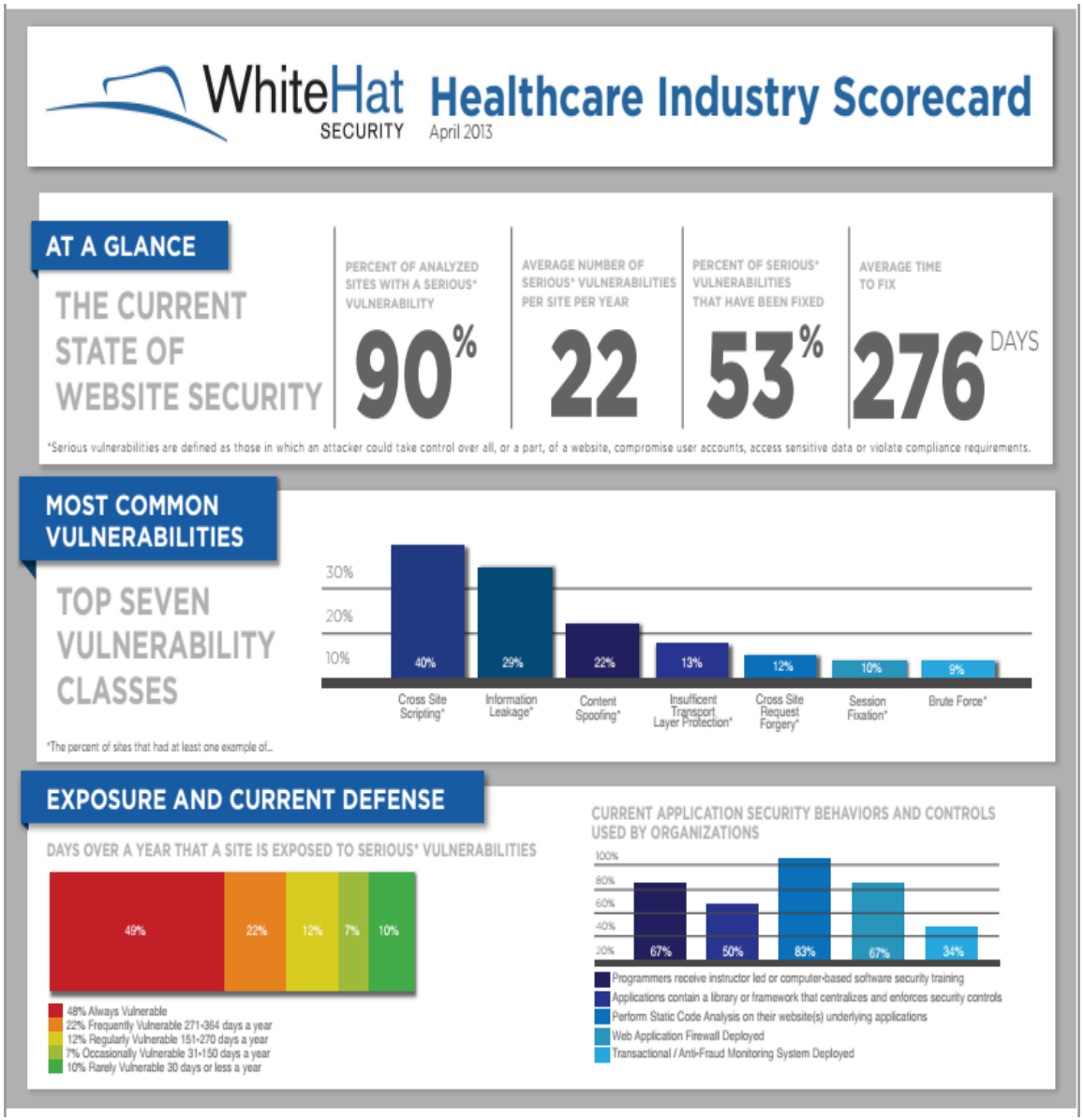
Seguridad, Performance, Gemas

Rodrigo Rodriguez
rorodr@gmail.com
Skype: rarodriguezr

Problemas comunes



WhiteHat SECURITY

Healthcare Industry Scorecard

April 2013

AT A GLANCE

THE CURRENT STATE OF WEBSITE SECURITY

PERCENT OF ANALYZED SITES WITH A SERIOUS* VULNERABILITY	AVERAGE NUMBER OF SERIOUS* VULNERABILITIES PER SITE PER YEAR	PERCENT OF SERIOUS* VULNERABILITIES THAT HAVE BEEN FIXED	AVERAGE TIME TO FIX
90%	22	53%	276 DAYS

*Serious vulnerabilities are defined as those in which an attacker could take control over all, or a part, of a website, compromise user accounts, access sensitive data or violate compliance requirements.

MOST COMMON VULNERABILITIES

TOP SEVEN VULNERABILITY CLASSES

*The percent of sites that had at least one example of...

EXPOSURE AND CURRENT DEFENSE

DAYS OVER A YEAR THAT A SITE IS EXPOSED TO SERIOUS* VULNERABILITIES

CURRENT APPLICATION SECURITY BEHAVIORS AND CONTROLS USED BY ORGANIZATIONS

SQL Injection

- Es un tipo de ataque que se enfoca en vulnerar el SQL, con el fin de mostrar al usuario más información de la debida.
- Por lo general, los ataques se efectúan de tal forma que no se guarda el ataque en la BD, sino que se utiliza en búsquedas, listados, entre otros llamados que requieren parámetros.

SQL Injection

```
name = params[:name]
@users = User.where(“name like ’” + name + “’”)
```

Que pasaría si el parámetro “name” fuera algo como esto:

```
"') OR 1#"
```

```
Algo’); TRUNCATE TABLE USERS; #--
```

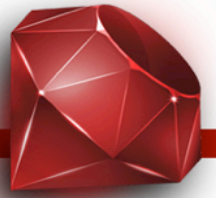
SQL injection

- Una aplicación de ejemplo:
  - https://github.com/presidentbeef/inject-some-sql
- Un video interesante con una explicación de una versión antigüa de Rails
  - http://railscasts.com/episodes/25-sql-injection

SQL injection: Prevención

- Ser precavido a la hora de generar sus consultas
  - Si ocupamos parámetros, hay que usar la forma recomendada por el framework, denominados Prepared statements (usando ? Para los parámetros). Ej:
    - User.where("name = ?", params[:name])
  - Usar Model.find(params[:id].to_i) o Model.where(name: params[:name])
- Usar herramientas como Brakeman o dawnscanner

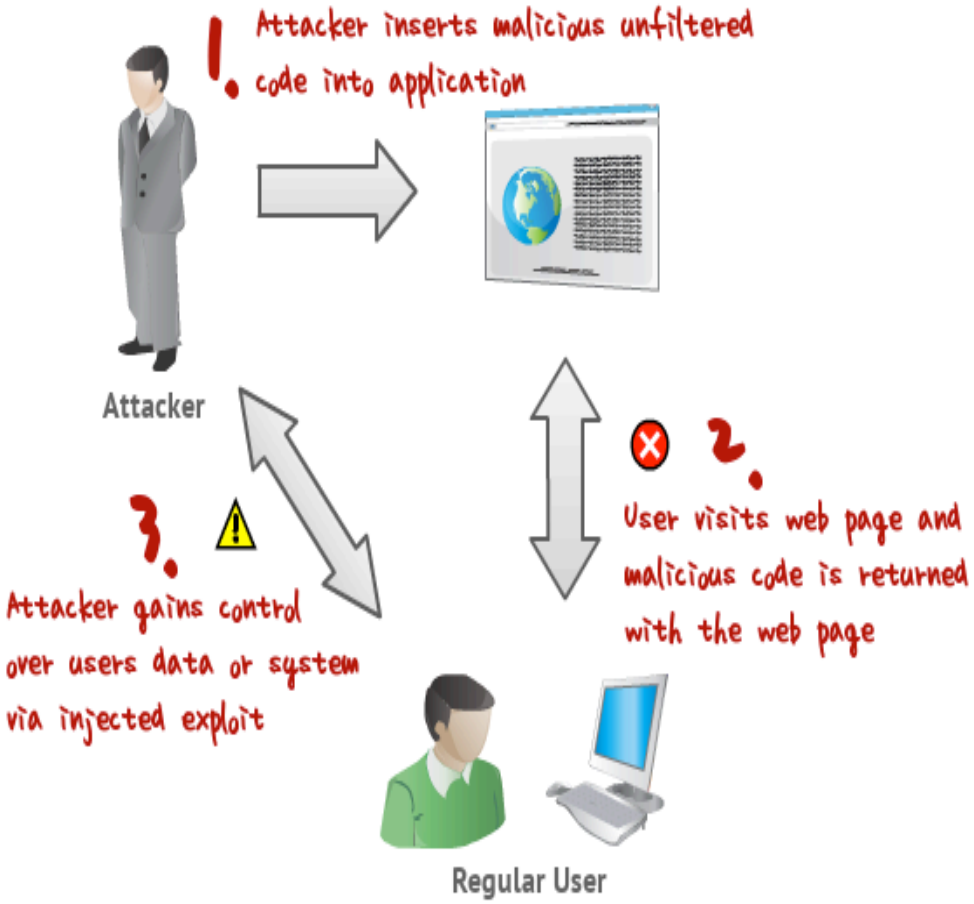
SQL injection: Prevención



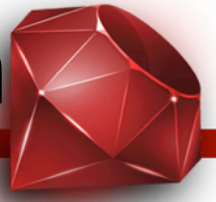
- Utilizar el método “sanitize_sql” para limpiar valores inválidos en el SQL
  - http://api.rubyonrails.org/classes/ActiveRecord/Sanitization/ClassMethods.html#method-i-sanitize_sql

Cross Site Scripting

- XXS permite al atacante ejecutar código javascript en las páginas vistas por otros usuarios

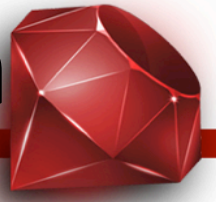


Cross Site Scripting: Prevención



- Rails 4 por defecto incluye métodos para limpiar el HTML que se muestra en pantalla, sin embargo hay que tratar de NUNCA utilizar “RAW” a no ser que sean datos que NO provengan del cliente
- Si se ocupa renderizar un HTML desde el cliente (ej. Código de un TinyMCE), utilizar SIEMPRE el método “sanitize”
- Se puede utilizar esta librería: https://github.com/twitter/secureheaders#readme para añadir algunos encabezados de seguridad y prevenir XSS.

Cross Site Scripting: Prevención



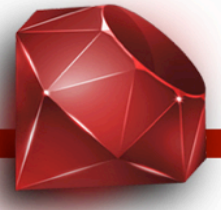
- Rails 4 por defecto incluye métodos para limpiar el HTML que se muestra en pantalla, sin embargo hay que tratar de NUNCA utilizar “RAW” a no ser que sean datos que NO provengan del cliente
- Si se ocupa renderizar un HTML desde el cliente (ej. Código de un TinyMCE), utilizar SIEMPRE el método “sanitize”
- Se puede utilizar esta librería: https://github.com/twitter/secureheaders#readme para añadir algunos encabezados de seguridad y prevenir XSS.

Mass Assignment

- Es un ataque que existe mayormente en Rails pues toma provecho de la metodología simple de asignar atributos en Rails

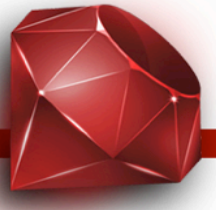
```
1  attrs = {:first => "John", :last => "Doe", :email => "john.doe@example
2  user = User.new
3  user.first = attrs[:first]
4  user.last  = attrs[:last]
5  user.email = attrs[:email]
6  user.first #=> "John"
7  user.last  #=> "Doe"
8  user.email #=> "john.doe@example.com"
```

Mass Assignment: Prevención



- Rails 4 provee un mecanismo nativo de prevención de estos ataques, por medio de la strong parameter (opción “permit” de los controladores), el cual es quien determina cuales atributos son “permitidos” para actualizarse
- También se puede complementar con: https://github.com/rails/protected_attributes, el cual era la métodología que se usaba en Rails 3.

Cross-Site Request forgery (XSRF)



- Es un tipo de exploit para Websites en el cual se ejecutan acciones (ejecución de POST/PUT) de un sitio, desde lugares remotos (ejemplo, ejecutar acciones al carga una imagen).
- Rails por defecto incluye “protect_from_forgery” para prevenir este tipo de ataques (únicamente aplica para POST/PUT/PATCH/DELETE).

Brakeman

- Librería que analiza de manera estática el código para prevenir fallas de seguridad
- Permite generar reportes bastante buenos sobre posibles problemas de seguridad en el código
  - gem install brakeman
  - brakeman –o output.html

Desempeño en Rails

- Existen diversos aspectos que considerar en el desempeño de una aplicación de Rails, siendo las más importantes:
  - Tiempo de carga de los Assets (imágenes, CSS, JS)
  - Tiempo de acceso a los datos que se utilizan en el sistema
  - Tiempo de análisis de la información presentada
  - Tiempo de carga de la página

Desempeño en Assets

- Assets pipeline genera un único archivo para los JS y CSS, sin embargo este no siempre está "minificado", por lo que es conveniente utilizar librerías externas que hacen el proceso automáticamente
  - http://edgeguides.rubyonrails.org/asset_pipeline.html#css-compression
  - rack-zippy
  - htmlcompressor

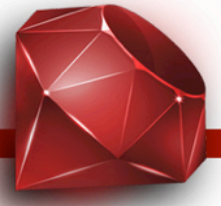
Desempeño en BD (1/2)

- La primera parte para mejorar el desempeño en la parte de BD hay diversas formas, entre ellas:
  - Agregar índices a las tablas (migración)
  - Agregar caché de BD (depende del sistema gestor de BD)
  - Eliminar consultas innecesarias

Desempeño en BD (2/2)

- Utilización de caché de contador
  - Se agrega un campo integer en la tabla y en los modelos se aplica algo semejante a:
    - `belongs_to :post, counter_cache: => true`
- Optimización de consultas
  - Se puede utilizar gemas como bullet para ayudar a optimizar las consultas (problema de N+1 consultas)

Problema de N + 1 consultas



- Rails intenta ejecutar de manera “óptima” las consultas, de modo que únicamente se efectúan aquellas consultas que realmente se ocupan (Lazy Load). Lo que provoca en algunos casos que se ejecuten N consultas a la BD.
- Para evitar esto, se puede utilizar ‘includes’ en vez de ‘joins’ a la hora de hacer consultas, con el fin de forzar “eager loading” (cargar todo de una vez)

Manejo de Caché (1/5)

- La mejor forma para optimizar el tiempo de carga es la utilización de caché
  - **Caché a Bajo Nivel:** Manejar caché a nivel de modelos

```
def competing_price
    Rails.cache.fetch("#{cache_key}/
competing_price", expires_in: 12.hours) do
      Competitor::API.find_price(id)
    end
  end
```

Manejo de Caché (2/5)

- **Caché de fragmento:** Permite manejar caché de una sección de un erb con el fin de optimizar la carga del fragmento.

```
<% cache('all_available_products') do %>
  All available products:
  <% Store.all.each do |s| %><%=s.name%><%end
%>
<%end%>
```

Manejo de Caché (3/5)

- Expirar fragmento automáticamente:

```
module ProductsHelper
  def cache_key_for_products
    count          = Product.count
    max_updated_at =
Product.maximum(:updated_at).try(:utc).try(:to_s
, :number)
    "products/all-#{count}-#{max_updated_at}"
  end
end
```

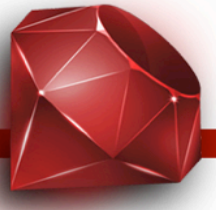
Manejo de Caché (4/5)

- **Caché de página:**
  - Permite generara un caché completo de la página, lo que hace que la carga sea SUPER rápida, pues al detectarse la página en caché, Rails la muestra directamente.
  - No se ejecuta ningún método del controlador, ni ninguna verificación, por lo que solo se puede usar en algunos casos.
  - Para ello se utiliza la gema "**actionpack-page_caching**"

Manejo de Caché (5/5)

- **Caché de Acción:**
  - Semejante a un caché de página (genera un caché de toda la información de la página), pero con la ventaja de que ejecuta los filtros de la acción.
  - Al igual que en un caché de página, genera un único HTML para la acción y los parámetros que recibe la acción
  - Para utilizarlo hay que agregar la gema: "**actionpack-action_caching**"

Algunas librerías de utilidad



- Bullet: https://github.com/flyerhzm/bullet
- new_relic gem: https://github.com/newrelic/rpm

Optimización de servidores

- Hay diversas versiones de Ruby y diversos servidores de aplicación
- Una de las opciones más robustas es la utilización de nginx con “Passenger” el cual brinda un buen desempeño con un excelente soporte con cualquier versión de Ruby
- Hay otras opciones como Puma que se enfocan en alta concurrencia y al integrarse con Rubinius o Jruby dan un excelente desempeño

Creación de gemas

- Crear una gema es bastante simple, se requiere simplemente crear un folder ‘lib’, un archivo ‘gemspec’ con la información de la gema y unos tests que verifiquen la gema.

- http://guides.rubygems.org/make-your-own-gem/

Recursos adicionales

- ***Libro:*** *Agile Web Development with Rails 4*
- http://guides.rubygems.org/make-your-own-gem/
- https://blog.engineyard.com/2014/improving-rails-app-performance-with-database-refactoring-and-caching