Generation of vulnerabilities and threat reports for web applications

Nom Estudiant: Miquel Rius Lletí

Programa: Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions (MISTIC)

Nom Consultor: Jordi Duch Gavaldà i Agustí Solanas Gómez

Centre: Universitat Oberta de Catalunya

Data Lliurament: 06/2015

FITXA DEL TREBALL FINAL

Títol del treball:	*Generation of vulnerabilities and threat reports for web applications*
Nom de l'autor:	*Miquel Rius Lleti*
Nom del consultor:	*Jordi Duch Gavaldà i Agustí Solanas Gómez*
Data de lliurament (mm/aaaa):	*06/2015*
Àrea del Treball Final:	*[M1.728]/[M1.828] Security of web applications*
Titulació:	Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions (MISTIC)

Resum del Treball (màxim 250 paraules):

En aquest treball s'ha desenvolupat un sistema per a monitoritzar de manera simultània la seguretat de diferents màquines web mitjançant la realització de tests periòdics.

L'element principal d'aquest sistema és un panel de control des d'on es visualitza l'estat de les màquines monitoritzades. Aquest panel té les funcionalitats d'afegir i treure màquines, seleccionar els tipus d'anàlisis i la periodicitat dels mateixos.

El sistema ha estat desenvolupat per a realitzar dos tipus d'anàlisis, aquests anàlisis són una prova de concepte per a demostrar la funcionalitat del sistema. El disseny però, és totalment modular i permet afegir nous anàlisis modificant unes poques línies de codi.

Els tests implementats són descrits en l'apartat OTG-CONF-006 de la guia de testos de la OWASP (Open Web Application Security Project) i permeten determinar si el host que es monitoritza té habilitats algun des 4 mètodes que es consideren no segurs: PUT, DELETE, CONNECT i TRACE. També es comprova si el sistema és vulnerable a atacs XST mitjançant el mètode TRACE.

El codi s'ha deixat disponible a https://github.com/sefanitro/Generation-of-vulnerabilities-and-threat-reports-for-web-applications

Abstract (in English, 250 words or less):

In this paper we developed a system to monitor simultaneously web applications by running periodic tests.

The main element of this system is a control panel from which you view the status of the monitored machines. This panel has the functionality to add and remove machines, select the type of analysis and the frequency.

The system has been developed to perform two types of analysis, these tests are a proof of concept to demonstrate the functionality of the system. But the design is modular and allows new analysis by changing a few lines of code.

The implemented tests are described in Section 006-OTG-CONF of Testing Guide - OWASP (Open Web Application Security Project). In the first one, we try to request to web server which HTTP methods are allowed, we know there are 4 unsafe methods (PUT, DELETE, TRACE, and CONNECT) can potentially pose a risk for web application. The second one, we test if the system is vulnerable to attacks by XST TRACE method.

All code is on https://github.com/sefanitro/Generation-of-vulnerabilities-and-threat-reports-for-web-applications

Paraules clau (entre 4 i 8):

seguretat – aplicació web – web – vulnerabilitats – anàlisis – tests – ruby – ruby on rails

Índex

Llista de figures

1 Introducció

1.1 Context i justificació del Treball

La seguretat de qualsevol sistema, i concretament dels sistemes web, és un dels majors reptes d'avui en dia[1]. És un repte que ens obliga a destinar recursos de manera recurrent per minimitzar l'impacte de noves vulnerabilitats que van apareixent. Això és així perquè el que avui considerem segur, pot no ser-ho passat un temps i ens obliga a estar constantment actualitzats.

En un entorn on el software insegur sembla la norma, cal destacar la funció d'organitzacions com la OWASP que mitjançant treballs com la Guia de Tests OWASP determinen uns mínims estàndards en termes d'enginyeria i tecnologia que haurien de tenir les aplicacions web per tal de considerar-les segures.

Els tests de seguretat per sí sols no permeten mesurar la seguretat d'una aplicació web. Hi ha infinites maneres que un atacant pot fer que una aplicació es trenqui, i és directament impossible realitzat un test de totes elles. El que sí es pot assegurar és que si en el desenvolupament de la nostra aplicació es segueixen una sèries d'estàndards en termes d'enginyeria i tecnologia, el punt de partida la nostra aplicació està més proper a allò que es pot considerar la programació segura.

Per realitzar tests de seguretat es poden trobar moltes solucions al mercat, fins i tot algunes que permeten realitzar anàlisis molt complerts sense cap tipus de cost, com seria el cas de Nessus o Nmap. Aquestes eines però, tenen l'inconvenient que realitzen l'anàlisi sobre una o diverses màquines però no permeten la monitorització recurrent o periòdica. Es fa necessari així, el tenir un sistema que permeti utilitzar aquestes eines d'una manera intel·ligent per a monitoritzar contínuament els recursos d'una organització.

1.2 Objectius del Treball

L'objectiu d'aquest treball és desenvolupar un sistema web que permeti monitoritzar simultàniament i de manera contínua la seguretat web de diferents màquines.

Aquest sistema haurà de tenir un panel de control per gestionar les màquines a monitoritzar i permetre la modificació d'alguns paràmetres bàsics de configuració. Per a cada màquina del panel de control, s'haurà de poder veure l'historial d'anàlisis que permeti veure la evolució de la seguretat web d'aquell sistema.

Finalment, s'ha de tenir present en el disseny que tingui una interfície simplista per a que pugui ser utilitzada per un usuari 'no expert' en seguretat.

1 Eoin Keary. OWASP Testing Guide.

1.3 Enfocament i mètode seguit

Aquesta temàtica ja ha estat tractada en altres projectes final del MISTIC, sota el títol «Generation of vulnerabilities and threat reports for web applications» en podem trobar alguns exemples. Cal destacar el que es realitzava ara fa un any «Seguridad en aplicaciones web»[2] i en el que amb un punt de partida similar s'arribava a integrar l'eina Nessus dintre d'un sistema web, permetent la monitorització de màquines web de manera periòdica.

En les conclusions d'aquell treball es destaquen com a pendents dos punts: ampliar algunes funcionalitat per recollir dades interessants de l'escànner Nessus i millorar la presentació de les dades i dels informes.

Coneixent aquesta experiència prèvia, en aquest treball s'incidirà en la millora de la presentació de les dades i en la realització d'un sistema amb un disseny modular que permeti treballar amb diferents escànners de vulnerabilitats. Com una prova de concepte s'implementaran algun anàlisis bàsics que apareixen a la OWASP Testing Guide.

Pel que fa a la tecnologia ens decantarem per un sistema web, ja que serà la manera més senzilla de tenir un sistema multiplataforma. Pel que fa al llenguatge i el framework, i tenint en compte que no tinc una gran experiència en programació web, em decantaré per treballar en un sistema de codi obert com és Ruby on Rails[3].

Finalment, i tot i no ser una qüestió principal, tot el codi s'ha publicat a GitHub (https://github.com/sefanitro/Generation-of-vulnerabilities-and-threat-reports-for-web-applications). Crec que és coherent que si volem dissenyar un sistema modular i que es puguin afegir nous escànners, que també deixi que tot el codi d'aquest treball estigui disponible per a tercers.

1.4 Planificació del Treball

Aquest treball es planifica per a tot el semestre de primavera del curs 2014-15, iniciant a finals de febrer i fins al juny de 2015. El treball es defineix en 4 fases, recerca d'informació, disseny del sistema, implementació i memòria i documentació.

1. Recerca d'informació. Fase inicial del projecte on s'ha cercat informació per a la realització del treball. Aquesta informació contempla la guia de testos de la OWASP (Open Web Application Security Project), escànners de seguretat i papers sobre seguretat web, eines de desenvolupament web, Ruby on Rails.

Període: 5 de febrer al 22 de febrer.

2. Disseny del sistema. A partir de la primera fase inicial i les instruccions dels tutors, s'ha iniciat el disseny del sistema web que s'implementarà en la següent fase.

Període: 16 de febrer al 29 de febrer.

2 TFM MISTIC: «Seguridad en aplicaciones web», Isidoro de la Cierva. 06/2014.

3 Ruby on Rails és un framework de codi obert per a desenvolupament d'aplicacions web pensat sota el paradigma model-vista-controlador (MVC) que facilita l'us d'estàndars web com XML o JSON per a la transferència de dades, i HTML, CSS i Javascript per a la visualització i l'interfície d'usuari.

3. Implementació del sistema. Fase principal del projecte on a partir del disseny de la fase anterior, es desenvolupa un sistema web que permeti assolir els objectius marcats. Al ser una fase molt important i que requereix molt de temps, es divideix en diverses parts o fites parcials:

a. Instal·lació del framework i primers passos. Instal·lació del framework Ruby on Rails i familiarització amb el mateix.

Període: 22 de febrer al 5 de març

b. Panel de control bàsic amb llista de servidors i configuració bàsica del servidors.

Període: del 6 de març 29 de març

c. Historial del servidor bàsic. Llista d'informes.

Període: del 23 de març al 12 d'abril

d. Incorporació d'un anàlisis segons guia OWASP i execució del mateix sota demanda.

Període: del 13 d'abril al 26 d'abril

e. Panel de control enriquit. Incorporació de semàfors per visualitzar l'estat actual del servidor.

Període: del 27 d'abril al 29 d'abril.

f. Historial del servidor enriquit. Gràfics de l'evolució.

Període: del 30 d'abril al 4 de maig.

g. Execució d'anàlisis en segon pla (threads).

Període: del 5 maig al 23 de maig.

h. Configuració avançada del servidor: periodicitat i selecció d'anàlisis.

Període: del 18 de maig al 23 de maig.

4. Memòria i documentació.

Període: del 25 de maig al 15 de juny.

1.5 Breu descripció dels altres capítols de la memòria

Capítol 2.1. MVC. Explicació de com funciona el paradigma MVC en entorn web i específicament sota Ruby on Rails.

Capítol 2.2 Model. Aspectes a considerar en el disseny del model de dades. En aquest capítol s'explica com Rails realitza el mapping entre base de dades i també es detallen els atributs principals de les classes Server i History. Al final de l'apartat i dintre de la classe History, trobem els dos anàlisis implementats com a prova de concepte.

Capítol 2.3 Controllador. Aspectes clau en el disseny del controlador.

Capítol 2.4 Vista. En aquest capítol s'explica l'estructura de directoris de la vista i es detallen les principals que s'ha dissenyat.

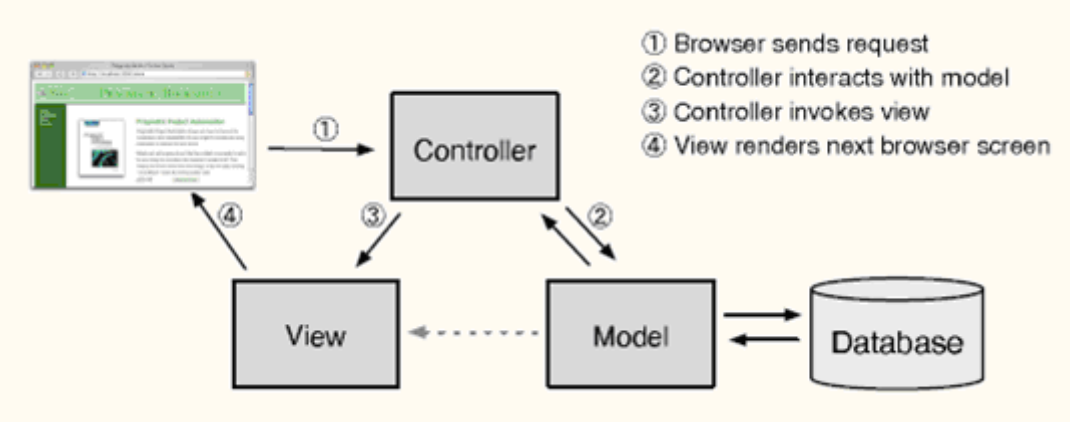
Capítol 2.5 Routing. En aquest capítol s'explica com funció l'enrutament sota Ruby on Rails i com s'ha configurat el fitxer `config/routes.rb`.

2 Resta de capítols

2.1 MVC

L'any 1979 Trygve Reenskaug va introduir una nova arquitectura per desenvolupar aplicacions interactives. En aquest disseny, les aplicacions es partien en tres components diferents: models, vistes i controladors.

El model es el responsable de mantenir l'estat de l'aplicació. A vegades aquest estat es transitori, i que només



Il·lustració 1: L'arquitectura model-vista-controlador

perdura unes poques interaccions amb l'usuari. Altres vegades es permanent i serà emmagatzemat fora de l'aplicació, sovint en una base de dades.

Un model és més que simplement unes dades, el model ha d'assegurar que les regles del negoci o l'organització s'apliquen correctament a les dades. Per exemple, si un descompte no hauria de ser aplicat a les comandes de menys de 20€, el mode és l'encarregat d'imposar aquesta restricció. Per tant el model no només emmagatzema les dades sinó que també fa la funció de garant de les dades.

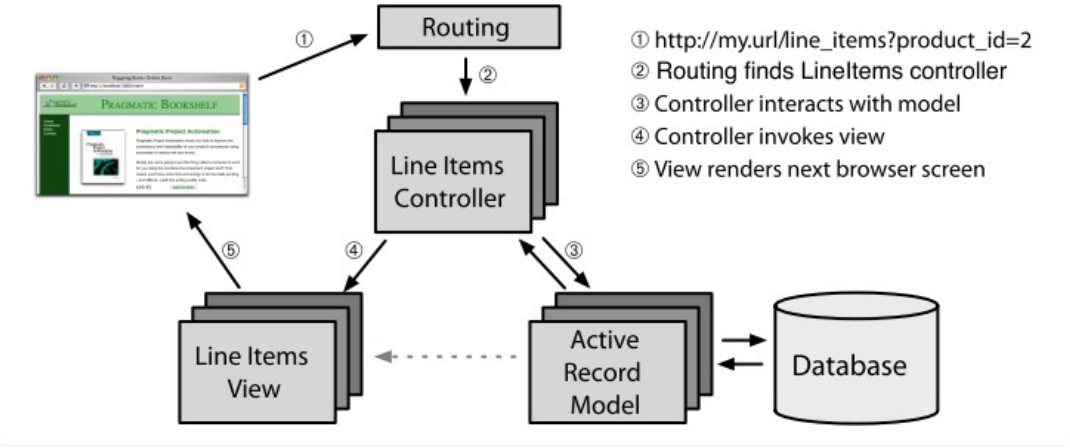
La vista és responsable de generar la interfície d'usuari, normalment basada en les dades del model. Per exemple, el panel de control és una llista dels host que volem monitortizar. Aquesta llista és accessible a través del model, però és la vista que accedeix a aquesta llista des del model i ho formata en codi HTML per a ser visualitzat per l'usuari.

El controlador orquestra l'aplicació. El controlador o controladors, rep els events de l'exterior, normalment per l'acció de l'usuari, interactiu amb el model, i mostra la vista apropiada a l'usuari.

L'arquitectura MVC va ser originalment prevista per a aplicacions convencionals GUI, on els desenvolupadors trobaven menys dificultats en la separació de problemes, que acabava derivant en un codi mes senzill d'escriure i de mantenir.

Ruby on Rails també és un framework MVC. Rails assegura una estructura per a l'aplicació, mentre es desenvolupen models, vistes i controladors en peces separades, i ho engalza tot junt mentre el programa

s'executa. Una de les joies de Rails és que aquest procés es basa en la utilització d'assumpcions intel·ligents que eviten l'us d'escriure configuracions externes per a fer-ho funcionar tot. Aquest és un exemple de la filosofia de Rails de *millor convenció que configuració.*



Il·lustració 2: Rails i MVC

En una aplicació Rails, una petició rebuda s'envia primer a l'enrutador (2.5), que decideix on s'ha d'enviar i com aquesta petició ha de ser interpretada. En ultima instancia, en aquesta fase s'identifica un mètode particular (sota Rails parlem d'acció o action) en alguna part del codi del controlador. L'acció pot agafar dades de la petició, pot interactuar amb el model i pot invocat altres accions. Finalment l'acció prepara la informació per la vista, que acabarà mostrant (render) alguna cosa a l'usuari.

2.2 Model (/app/models/)

En general, volem que la nostra aplicació web mantingui la informació en una base de dades relacional. Per exemple, en sistemes de gestió de comandes guarden ordres, gama de productes i detalls dels clients en taules de bases de dades. Inclus aplicacions que normalment utilitzen text desestructurat, com blogs o portals de noticies, habitualment utilitzen bases de dades com a suport per a emmagatzemar les dades.

Encara que pot no sembla evident amb l'ús de SQL per accedir a elles, la bases de dades relacionals estan de fet dissenyades matemàticament sota la teoria de conjunts. Tot i que això és bo des d'un punt de vista conceptual, dificulta la combinació de bases de dades relacionals amb llenguatges de programació orientats a objectes. Objectes es tot sobre data i operacions, i bases de dades es tot sobre conjunt de valors. Operacions que son senzilles d'expressar en termes relacionals son sovint complicades d'implementar en un sistema orientat a objectes. L'afirmació contraria també es certa.

La reconciliació de relacionals i orientació a objectes es gestionat per Rails d'una manera molt transparent, ho veiem en el següent apartat.

2.2.1 Emparellament objecte-relacional (Object-Relational Mapping o ORM)

Les llibreries ORM emparellen les taules de la base de dades amb classes. Per exemple, si creem una taula per emmagatzemar hosts que li diem Servers, el nostre programa tindra una clase que es dira Server. Les files de la taula corresponen a objectes de la classe, un host que volem monitoritzar es representa com un objecte de la classe Server. Dintre d'aquest objecte, els atributs s'utilitzen per llegir i escriure en les columnes.

Aixi, una capa ORM emparella taules amb classes, files amb objectes, i columnes amb atributs d'aquests objectes. Els metodes de classe s'utilitzen per realitzar operacions a nivell de taula, i metodes d'instancia realitzen operacions en files individuals.

2.2.2 Active Record

Active Record es la capa ORM que proporciona Rails. Segueix fidelment l'estàndard del model ORM: taules amb classes, files amb objectes, i columnes amb atributs d'objectes. Es diferencia de moltes altres llibreries ORM en la manera en que és configurat. Basant-se en la convenció i en assumpcions raonables, Active Record minimitza la quantitat de configuració que s'ha d'implementar. De nou veiem la premissa *millor convenció que configuració.*

Active Record ens allibera de la molèstia de treballar amb la base de dades per deixar temps lliure per treballar en la lògica de l'aplicació.

En aquest procte hem definit dos classes Active Record, la classe `Server` i la classe `History`. La primera per guardar els hosts que volem monitortizar i la segona per guardar els tests que realitzen sobre cada host. Aquestes dos classes van emparellades en dos taules a la base de dades que es relacionen amb una relació de un a molts.

2.2.3 classe Server (app/models/server.rb)

Aquesta classe hereda d'Active Record i conté la informació de configuració dels hosts que es volen monitoritzar. Cada instància de la classe Server és un host del panel de configuració. Per emmagatzemar aquesta informació necessitem tres camps de la base de dades i per tant tres atributs de la classe Server, aquest són:

- host del tipus string, per definir la URL a monitoritzar.
- period del tipus enter, per definir el període de monitorització.
- test_selection del tipus enter, per seleccionar els anàlisis que volem activar per a un host en concret.

1. period

El període de monitorització s'emmagatzema com un enter, la seva selecció però ha de ser més entendible per a l'usuari i es fa necessari així la definició d'una variable Hash. En el disseny s'ha definit 6 periodes diferents des d'1 hora fins a 1 dia i aquest s'han assignat a un enter que serà emmagatzemat a l'atribut `period`.

```
    PERIOD_OPTIONS = {'1h' => 1,
                      '2h' => 2,
                      '3h' => 3,
                      '6h' => 4,
                      '12h' => 5,
                      '24h' => 6
                      }
```

app/model/Server.rb

La definició d'aquest hash ens permet que la implementació de la llista de selecció de la vista `_form` (utilitzada tant per crear un nou host com per configurar un host existent) sigui directa amb la crida `Server::PERIOD_OPTIONS`.

```
{...}
<div class="field">
  Canvia el període de monitorització:
  <%= f.select :period, Server::PERIOD_OPTIONS %>
</div>
{...}
```

app/views/servers/_form.html.erb

2. *test_selection*

Una funcionalitat que s'ha volgut aconseguir és que l'usuari tingui la llibertat d'activar i descactivar aquells anàlisis que li interessi. Per aconseguir aquesta funcionalitat la solució més senzilla és la de crear una variable cert/fals per a cada anàlisis implementat. Tot i ser una solució senzilla, el fet d'afegir nous anàlisis implicarà que per força s'haurà de modificar l'esquema de la base de dades per anar a afegint variables per a cada anàlisis. Aquesta solució tot i ser correcta, crec que és millorable amb la utilització d'una unica variable enter i interpretar-la bit a bit en forma de màscara. Per a implementar la màscara s'ha fet ús de `bitmask`, una llibreria (en termes de Ruby, gem o gemma) que facilita el treball sobre una variable de tipus enter per a ser interpretada en forma de bit.

```
bitmask :test_selection, as: [:Supported_http_methods, :XST_vulnerability, :Other]
```

app/models/Server.rb

Amb aquesta una única línia s'ha definit tres anàlisis diferents. De fet només dos d'ells estan implementats, mentre el tercer s'ha deixat de mostra sota el nom `Other`. Per a la creació d'un nou anàlisi s'haura d'afegir un element a l'array `test_selection` i s'haurà de crear un mètode d'instància a la classe History que executarà el codi d'aquest nou anàlisis.

Amb aquest disseny, afegir nous anàlisis no requereix de la moficiació de les vistes, ja que aquestes estan preparades per llegir l'atribut `test_selection` i generar un `check_box_tag` per a cadascun dels seus valors.

```
{...}
 <div class="field">
   Selecciona els anàlisis:<br>
   <% Server.values_for_test_selection.each do |test| %>
     <%= check_box_tag 'server[test_selection][]', test,
       @server.test_selection.include?(test), id: test %>
     <%= label_tag test, test.to_s %><br>
   <% end %>
 </div>
{...}
```

app/views/servers/_form.html.erb

Ens faltaria per veure com es realitza l'assignació entre un element de `test_selection` i l'execució de l'anàlisis. Just unes línies mes avall veurem com aquesta assignació es realitza dintre del mètode `run_server_monitoring`.

3. *Mètode run_server_monitoring*

Aquest és un mètode d'instància i és el principal mètode de la classe Server. Aquest mètode no necessita de la interacció de l'usuari per a executar-se. La seva execució es realitza en l'arrencada del servidor web i s'executa

amb un fil (thread) independent per a cada instància de la classe Server, per tant, per a cada host del panel de control.

Un fet a tenir en compte en l'arrencada de tots els fils, és que aquesta concurrència ens pot portar a executar tots els anàlisis concurrentment. Tot i que el resultat de caddascun dels anàlisis seria correcte, el fet d'iniciar moltes conexions en un interval reduït pot donar peu a problemes (desconnexions, retards, pèrdues de paquets, ...) que es poden minimtzar si espaiem els anàlisis amb el temps. Cal tenir en compte també que aquest problema es veuria agreujat a mesura que afegim nous hosts per a la monitorització, a més nombre de hosts més nombre de connexions durant el mateix interval de temps.

Per aquest motiu en l'inici del mètode run_server_monitoring realitzem una espera aleatoria de 30 minuts. D'aquesta manera evitem alguns problemes que ens podrien apareixer amb la concurrència.

```
class Server < ActiveRecord::Base

{...}

def run_server_monitoring
  puts "***** Iniciem...on #{host}: #{test_selection}\n"
  sleep(rand(1800))

{...}
```

app/model/servers/Server.rb

Després d'aquesta espera aleatòria, el mètode entra en un bucle infinit on en cada iteració crearà una nova instància de History i executarà els tests que l'usuari tingui actius per a aquest host.

```
def run_server_monitoring

{...}

loop do
  puts "******  New server test on #{host}..."
  test = History.new(date: Time.now, server_id: self.id)
  test.discover_supported_http_methods if test_selection?(:Supported_http_methods)
  test.test_XST_potential if test_selection?(:XST_vulnerability)
  #test.NEW_METHOD_TEST if test_selection?(:Other)
  test.save
  sleep periode
end
```

app/model/servers/Server.rb

És en aquest punt on es realitza l'assignació entre l'element de l'array test_selection i el mètode de la classe History. Secuencialment es realitza la comprovació de tots els valors de la màscara test_selection i en el cas que estigui actiu, s'executa un mètode per a cadascun d'ells. Aquí podem veure els dos mètodes que s'ha implementat: discover_supported_http_methods i test_XST_potential.

Amb aquest disseny es força senzill implementar nous anàlisis, tres són els passos a seguir:

1. Afegir un element a l'array test_selection.
2. Afegir una línia dintre del mètode run_server_monitoring que enllaci el nou element i el mètode a executar.
3. Escriure el codi de l'anàlisi com un mètode de la classe History.

4. Relació classes Server i History

Com ja s'ha comentat a l'apartat anterior, `Server` i `History` tenen una relació de un a molts a la base de dades, això es determina mitjançant la instrucció `has_many` o `belongs_to` a l'inici de cadascuna de les classes.

```
class Server < ActiveRecord::Base
  has_many :history
{...}
```

app/model/server.rb

```
class History < ActiveRecord::Base
  belongs_to :server
{...}
```

app/model/history.rb

5. Threads

Dos mètodes implementen la concurrència, el mètode de classe `self.start_monitoring` i el mètode d'instancia `run_server_monitoring`.

El mètode de classe `start_monitoring` s'executa tant bon punt iniciem el servidor, la seva única funció es executar per a cada instancia de servidor, es a dir cada host del panel, un thread que executi el mètode d'instància `run_server_monitoring`.

```
def self.start_monitoring
    Server.all.each do |server|
       Thread.new { server.run_server_monitoring }
    end
 end
```

app/models/servers/Server.rb

2.2.4 classe History (app/models/history.rb)

Aquesta classe també hereda d'Active Record i conté el resultat d'un bloc d'anàlisis. S'entén per bloc d'anàlisis, tots aquells tests que es realitzen durant un període de monitorització. El resultat d'aquest bloc es guarda en una instància de la classe history el que seria una fila de la taula Histories de la base de dades.

Els atributs necessaris per recollir la informació de l'anàlisis són els següents:

- date del tipus datetime, per guardar l'instant d'inici de l'anàlisis.
- output del tipus string, camp d'informe on es guarda la sortida de l'anàlisis. Aquest camp ens permet identificar el motiu dels errors i warnings.
- num_tests del tipus enter, per guardar el nombre d'anàlisis realitzat en un bloc.
- num_errors del tipus enter, per guardar el nombre d'errors trobats.
- num_warnings del tipus enter, per guardar el nombre de warnings trobats.

En aquesta classe s'ha implementat dos mètodes, que es corresponen als dos anàlisis que s'ha implementat.

1. discover_supported_http_methods (OTG-CONFIG-006)

HTTP permet un nombre de mètodes que es poden utilitzar per executar accions sobre un servidor web. Alguns d'aquest mètodes estan dissenyats per ajudar els desenvolupadors a desenvolupar i testar applicacions HTTP. Aquests mètodes poden ser utilitats de manera maliciosa si el servidor web està mal configurat.

La versió HTTP 1.1 defineix els següents vuit mètodes: HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS i CONNECT.

Alguns d'aquests són potencialment perilloso per a una aplicació web, ja que habiliten que un atacant pugui modificar els fitxers guardats al servidor web i, en alguns casos, robar credencials d'usuaris legítims. Concretament, els mètodes que haurien d'estar deshabilitats són:

- PUT: Aquest mètode permet a un client pujar nous fitxers al servidor web. Un atacant pot explotar-ho per pujar documents maliciosos (un fitxer asp que executi comandes mitjançant cmd.exe), o que simplement utilitzi el servidor web de la víctima com a repositori de fitxers.
- DELETE: Aquest mètode permet que un client elimini un fitxer del servidor web. Un atacant pot explotar-ho d'una manera molt simple i directa per fer un *deface* d'una web o per montar un atact de DoS[4].
- CONNECT: Aquest mètode pot permetre a un client utilitzar el servidor web com un proxy.
- TRACE: Aquest mètode simplement realitza un eco de tornada al client de qualsevol cadena que se li envii al servidor, i té la utilitat principal com a mètode de *debug*. Aquest mètode que semblava originalment inofensiu, pot ser utilitzat per montar un conegut atac de Cross Site Tracing[5].

Cal tenir present, però, que algunes aplicacions web necessiten alguns d'aquest mètodes, per exemple PUT o DELETE, per a tenir un correcte funcionament (aquest mateix projecte n'és un exemple). En casos com aquest és important comprovar que el seu ús el limita a usuaris legítims i en condicions de seguretat.

En la implementació d'aquest test s'ha tingut en compte això i s'ha executat 4 comprovacions, una per cada mètode HTTP. Per l'especificitat de de PUT i DELETE, es considera que són warnings i tant CONNECT com TRACE es consideren errors i haurien d'estar inhabilitats.

Per comprovar quins són els mètodes HTTP permesos pel host es fa ús de l'eina netcat (nc) i el metode HTTP OPTIONS. Sabent que la línia on apareixen els mètodes permesos s'inicia amb la cadena Allow, filtrem la resposta de nc amb l'eina grep i obtenim tots els mètodes que permet el servidor. Ara només ens queda cercar cadascun dels 4 mètodes crítics i ja tenim el test implementat.

```
class History < ActiveRecord::Base
 {...}

  def discover_supported_http_methods
    {...}
    strcmd = "printf \"OPTIONS / HTTP/1.1\r\nHost: #{myHost}\r\n\r\n\" | nc #{myHost}
80 | grep \"Allow:\""
    stdin, stdout, stderr = Open3.popen3(strcmd)
    {...}
```

app/models/histories/History.rb

2. *test_XST_potential (OTG-CONFIG-006)*

El mètode TRACE, aparentment inofensiu, pot desencadenar en alguns escenaris amb el robatori de credencials d'usuaris legítims. La tècnica de l'atac va ser descoberta per Jeremiah Grossman el 2003, en un intent de sobrepassar l'etiqueta HTTPOnly que Microsoft va introduir a l'Internet Explorer 6 SP1 per protegir les galetes de ser accessibles per JavaScript.

Si el cos de la resposta és exactament una còpia de la petició original, signigica que el host permet aquest mètode. Si s'intrueix al navegador per a que realitzi una petició TRACE a un servidor web, i aquest navegador té la galleta d'aquell domini, la galleta serà automàticament inclosa a la capçalera de la petició, i serà per tant retornada en l'eco de la resposta del servidor. En aquest put, la cadena de la galleta sera accessible per

4 Atac de denegació de servei, de l'anglès Denial of Service.

5 Jeremiah Grossman: "Cross Site Tracing (XST)" - http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf

JavaScript i podrà ser enviada finalment a terceres persones inclòs en el cas que la galleta estigui etiquetada com a httpOnly.

La comprovació d'aquesta vulnerabilitat és molt senzilla i segueix la mateixa estructura que l'anterior. En aquest cas però el que busquem és que aparegui la cadena que enviem al servidor (després del printf) i per això és la mateixa cadena que especifiquem a la comanda grep. S'ha de tenir present que per saber si es vulnerables s'haurà de comprovar que la cadena apareix dos vegades, la que enviem nosaltres i la de tornada.

```
strcmd = "printf \"TRACE / HTTP/1.1\r\nHost: #{myHost}\r\n\r\n\" | nc #{myHost} 80 |
grep \"TRACE / HTTP/1.1\r\nHost: #{myHost}\""
```

app/models/histories/History.rb

2.3 Controlador (/app/controllers/)

«Fat model skinny controller». Aquesta premissa bàsica de l'arquitectura MVC ha estat el principal criteri per al disseny dels dos controladors, `histories_controller` i `servers_controller`.

Dintre del patró MVC, el controlador és l'encarregat de recollir les peticions dels usuari i enviar invocacions al model. El controlador també es l'encarregat del cami de tornada i recollir la resposta del model i presentar-la a traves de la vista. Normalment aquest ultim pas es realitza amb la funcio render. Tot i que la invocacio a la funcio render pot ser explicita, ruby assumeix una vista per defecte per a cada mètode.

El codi que contenen aquests controladors és el mínim per a permetre a les vistes accedir a la base de dades deixant la lògica del sistema en el model.

2.4 Vista (/app/views/)

Quan creem una vista, estem creant una plantilla: alguna cosa sobre la que es generara el resultat final. Les plantilles les podem trobar al directori `app/views` de l'aplicació. Dintre d'aquest directori, per convencio es troben subcarpetes per a les vistes de cada controlador. Aixi tenim que les plantilles separades a `app/views/servers` i `app/views/histories`. L'estructura del directori views la podem veure a continuació:

```
views
    ├── servers
    │   ├── edit.html.erb
    │   ├── _form.html.erb
    │   ├── index.html.erb
    │   └── show.html.erb
    ├── layouts
    │   └── application.html.erb
    └── histories
        └── index.html.erb
```

Il·lustració 3: Esquema directori de vistes

Rails permet fer renders de pagines de manera niuada dintre del render d'altres pagines. La manera mes habitual es la de definir un frame estàndard per a la nostra aplicació (títol, capçalera i barra lateral) i que sigui compartit per totes les plantilles.

Quan Rails respon a una petició de fer render d'una plantilla des d'un controlador, de fer s'esta fent render de dos plantilles. Òbviament, es fa render de la que se li ha sol·licitat però Rails també intenta de fer render de la plantilla layout. Si la troba, insereix el codi HTML de resposta dintre del codi HTML produït pel layout.

2.4.1 layout/application.html.erb

En el head d'aquesta plantilla s'ha afegit parametres bàsics de configuració que es vol que siguin accessibles des de qualsevol vista. Aquesta solució té un doble objectiu, per una banda que el disseny de la resta de plantilles sigui més net i ens facilita la feina de manteniment i actualització de l'aplicació mitjançant la reutilització de codi. Així realitzant canvis en una única plantilla s'apliquen al conjunt de plantilles de la nostra aplicació.

```
<head>
  <title>Generation of vulnerabilities and threat reports for web
applications</title>
  <%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track'
=> true %>
  <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
  <%= javascript_include_tag "//www.google.com/jsapi", "chartkick" %>
  <%= csrf_meta_tags %>
</head>
```

app/views/layouts/application.html.erb

El cos d'aquesta plantilla és molt bàsic i consta de 3 elements: un títol o banner, una barra lateral i el frame principal. El títol permet donar una visió d'uniformitat al conjunt de l'aplicació, mostrant sempre el mateix contingut, sigui quina sigui la plantilla visualitzada. La barra laterial facilita la navegació dintre de la mateixa aplicació, així podem accedir al panel de control o home (`/`) i a la llista dels últims informes realitzats (`/histories`). El títol i la barra lateral són els elements en gris que es poden veure a la Il·lustació 4.

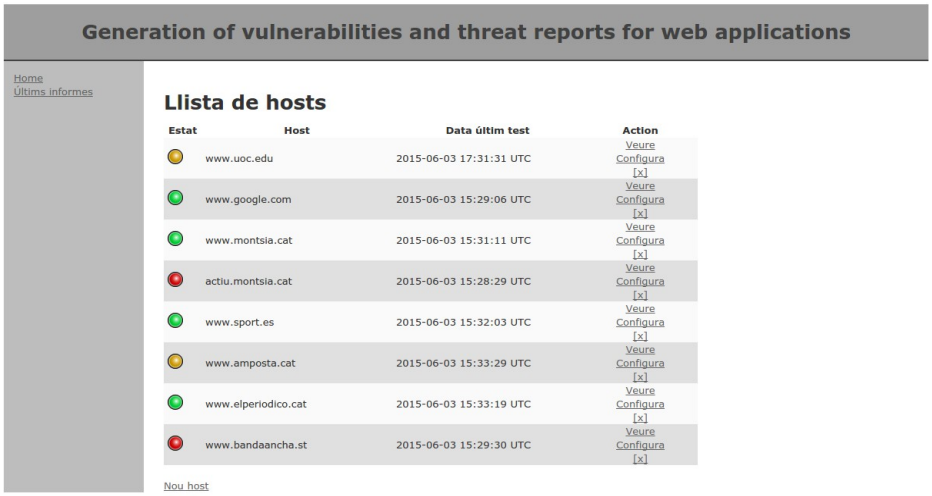
Finalment la màgia del doble render es veu amb l'última comanda, `yield`, on s'indica a Rails on pot carregar la plantilla que s'ha sol·licitat.

```
<body>
  <div id="banner">
    <h1> Generation of vulnerabilities and threat reports for web applications
</h1>
  </div>
  <div id="columns">
    <div id="side">
      <a href="/">Home</a><br />
      <a href="/histories">Últims informes</a><br />
    </div>
    <div id="main">
      <%= yield %>
    </div>
  </div>
</body>
```

app/views/layouts/application.html.erb

2.4.2 servers/index.html.erb

Aquesta plantilla és el panel de control i nucli de l'aplicació. És la vista arrel (/ o home) i ens permet visualitzar l'estat de la monitorització. Des d'aquesta vista podem afegir nous servidors per a monitoritzar (Nou host), configurar els paràmetres d'un host (configura), visualitzar en detall la monitorització d'un host (veure) i també eliminar un host de la llista ([x]).



Il·lustració 4: Vista Servers#index

Amb l'objectiu que sigui fàcil d'interpretar s'ha dissenyat aquesta vista amb la mínima informació necessària. Així només tenim 4 columnes: el semàfor d'estat, el host, la data de l'últim anàlisi i la columna d'accions. Amb una mirada al panel de control podem identificar quins hosts requereixen de la nostra atenció.

```
  <thead>
    <tr>
      <th class="row-status">Estat</th>
      <th class="row-host">Host</th>
      <th class="row-date">Data últim test</th>
      <th class="row-action">Action</th>
      <th rowspan="3"></th>
    </tr>
  </thead>

<tbody>
   <% @servers.each do |server| %>
      <tr class="<%= cycle('list_line_odd', 'list_line_even') %>">
        <td><%= image_tag get_image_status(server.id), height: "20" %></td>
        <td><%= server.host %></td>
        <td><%= get_last_history server.id %></td>
        <td class="row-action">
          <%= link_to 'Veure', server %><br />
          <%= link_to 'Configura', edit_server_path(server) %><br />
          <%= link_to '[x]', server, method: :delete, data: { confirm: 'Si
elimines el Host, eliminaràs també l\'historial d\'informes d\'aquest Host.
Estàs segur?' } %></td>
      </tr>
    <% end %>
  </tbody>
```

app/views/servers/index.html.erb

En el semàfor s'ha definit 4 estats:

- Verd. En l'últim anàlisis realitzat no hi ha ni errors ni warnings. El host es considera segur.
- Ambar. En l'últim anàlisis hi ha com a mínim un warning però no hi ha errors. El host té algun element millorable i s'ha d'estar alerta.
- Roig. En l'últim anàlisis s'ha detectat almenys un error. Aquesta categoria no s'hauria de considerar acceptable i s'haurien d'aplicar mesures correctores.
- Gris. Aquest quart estat permet identificar hosts que no es troben en cap de les anteriors categories, per exemple un host que no té configurat cap anàlisis o un host que s'hagi aturat la monitorització.

La columna data ens dona informació de quan s'ha realitzat l'últim anàlisis i per tant quina vigència té el semàfor d'estat. També pot ser un indicar per detectar fallades del sistema, si veiem que ha passat molt temps de l'últim anàlisis podem deduir que l'aplicació no està funcionant correctament.

L'última columna està reservada a les accions que es poden fer sobre el host: visualitzar, ens envia a la URL `/servers/:id` (plantilla `servers/show`) i ens permet veure el detall de la monitorització d'un host; configura, ens permet canviar els paràmetres d'anàlisis enviant-nos a la URL `/servers/:id/edit` (plantilla `servers/edit.html`); i elimina ens permet treure un host del panel de control després de demanar-nos la confirmació mitjançant un pop-up.

2.4.3 servers/show.html.erb

Aquesta plantilla és la que ens permet veure el detall de la monitorització d'un host. Amb un gràfic de barres, cada barra representa el conjunt de tests realitzats durant el període de monitorització. Els resultats de cada barra es troben apilats pel resultat del test on ok és verd, ambar és warning i roig és error. D'aquesta manera amb una observació superficial podem veure si han aparegut nous errors i quan han aparegut.

```
<%= column_chart [
    {name: "#OK", data: (@historygraph.map{|f| [f.date, f.num_tests -
f.num_errors - f.num_warnings]})},
    {name: "#Errors", data: (@historygraph.map{|f| [f.date, f.num_errors]})},
    {name: "#Warnings", data: (@historygraph.map{|f| [f.date,
f.num_warnings]})}
], colors: ["green", "red","orange"], height: "400px", stacked: true, library:
{"title":"Últims tests", "width":"1000px"} %>
```

app/views/servers/show.html.erb

En cas de voler tenir més informació sobre un error o un warning, es mostra també una taula amb la informació dels últims anàlisis. En aquesta taula tenim accés la columna output (atribut output de la base de dades history) que és l'informe de sortida dels anàlisis realitzats. Amb aquesta informació podríem identificar quin ha estat l'error que s'ha detectat i què s'ha de corregir.

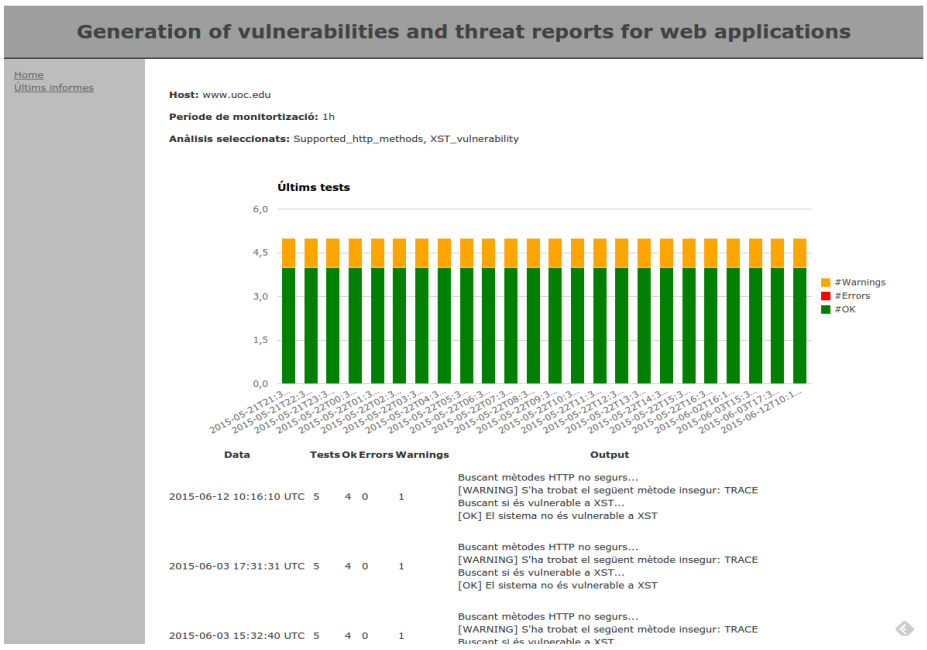


Ilustración 5: Vista Servers#show

El codi de la taula és directe, ja que quasi tots els paràmetres surten dels atributs de la classe History.

```
<tbody>
    <% @history.each do |history| %>
      <tr>
        <td><%= history.date %></td>
        <td><%= history.num_tests %></td>
        <td><%= history.num_tests - history.num_errors - history.num_warnings %></td>
        <td><%= history.num_errors%></td>
        <td><%= history.num_warnings%></td>
        <td><%= simple_format(history.output) %></td>
      </tr>
    <% end %>
  </tbody>
```

app/views/servers/show.html.erb

Cal tenir en compte que les dades de la taula (@history) i del gràfic (@historygraph) són variables diferents però són les mateixes dades. La diferència entre una variable i l'altra és l'ordre en que estan les dades. D'aquesta manera en el gràfic la dada més recent es mostra a la dreta i en la taula la dada més recent es mostra a dalt.

```
  def show
    {...}
    @history = History.where(server_id: params[:id]).order('date DESC').limit(24)
    @historygraph = @history.reverse
  end
```

app/controllers/servers_controller.rb

2.4.4 servers/edit.html.erb

Aquesta plantilla fa ús de plantilla auxiliar `_form.html.erb`. L'ús del render aniuat és molt útil en aquest cas perquè ens permet la reutilització del formulari per a diferents vistes, el formulari és el mateix per actualitzar paràmetres d'un host que per a crear un nou host. Aquest aprofitament de codi ens permet agilitzar les tasques de manteniment i actualització de l'aplicació.

```
<h1>Editing Server</h1>

<%= render 'form' %>

<%= link_to 'Show', @server %> |
<%= link_to 'Back', servers_path %>
```

app/views/servers/edit.html.erb

La plantilla _form ens permet configurar un host mitjançant tres paràmetres: el host o URL, el període i els tests a realitzar (Il·lustació 6).

Il·lustració 6: Vista Server#edit

El període va enllaçat amb l'atribut `period` de la classe `Server` i s'ha implementat en forma en forma de llista on és permet seleccionar un únic element. Els elements d'aquesta llista són 1h, 2h, 3h, 6h, 12h i 24h.

```
<div class="field">
  Canvia el periode de monitorització:
  <%= f.select :period, Server::PERIOD_OPTIONS %>
</div>
```

app/views/servers/_form.html.erb

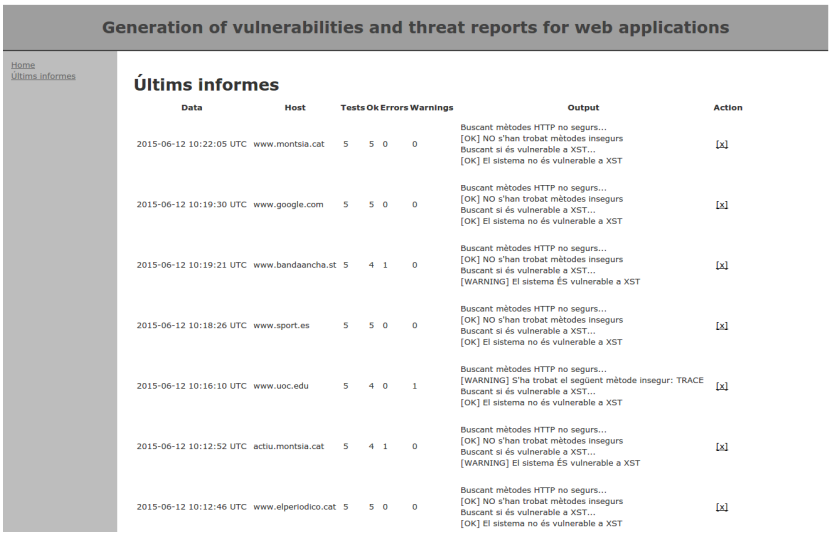
El codi de l´últim paràmetre és una mica més complexe però la complexitat és una conseqüència de treballar en màscares enlloc de fer-ho sobre variables cert/fals. El que ens permeten les màscares és poder afegir nous anàlisis sense modificar aquesta vista ni l'esquema de la base de dades. Afegir un nou análisis es realitzar modificant el codi de la classe `Server`.

```
<div class="field">
  Selecciona els anàlisis:<br>
  <% Server.values_for_test_selection.each do |test| %>
    <%= check_box_tag 'server[test_selection][]', test,
      @server.test_selection.include?(test), id: test %>
    <%= label_tag test, test.to_s %><br>
  <% end %>
</div>
```

app/views/servers/_form.html.erb

2.4.5 histories/index.html.erb

Aquesta plantilla és una plantilla auxiliar i el seu ús es principalment per a visualitzar l'estat de l'aplicació i que està funcionant correctament. El codi és molt senzill i només consisteix en una taula amb el llistat dels últims anàlisis realitzats.



Il·lustració 7: Vista Histories#index

Les columnes d'aquesta taula són directament atributs del model `History` i per tant columnes de la taula `Histories`. Així per obtenir `data`, `tests`, `errors`, `warnings` i `output` només s'ha de cridar a

`history.date`, `history.num_tests`, `history.num_errors`, `history.num_warings` i `history.output`[6] respectivament.

El nombre de tests correctes no s'emmagatzema enlloc, però es pot obtenir directament restant del nombre de tests, el nombre de warnings i errors.

Finalment per saber el host s'ha de fer ús d'una funció auxiliar que ens consulti a quin host (Server) es correspon l'analisi (History).

```
<table>
{...}
  <tbody>
    <% @histories.each do |history| %>
      <tr>
        <td><%= history.date %></td>
        <td><%= get_server_from_id history.server_id %></td>
        <td><%= history.num_tests %></td>
        <td><%= history.num_tests - history.num_errors - history.num_warnings %></td>
        <td><%= history.num_errors %></td>
        <td><%= history.num_warnings %></td>
        <td><%= simple_format(history.output) %></td>
        <td><%= link_to '[x]', history, method: :delete, data: { confirm: 'Are you
sure?' } %></td>
      </tr>
    <% end %>
  </tbody>
</table>
```

app/views/histories/index.html.erb

2.5 Routing

Quan una petició HTTP arriba des del navegador de l'usuari, aquesta necessita saber quin acció del controlador, o dit d'una altra manera, quin mètode s'hauria d'executar. L'enrutador, es bàsicament un emparellador.

L'enrutador, mira el verb HTTP (GET, POST, PUT, DELETE) i la URL que ens està sol·licitant l'usuari i l'emparella amb l'acció concreta que el controlador ha d'executar. És una funció molt senzilla però essencial. Si no és capaç de trobar la ruta de la petició, l'aplicació llençarà un error.

Per configurar les routes s'ha de modificar el fitxer localitzat a `config/routes.rb` i els paràmetres escollits són els següents:

```
1 resources :servers
```

Amb aquesta instrucció es generen les següents set rutes que s'emparellen amb el controlador `servers`:

Verb HTTP	Path	Controller#Action	Utilitzat per
GET	/servers	servers#index	Mostrar el panell de hosts
GET	/servers/new	servers#new	Accedir al form HTML per crear un nou host
POST	/servers	servers#create	Afegir un nou host a la llista
GET	/servers/:id	servers#show	Veure el host (i l'historial)
GET	/servers/:id/edit	servers#edit	Accedir al form HTML per editar el host

6 Per una millor visualització de l'atribut `output` es fa ús de la funció `simple_format`. Aquesta funció evita que `output` es vegi com una única línia i sigui més fàcil d'interpretar.

PATCH/PUT	/servers/:id	servers#update	Actualitzar la configuracio del host
DELETE	/servers/:id	servers#destroy	Eliminar un servidor de la llista de hosts

```
2 resources :histories, :only => [:index, :show, :destroy]
```

Amb aquesta instrucció habilitem només 3 rutes, index, show i destroy, de les set rutes per defecte. Es considera que la resta de rutes no han de ser accessible per l'usuari i per tant no han d'estar actives. Així les rutes que s'emparellen amb el controlador `histories` són:

Verb HTTP	**Path**	**Controller#Action**	**Utilitzat per**
GET	/histories	histories#index	Mostra els últims anàlisis.
GET	/histories/:id	histories#show	Veure el resultat d'un anàlisis.
DELETE	/histories/:id	histories#destroy	Eliminar un anàlisis.

```
3 root 'servers#index'
```

Finalment amb aquesta instrucció fixem l'arrel de l'aplicació web a `servers#index` i que acabarà fent un render de la vista `/app/views/servers/index.html.erb`.

3 3. Conclusions

Arribats a aquest punt podem afirmar que els objectius del treball s'han assolit i el sistema per monitoritzar s'ha implementat seguint les premises. Assolir els objectius però, no sempre es pot considerar un éxit. Crec que un éxit seria que aquest treball fós utilitzat per alguna organització, empresa o fins i tot particular i a dia d'avui crec que serà força difícil. Tot i així, un éxit també podria ser que aquest codi sigui modificat per algú per afegir moduls amb funcionalitats més avançades, com podrien ser la implementació d'algun anàlisis de vulnerabilitats extern.

Amb la implementació actual s'ha creat l'esquelet de l'aplicació per poder visualitzar l'estat dels servidors i realitzar configuracions bàsiques. En el disseny s'ha tingut especial cura en que aquest sigui modular, com per exemple la màscara per a la selecció dels anàlisis, de manera que assolir l'éxit per la segona via crec que és possible. De fet el codi està disponible a GitHub tant per al seu ús com per a la seva modificació.

La planificació en un projecte d'aquest tipus sempre és difícil de seguir, en aquest cas s'ha hagut de fer alguns ajusts i hi ha alguna cosa que no s'ha pogut implementar per falta de temps. El principal punt que s'ha quedat pendent és la implementació d'alguna eina d'anàlisis més complexa, tipus NMAP o WHATWEB, crec que amb això el projecte hagués guanyat utilitat.

Una altra funcionalitat pendent i que no s'ha resolt del tot correctament és la gestió dels threads. En el disseny actual aquests s'inicien quan arranca l'aplicació però una solució més correcta passaria per tenir un major control sobre els threads i poder pausar i engegar des del panel de control.

4 5. Bibliografia

[1] Sam Ruby, Dave Thomas i David Heinemeier, “Agile web development with Rails”, 4a edició The Pragmatic Bookshelf, 2011.

[2] Obie Fernandez, “The Rails 3 way”, Addison Wesley, 2011.

[3] https://www.owasp.org/index.php/Main_Page, Març 2015.

[4] OWASP Testing Guide v4, Matteo Meucci and Andrew Muller, 2014.

[5] Jeremiah Grossman: “Cross Site Tracing (XST)” - http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf