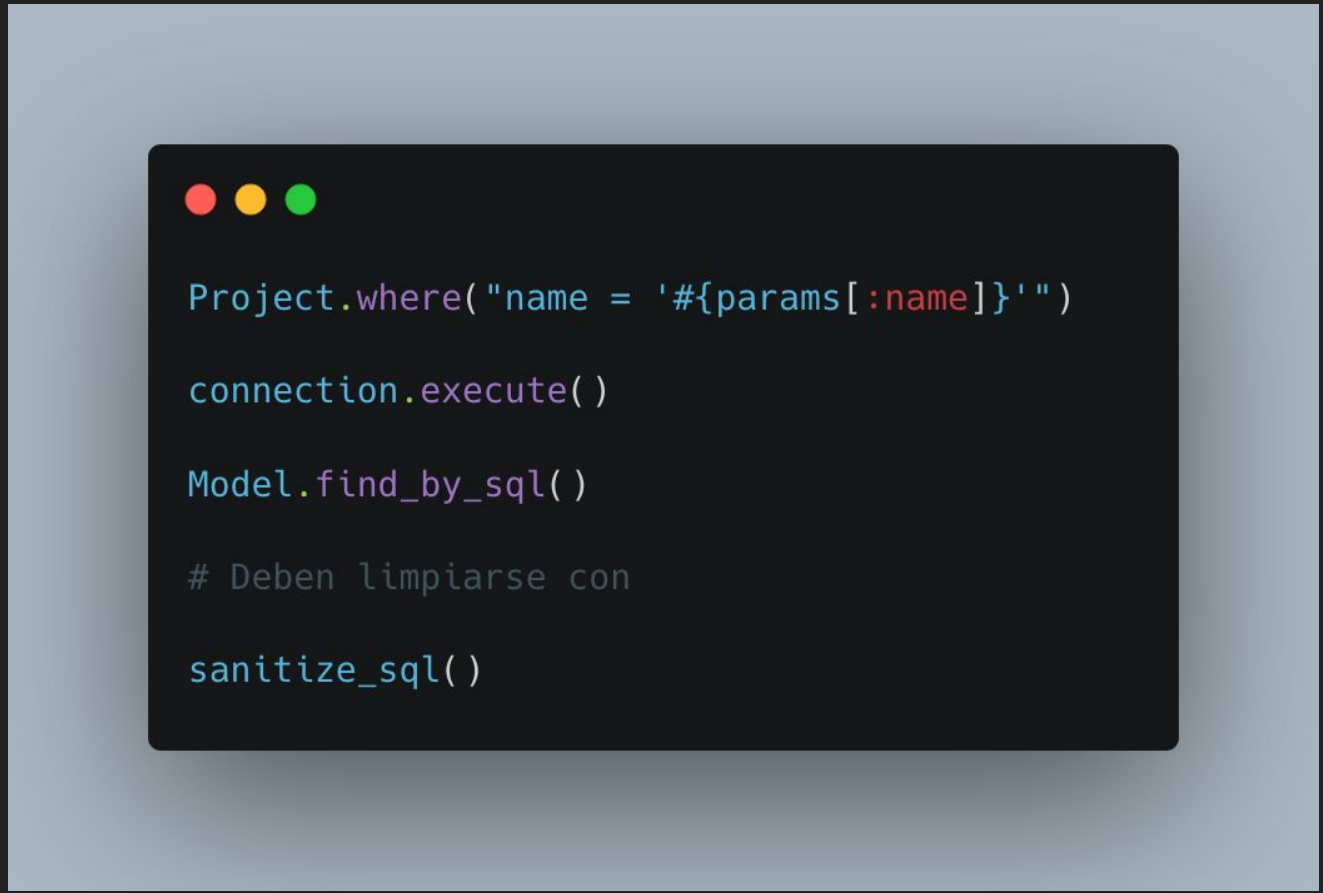
Cross-Site Scripting o Inyecciones 

- JavaScript Injection
- SQL Injection
- CSS Injection
- Command Line Injection
- Header Injection

“Cuando se limpie, verifique o proteja algo, prefiere listas permitidas que listas restringidas.”

- Guías de Rails

SQL Injection 


Project.where("name = '#{params[:name]}'")
connection.execute()
Model.find_by_sql()
# Deben limpiarse con
sanitize_sql()

```ruby
Model.where("zip_code = ? AND quantity >= ?", entered_zip_code, entered_quantity).first

values = { zip: entered_zip_code, qty: entered_quantity }
Model.where("zip_code = :zip AND quantity >= :qty", values).first
```

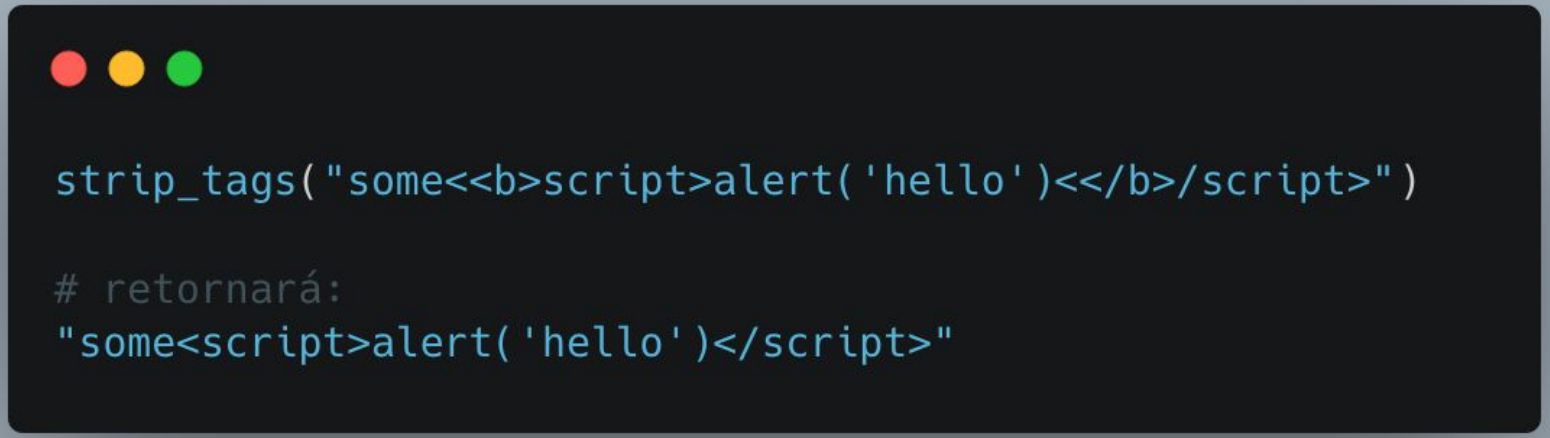
“Haz un hábito el pensar en las consecuencias de seguridad cuando permitas SQL externo sin limpiar.”

- Guías de Rails

JavaScript Injection 

“Cuando se limpie, verifique o proteja algo, prefiere listas permitidas que listas restringidas.”

- Guías de Rails


strip_tags("some<script>alert('hello')</script>")
# retornará:
"some<script>alert('hello')</script>"

```ruby
tags = %w(a acronym b strong i em li ul ol h1 h2 h3 h4 h5 h6 blockquote br cite sub sup ins p)

s = sanitize(user_input, tags: tags, attributes: %w(href title))
```

Content Security Policy 🛑

```ruby
Rails.application.config.content_security_policy do |policy|
  policy.default_src :self, :https
  policy.font_src    :self, :https, :data
  policy.img_src     :self, :https, :data
  policy.object_src  :none
  policy.script_src  :self, :https
  policy.style_src   :self, :https
  # If you are using webpack-dev-server then specify webpack-dev-server host
  policy.connect_src :self, :https, "http://localhost:3035", "ws://localhost:3035" if
Rails.env.development?

  # Specify URI for violation reports
  # policy.report_uri "/csp-violation-report-endpoint"
end
```

- Usar HTTP como se debe
- Revisar recomendaciones de OWASP
- Limpiar todo lo que se recibe de un usuario
- Listas permitidas en lugar de restringidas
- Mantener librerías actualizadas

Fuentes: “trust me, bro”

- “Chuletas” de OWASP -> https://cheatsheetseries.owasp.org/
- SQL Injection en MDN -> https://developer.mozilla.org/en-US/docs/Glossary/SQL_Injection
- Sección Seguridad en Guías de Rails -> https://guides.rubyonrails.org/security.html
-

¡Gracias!