UNIVERSITAT OBERTA DE CATALUNYA

Master interuniversitario de seguridad de las tecnologías de la información y de las comunicaciones

Trabajo de Fin de Máster

especialidad Seguridad en servicios y aplicaciones

Generación de reportes de vulnerabilidades y amenazas para aplicaciones web

Autor: Eva González de Canales Glez.

Directores: Jordi Duch

Agusti Solanas

A Coruña, 14 de enero de 2014

Resumen

Este proyecto consiste en el diseño e implementación de una aplicación web que proporcione diversas funcionalidades relacionadas con el análisis de vulnerabilidades web.

La aplicación se centra en la monitorización de los *hosts* y proporciona herramientas para la detección y solución de las vulnerabilidades existentes en las aplicaciones web desplegadas en dichos *hosts*.

Para el desarrollo del proyecto se ha utilizado la plataforma Java Enterprise Edition (Java EE con el framework *open source* Spring MVC y las funcionalidades propias de análisis se delegan en la herramienta OWASP Zed Attack Proxy.

This project involves the design and implementation of a web application providing various functionalities related to the analysis of web vulnerabilities.

The application focuses on the monitoring of the *hosts* and provides tools for the detection and resolution of existing vulnerabilities in web applications deployed in such *hosts*.

The project has been developed under the Java Enterprise Edition (Java EE) platform with the *open source* framework Spring MVC and the analysis functions are delegated to the OWASP Zed Attack Proxy tool.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación y objetivos

Las vulnerabilidades de un sitio web exponen una serie de amenazas que permiten a un atacante comprometer la seguridad de la misma e incluso de todo el sistema. Para evitar esto, los desarrolladores deben identificar y eliminar en la media de lo posible dichas vulnerabilidades.

El objetivo de este proyecto es proporcionar una herramienta sencilla e intuitiva que identifique las vulnerabilidades existentes en un sitio web y que genere informes de ayuda para que el usuario las solucione. La herramienta en sí no implementa la lógica de detección de las vulnerabilidades ya que esto se delega en una herramienta externa de análisis y exploración.

1.2. Metodología

Para la realización de este proyecto, por ser un período de tiempo corto, se ha adoptado el modelo de desarrollo de *software* en cascada [1].

El modelo consta de cuatro fases diferenciadas:

1. Fase de análisis: en esta fase se describe completamente el sistema mediante un análisis de requisitos. Se elaboran los escenarios y casos de uso del sistema.

2. Fase de diseño: en esta fase se identifican los objetos del dominio. Se define la arquitectura y se realiza la elección de las tecnologías.

3. Fase de implementación: en esta fase se convierte el diseño al lenguaje de programación escogido especificando las estructuras de datos e interfaces.

4. Fase de pruebas: en esta fase se realizan las pruebas de unidad e integración.

Cada fase se realiza en un período de tiempo representado en el diagrama de Gantt de la figura 1.1:

- En la primera fase se ha realizado un estudio de las principales vulnerabilidades web, en concreto, vulnerabilidades de validación de datos [**?**]. También se ha realizado un análisis de las principales herramientas de detección y análisis de vulnerabilidades. Además se ha realizado un análisis de requisitos para determinar las funcionalidades a implementar.

- En la segunda y tercera fase se ha realizado el diseño e implementación de las funcionalidades del sistema. Además se ha escrito la documentación del proyecto (incluido este documento).

- En la cuarta fase se ha realizado una batería de pruebas.

Además, durante el desarrollo se realizaron consultas periódicas con el director para la resolución de problemas y dudas.

1.3. Estructura de la memoria

El segundo y siguiente capítulo de la memoria describe y analiza las herramientas de detección de vulnerabilidades más utilizadas actualmente.

En el tercer capítulo se realiza una descripción general de la herramienta, un análisis de requisitos y una especificación de los casos de uso.

En el cuarto capítulo se describen las fases de diseño e implementación del proyecto, citando primero las tecnologías utilizadas justificando su elección, seguido de la arquitectura general del sistema.

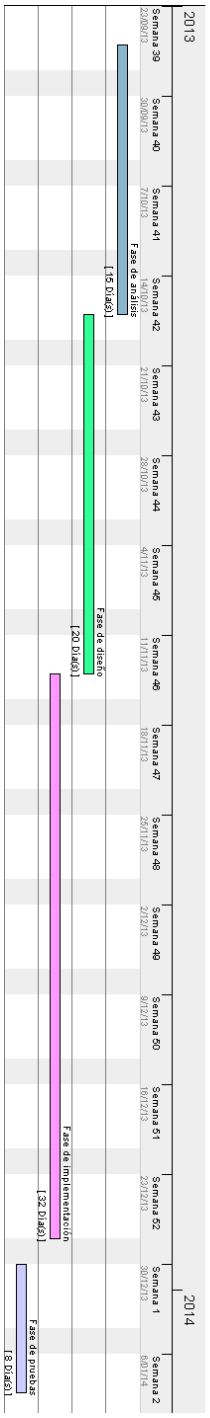

Figura 1.1: Diagrama de Gantt con las fases realizadas

El quinto y último capítulo expone las conclusiones y posibles líneas de trabajo futuras.

Capítulo 2

Fundamentos teóricos

2.1. Vulnerabilidades web y herramientas para realizar tests de penetración web

Para interpretar los resultados del análisis de vulnerabilidades que realiza la aplicación, es necesario comprender primero cada tipo de vulnerabilidad web, es decir, cómo puede ser explotada y su impacto en el sistema. A continuación se describen las vulnerabilidades más comunes según el OWASP Top Ten Project.

2.1.1. OWASP Top Ten Project

OWASP Top Ten Project es un proyecto de Open Web Application Security Project (OWASP) [2] en el que anualmente se publica un documento en el que se describen las vulnerabilidades web más comunes. El documento publicado en 2013 incluye las siguientes vulnerabilidades:

1. Inyección

   La inyección es un método de infiltración de código que explota una vulnerabilidad de la aplicación al realizar la validación de los datos de entrada. Puede causar pérdida o corrupción de datos, pérdida de privilegios o denegación de acceso de manera que pueden comprometer todo el sistema. Hay varios tipos de inyecciones:

   - Inyección SQL

Afecta a las bases de datos. Se explota mediante la introducción de una consulta SQL parcial o completa en un campo de texto de entrada de la aplicación o mediante otros modos de entrada. Es bastante común alterar la sentencia *where* de una *select* añadiendo una condición

```
"or 1=1"
```

que provoca que la condición del *where* sea siempre verdad, por ejemplo, dada una sentencia

```
select * from empleados where id=\$id
```

al modificar la consulta para buscar el empleado con

```
\$id "10 or 1=1"
```

devuelve todas las filas de la tabla empleados.

- Inyección LDAP (Lightweight Directory Access Protocol)

  Es un ataque que se realiza desde el servidor que afecta a los directorios LDAP y puede provocar que información sensible acerca de usuarios y *hosts* sea revelada, creada o modificada. Se explota mediante la introducción de una consulta LDAP parcial o completa, por ejemplo, la consulta

  ```
  searchfilter="(cn="+user+")"
  ```

  se crea a partir de la petición

  ```
  HTTP  http://www.example.com/ldapsearch?user=Juan
  ```

  si se reemplaza por

  ```
  http://www.example.com/ldapsearch?user=*
  ```

  la consulta resulta en

  ```
  searchfilter="(cn=*)"
  ```

  que devuelve cualquier objeto con el atributo *cn* igual a cualquier valor.

- Inyección XPATH

XPATH es un lenguaje que permite, entre otras cosas, la búsqueda de elementos en un fichero xml. Si el atacante tiene éxito podrá saltarse los mecanismos de autenticación o acceder a determinada información sin la autorización necesaria. Por ejemplo, es posible realizar un ataque similar al mostrado anteriormente en el apartado de Inyección SQL. Dado un xml en el que se almacenen nombres y contraseñas de usuario, que se utilice para la autenticación en el sistema, si se utiliza la siguiente consulta XPATH:

```
string(//user[username/text()='Juan' and password/text()='!c3']
/account/text())
```

y si la aplicación no valida correctamente los parámetros de entrada, se podría introducir

```
"' or '1' = '1"
```

en el campo *username* y en el campo *password*. Esto permitiría entrar en el sistema sin proporcionar un nombre de usuario ni una contraseña.

2. Pérdida de autenticación y gestión de sesiones

   Este punto se refiere a vulnerabilidades relacionadas con la autenticación y gestión de sesiones (cuentas expuestas, contraseñas, identificadores de sesión, etc.) que se pueden explotar con el objetivo de suplantar la identidad de los usuarios. Tiene un serio impacto sobre las cuentas de usuario, sobre todo si las cuentas poseen privilegios de administrador. Por ejemplo, si un usuario no cierra sesión explícitamente y si otra persona accede más tarde (antes de que expire la sesión) desde el mismo navegador, podrá suplantar su identidad dentro de la aplicación.

3. Cross-Site Scripting (XSS)

   Este ataque se produce cuando un atacante inyecta código ejecutable por el navegador dentro de una sola respuesta HTTP. Puede tener impacto en la propia aplicación ya que el atacante puede alterar su apariencia e insertar código malicioso y sobre los usuarios ya que el atacante puede secuestrar sus sesiones, redirigir usuarios a otras páginas, secuestrar su navegador, etc.

Hay dos tipos:

- Directo o persistente

  Consiste almacenar el código inyectado en una fuente de datos persistente (por ejemplo, una base de datos o un fichero), por ello es la variante de XSS más peligrosa ya que el ataque se producirá cada vez que la aplicación acceda a esta fuente de datos. Por ejemplo, un atacante podría almacenar lo siguiente:

  ```
  <script>alert(document.cookie)</script>
  ```

  en una fila de base de datos, para que se muestre la *cookie* por pantalla al acceder a ese dato desde la aplicación.

- Indirecto o reflejado

  En este caso el código no se almacena. Por ejemplo, el atacante puede modificar un enlace y luego enviarlo a la víctima para que lo abra.

4. Referencia insegura directa a objetos

   Si no se realizan las comprobaciones de seguridad necesarias para proteger la información o no se hacen correctamente, un atacante que también sea un usuario autorizado en la aplicación, podría llegar a obtener información de otros usuarios sin estar autorizado. Esto tiene impacto directo sobre todos los datos que se referencian por parámetro. Por ejemplo, un usuario crea la siguiente url:

   ```
   http://www.people.com/showMyPersonalData?id=3
   ```

   introduciendo directamente por parámetro un valor que no esté relacionado con sus datos, por ejemplo, el identificador de otra persona, eso le permite ver información privada de otros usuarios.

5. Configuración de seguridad incorrecta

   Si alguno de los recursos que utiliza la aplicación (el servidor web, el servidor de aplicaciones, la base de datos, *frameworks*) o la aplicación en sí, se configura de

forma incorrecta se estarían exponiendo vulnerabilidades que un atacante podría explotar para realizar un ataque y el sistema entero podría verse comprometido. Un posible ataque es buscar contraseñas por defecto, páginas que no se usan, ficheros no protegidos o trazas de error del servidor de aplicaciones o la base de datos con el objetivo de obtener información adicional sobre el sistema.

6. Exposición de datos sensibles

   Si no se cifran los datos sensibles que maneja la aplicación o se cifran de una manera que no es totalmente segura, se estará exponiendo información que podría ser recuperada por un atacante. Por lo tanto, un ataque de este tipo tiene impacto sobre los datos que no hayan sido protegidos y es grave si la información comprometida es sensible (información personal, de salud, bancaria, etc.). Para explotar este tipo de vulnerabilidad los atacantes pueden robar claves o directamente información no cifrada, realizar ataques *man-in-the-middle*, etc.

7. Ausencia de control de acceso a funciones

   Si la aplicación no controla el acceso a funciones reservadas a usuarios con determinados privilegios, un atacante, ya sea un usuario anónimo o autorizado, podría realizar dichas acciones. Las funciones de administración son el objetivo de este tipo de ataques.

8. Cross-Site Request Forgery (CSRF)

   Es un ataque que fuerza al usuario a realizar acciones que no desea en la aplicación web en la que se encuentra actualmente autenticado. El atacante envía al usuario un enlace camuflado (dentro de un *script*, de una imagen, etc.) por ejemplo por correo electrónico y se realiza una petición HTTP que ejecuta una determinada acción sin que el usuario lo sepa. Un ataque de CSRF exitoso puede comprometer la información y las acciones del usuario y si éste posee una cuenta con privilegios de administrador puede comprometer la aplicación entera. Un ejemplo de ataque sería:

```
<img src="https://www.mystore.com/buy?id=3" width="0" height="0">
```

de esta forma cuando se muestre la imagen en realidad se estará realizando una petición HTTP que realiza una determinada acción en la aplicación en la que el usuario se encuentra autenticado.

9. Utilización de componentes con vulnerabilidades conocidas

   Un atacante puede obtener información a través de escaneos o análisis manuales o si hay información sensible expuesta para descubrir y explotar vulnerabilidades conocidas en los componentes que utilice la aplicación. Este tipo de ataques tiene un impacto muy variable en el sistema ya que abarca vulnerabilidades muy diferentes, el impacto podría ser mínimo o podría comprometer el sistema entero.

10. Redirecciones y reenvios no validados

    Un atacante modifica un enlace de redirección de la aplicación para que redirija a un sitio malicioso y lo envía a la víctima. De esta manera consigue instalar *malware* o instar a la víctima para que revele contraseñas o cualquier otra información sensible. También podría utilizar un enlace de la aplicación que lo reenvíe a una página de administración para la cual no está autorizado.

2.2. Herramientas de análisis de vulnerabilidades

2.2.1. Vega

Vega [3] es una herramienta de análisis de vulnerabilidades *open source* desarrollada por Subgraph y una plataforma para testear la seguridad de aplicaciones web. Es una aplicación de escritorio desarrollada en Java que funciona en Linux, OS X y Windows. Incluye un escáner automático y un *proxy* que intercepta peticiones. Además proporciona funciones Javascript para la integración con otras aplicaciones.
Vega proporciona, entre otras, las siguientes funcionalidades:

- Puede funcionar en modo *proxy*, interceptando las peticiones que se realizan durante la navegación y analizándolas para obtener información. Permite establecer puntos de ruptura y criterios de interceptación de peticiones salientes (desde el navegador) o entrantes (desde el servidor).

- Funciona en modo *proxy* de escaneo. Permite hacer *fuzz* de parámetros(realizar peticiones automáticas con múltiples parámetros) y probar activamente las páginas que se visiten.

- Soporta módulos que procesan las respuestas recibidas (normalmente para buscar información) por el escáner o por el *proxy*.

- Genera alertas que incorporan una combinación de contenido dinámico desde el módulo, y el contenido estático en un archivo XML especificado por la alerta.

- Cuenta con una amplia comunidad de desarrollo en la que colaboran tanto desarrolladores individuales como colectivos y cuenta con una amplia documentación sobre la propia herramienta y sobre las librerías en Javascript que proporciona.

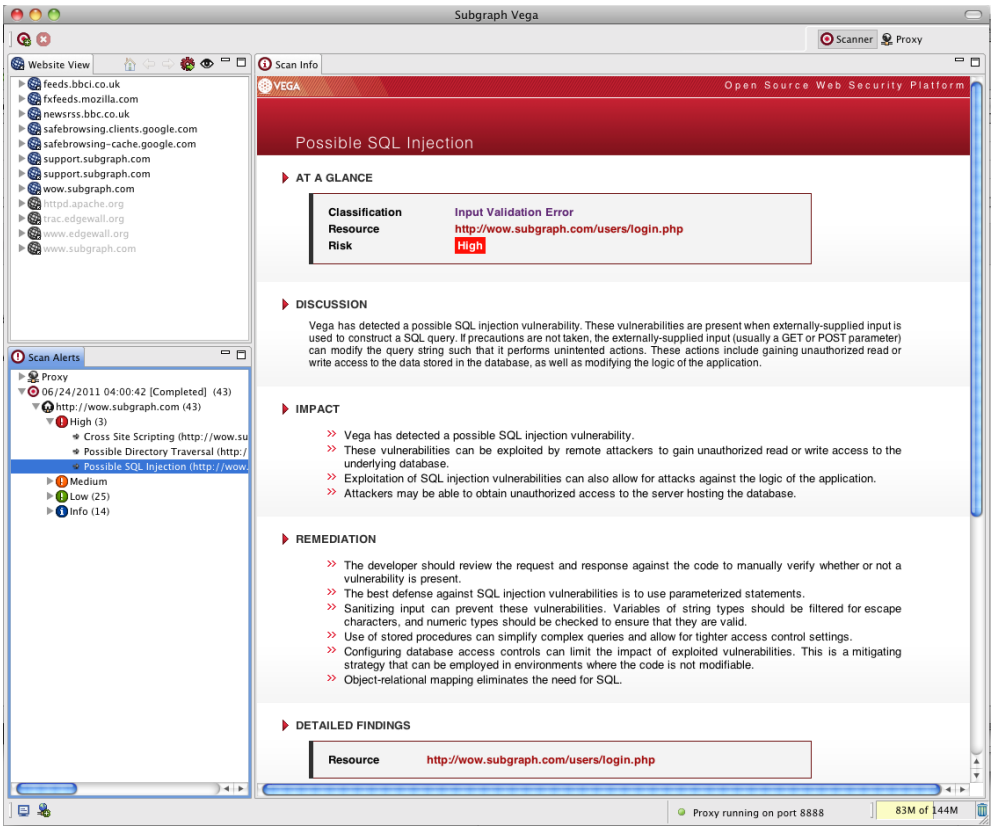

Figura 2.1: Captura de pantalla de la herramienta Vega

2.2.2. Nikto

Nikto [4] es una herramienta *open source* para Linux que escanea servidores web. Realiza pruebas exhaustivas sobre archivos potencialmente peligrosos, CGIs, comprueba versiones no actualizadas y problemas específicos de cada versión. También comprueba la configuración del servidor: si hay archivos de índices, las opciones del servidor HTTP y trata de identificar servidores web y el software instalado.

Estas son algunas de las características principales de Nikto:

- Soporte SSL.
- Funciona como proxy interceptor HTTP.
- Comprueba la existencia de componentes de servidor desactualizados.
- Genera informes en texto plano, XML, HTML, CSV o NBE.
- Motor de plantillas para personalizar informes fácilmente.
- Permite escanear varios puertos en un servidor o varios servidores a través de un archivo de entrada(incluyendo la salida de *nmap* [5]).
- Identifica el software instalado mediante encabezados, *favicons* y archivos.
- Ajuste del escaneado para incluir o excluir comprobaciones de clases de vulnerabilidades.
- Mejora la reducción de falsos positivos a través de varios métodos: encabezados, contenido de la página y *hashing* de contenido.

2.2.3. Wikto

Wikto [6] es una versión de Nikto para Windows que añade algunas funcionalidades extras, por ejemplo, lógica difusa de comprobación de errores, minería de datos minería de directorios asistida por Google y monitorización en tiempo real de petición/respuesta HTTP.

2.2.4. Wapiti

Wapiti [7] es una herramienta para auditar la seguridad de las aplicaciones web. Realiza análisis de caja negra en busca de *scripts* y formularios para actuar como un *fuzzer* e inyectar *payloads* con el objetivo de encontrar vulnerabilidades.

Las características generales de Wapiti son:

- Genera informes de vulnerabilidad en diversos formatos (HTML, XML, JSON, texto plano, etc.).
- Puede suspender y reanudar una exploración o un ataque.
- Permite personalizar la terminal con distintos colores y ajustar los niveles de verbosidad.
- Los módulos de ataque se pueden activar y desactivar rápida y fácilmente.
- Facilidad para añadir nuevos *payloads*.

2.2.5. Paros

Paros [8] es un proxy HTTP/HTTPS basado en Java para la evaluación de vulnerabilidades web. Soporta visualización y edición de peticiones HTTP, incluye *spiders*, certificados de cliente y un escáner de XSS e inyecciones SQL.

Paros es un proyecto descontinuado, pero existe una rama de esta herramienta que aporta funcionalidades a mayores y que se encuentra en actualización continua: OWASP Zed Attack Proxy.

2.2.6. OWASP Zed Attack Proxy Project

Zed Attack Proxy (ZAP) [9] es una herramienta *open source* para la realización de tests de penetración desarrollada por OWASP, que permite detectar vulnerabilidades en aplicaciones web.

Algunas de las funcionalidades de ZAP se citan a continuación:

- Funciona como un *proxy* que intercepta todas las peticiones que se realizan y las respuestas recibidas (útil sobre todo para ver peticiones que no sean evidentes a simple vista). También permite establecer puntos de ruptura para cambiar peticiones y respuesta sobre la marcha.
- Análisis activo de vulnerabilidades. ZAP realiza ataques conocidos contra las aplicaciones web seleccionadas con el objetivo de encontrar vulnerabilidades potenciales. Este análisis sólo puede encontrar ciertos tipos de vulnerabilidades, por lo que debe completarse con pruebas de penetración manual.
- Análisis pasivo de vulnerabilidades. Este tipo de análisis no modifica las respuestas y no ralentiza la exploración ya que se realiza en segundo plano.
- Detección automática de nuevos recursos (*spider*). A partir de una lista de enlaces, esta herramienta identifica todos los hipervínculos de la página y accede a ellos repitiendo el proceso de forma recursiva.
- Autenticación y gestión de sesiones.
- Proporciona un API Rest (disponible en JSON, HTML y XML) para la integración con otras herramientas, que permite el uso de las funcionalidades de escaneo activo y *spider*, aunque en futuras versiones de ZAP se aumentará el número de funcionalidades disponibles a partir del API.
- Actualizaciones automáticas.
- *Plugins* integrados y un *marketplace* de *plugins* en constante actualización.

Se ha escogido la herramienta OWASP Zed Attack Proxy por ser una de las pocas herramientas que facilitan su integración con otras aplicaciones al disponer sus funcionalidades en un API Rest. De esta manera, la aplicación puede delegar funcionalidades sin acoplar su implementación a una herramienta en concreto.

Además es una herramienta en constante actualización desarrollada por un organismo importante dentro del campo de la seguridad informática.

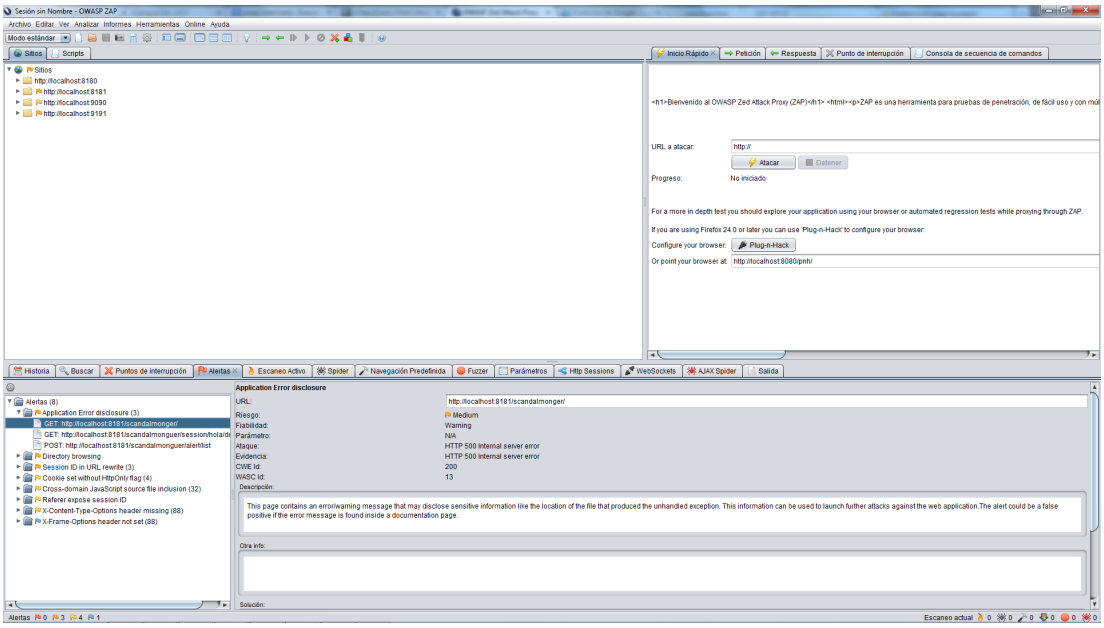

Figura 2.2: Captura de pantalla de la herramienta Zed Attack Proxy

Capítulo 3

Análisis

3.1. Descripción general del sistema

Este proyecto consiste en diseñar e implementar una aplicación web que permita monitorizar el estado de seguridad de cada una de las aplicaciones web propias mediante análisis activos de detección de posibles vulnerabilidades.

La aplicación proporciona diversas utilidades que se describen a continuación:

- Los análisis realizados se guardan bajo una sesión que agrupa el conjunto de aplicaciones escaneadas y las máquinas en las que se ubican. Esto permite ver un histórico de los análisis realizados ordenado por fecha y las vulnerabilidades encontradas en el momento del análisis.

- Cada aplicación escaneada tiene un estado que depende de las vulnerabilidades que se hayan encontrado al realizar el análisis. El estado de cada aplicación será *muy vulnerable* si hay alguna vulnerabilidad de riesgo alto, *vulnerable* si hay alguna de riesgo medio y *seguro* si hay alguna de riesgo bajo, un aviso o no hay ninguna. El estado de la máquina es similar, será *muy vulnerable* si hay alguna aplicación con alguna vulnerabilidad de riesgo alto, *vulnerable* si hay alguna con alguna vulnerabilidad de riesgo medio y *seguro* si hay alguna con alguna vulnerabilidad de riesgo bajo, un aviso o no hay ninguna. Es importante destacar que, aunque el estado de la aplicación o la máquina se indique como *seguro*, no

implica que sea seguro al 100 %, ya que no es posible asegurar que una aplicación esté libre de vulnerabilidades. Además se indica la fecha y hora del último análisis realizado.

- La aplicación realiza análisis activos para detectar posibles vulnerabilidades. Para ello realiza ataques conocidos, si alguno de ellos tiene éxito significa que existe una vulnerabilidad. Debido a estos ataques, los análisis sólo deben realizarse sobre aplicaciones de desarrollo propio y nunca sobre aplicaciones externas. El resultado de los análisis es un listado de las vulnerabilidades encontradas que detalla la vulnerabilidad, la url a la que afecta, una posible solución, referencias a documentación, etc.

3.2. Requisitos del sistema y casos de uso

Los casos de uso representan el comportamiento del sistema especificando los requerimientos funcionales que debe cumplir la aplicación desde la perspectiva de los usuarios. Describen la interacción entre el sistema y los actores (un usuario, subsistema o dispositivo) mediante secuencias de mensajes.

3.2.1. Actores del sistema

A partir del análisis de requisitos se identifica un único actor en la aplicación, que podría ser el administrador del sistema o un profesional de la seguridad informática y que tendría acceso a todo el sistema.

3.2.2. Casos de uso

- Crear nueva sesión: los resultados de los análisis que se realizan se guardan asociados a una sesión, por lo tanto, para realizar un análisis es necesario crear un nueva sesión o abrir una sesión existente.

- Abrir sesión existente.

- Ver máquinas y aplicaciones escaneadas: las aplicaciones escaneadas se ubican en una máquina determinada. Cada máquina puede ubicar varias aplicaciones.
- Escanear aplicación: se realiza un análisis activo de la aplicación para detectar vulnerabilidades.
- Ver histórico de vulnerabilidades: los resultados de todos los análisis realizados se guardan en la base de datos para que sea posible consultarlos en cualquier momento.

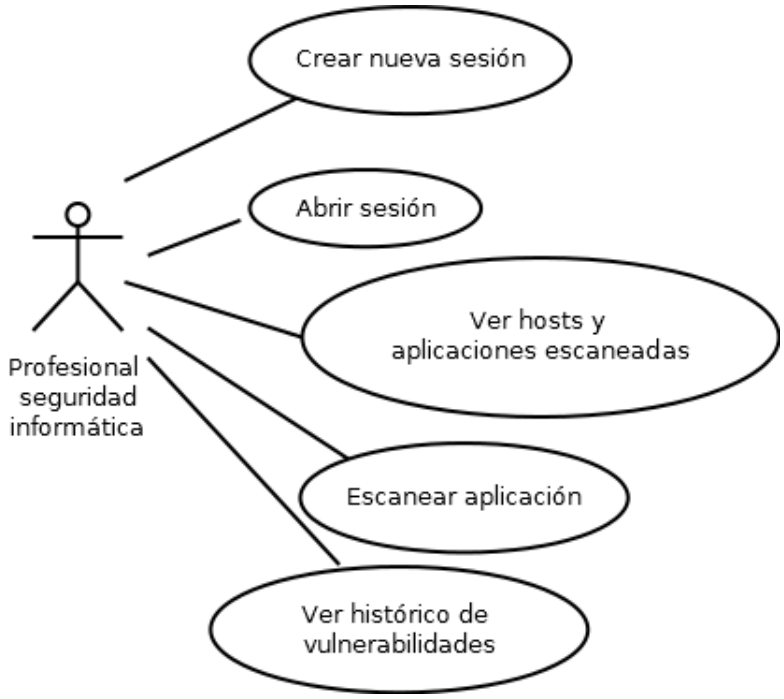

Figura 3.1: Diagrama de casos de uso

3.3. Análisis del funcionamiento OWASP Zed Attack Proxy

En contraste con la interfaz gráfica de ZAP, que posee una gran cantidad de documentación, incluyendo vídeos y manuales de usuario, la documentación del API Rest de ZAP es casi inexistente. Por este motivo, gran parte del tiempo de análisis de este

proyecto se ha dedicado a analizar la herramienta en sí y el API Rest. Este análisis ha consistido en inspección del código fuente, búsquedas en Internet (artículos de varios *blogs*, sección de incidencias del proyecto, etc.) y visualización de vídeos.

Capítulo 4

Diseño e implementación

4.1. Tecnologías

La figura 4.1 muestra un diagrama estructural de la aplicación con las tecnologías utilizadas en este proyecto. A continuación se describen brevemente, justificando su uso y comparándolas con otras tecnologías existentes.

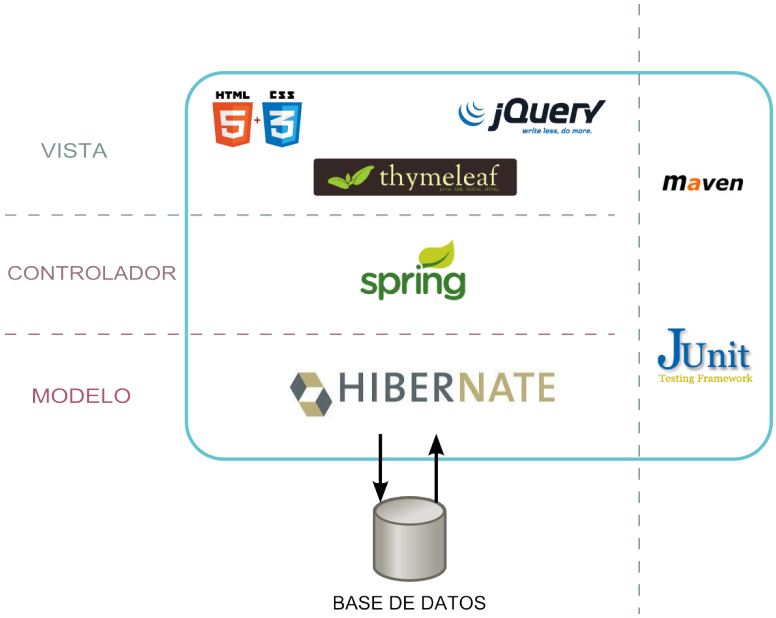

Figura 4.1: Diagrama que muestra el *stack* tecnológico de la aplicación

4.1.1. Elección de la plataforma

Java EE (Java Enterpise Edition, anteriormente conocido como J2EE) [10], es una plataforma para desarrollar aplicaciones empresariales compuesta por un conjunto de especificaciones para el lenguaje Java.

Una aplicación construida con esta plataforma no depende de una implementación concreta, existiendo múltiples implementaciones de distintos fabricantes, algunas de las cuales son *open source*.

4.1.2. Servidor de aplicaciones

Un servidor de aplicaciones se encarga de procesar peticiones HTTP y devolver respuestas dinámicas. Además proporciona acceso a una serie de servicios: seguridad, gestión de transacciones, directorio de nombres JNDI, conectividad remota, etc.

Existen múltiples servidores de aplicaciones comerciales y *open source* para Java EE. Para el desarrollo de este proyecto se ha utilizado Apache Tomcat [11] pero se podría haber utilizado cualquier otro, por ejemplo Jetty [12].

4.1.3. Sistema de gestión de bases de datos (SGBD)

Un SGBD o Sistema de Gestión de Bases de Datos permite crear y mantener una base de datos asegurando su integridad, confidencialidad y seguridad.

El proyecto se ha diseñado de manera que el acceso a la base de datos se realice de forma transparente, por ello cualquier SGBD compatible con una base de datos relacional es perfectamente válido. Entre los SGBD *open source* se encuentran PostgreSQL [13] y MySQL [14]. En este caso, se ha escogido la segunda. Otra opción igualmente válida sería Oracle [15], que aunque se trata de *software* comercial ofrece licencias gratuitas para estudiantes.

4.1.4. Hibernate

JDBC (Java DataBase Connectivity) es un API estándar que permite establecer conexión con una base de datos relacional o acceder a una fuente de datos tabular, construir consultas y procesar los resultados.

Hibernate es un *framework open source* que proporciona una capa de abstraccción sobre JDBC. Hibernate establece una relación entre el modelo de objetos del dominio de la aplicación y una base de datos relacional. Este tipo de herramientas se conocen como mapeadores objeto/relacional (ORM).

4.1.5. Spring MVC

Spring MVC [16] es un *framework* web *open source* orientado a acción para implementar las capas modelo, controlador y vista de una aplicación J2EE.

Se ha escogido Spring MVC para la realización de este proyecto porque permite implementar aplicaciones con una estructura clara de capas y como consecuencia un código más limpio y sencillo.

4.1.6. Thymeleaf

Thymeleaf [17] es una librería Java que implementa un motor de plantillas de XML /XHTML/HTML5 (también extensible a otros formatos) que puede ser utilizado tanto en modo web como en otros entornos no web.

JSP (JavaServer Pages) [18] es una tecnología Java que permite generar contenido dinámico para la web, en forma de documentos HTML, XHTML, XML o de otro tipo.

Se ha escogido Thymeleaf para la realización de este proyecto por ser una mejor alternativa a JSP y porque proporciona un módulo para la integración con Spring MVC.

4.1.7. XHTML, CSS y JQuery

Para el diseño de la interfaz de usuario se han utilizado las tecnologías: XHTML 1.0 Estricto para definir la estructura, CSS 3.0 para definir el estilo y JQuery [19] para el comportamiento.

4.1.8. JUnit

Para realizar las pruebas de unidad de la capa modelo se ha utilizado JUnit 4 [20], un *framework open source* que utiliza aserciones para comprobar los resultados esperados,

organiza los casos de test en grupos y proporciona entornos de ejecución tanto gráficos como textuales.

Puesto que cuentan con muchas similitudes, se podrían haber utilizado igualmente otros *frameworks* de prueba como TestNG [21] o JTiger [22]. Sin embargo, se ha elegido JUnit 4 por ser uno de los *frameworks* de *testing* más antiguo y un estándar de facto.

4.1.9. Maven

Maven [23] es una herramienta de integración continua. Facilita el proceso de compilación, la gestión de dependencias, la realización de pruebas de unidad e integración, la generación de documentación del proyecto, etc. Es un estándar de facto para la realización de estas tareas.

4.2. Arquitectura

El diseño de este proyecto se ha realizado siguiendo la arquitectura presentada en la figura 4.2.

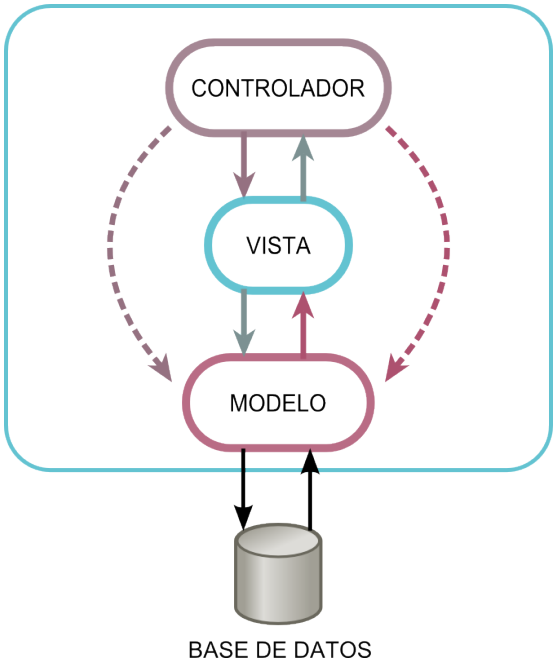

Figura 4.2: Arquitectura de la aplicación

El patrón arquitectónico Model-View-Controller (MVC) se basa en el patrón arquitectónico Layers, el cual permite estructurar la aplicación en capas independientes ocultando la tecnología y evitando acoplamientos innecesarios. Se han utilizado además otros patrones de diseño aplicados para el diseño de funcionalidades específicas [24]. El patrón MVC define tres capas:

- Capa modelo: encapsula el estado de los objetos de la aplicación. Contiene lógica de negocio necesaria para implementar los casos de uso de la aplicación y gestionar las tareas de persistencia en la base de datos.
- Capa vista: proporciona una interfaz que representa los datos obtenidos del modelo y recoge eventos del usuario que envía al controlador. Es responsable de mantener la consistencia de su representación cuando se producen cambios en el modelo.
- Capa controlador: traduce los eventos del usuario invocando acciones en el modelo y comunicando cambios a la vista.

4.3. Descripción de la capa modelo

4.3.1. Modelo de objetos del dominio

A partir del análisis de requisitos se han obtenido los siguientes objetos del dominio de la capa modelo: sesión(*session*), escaneo (*scanning*), alerta (*alert*), máquina (*host* y aplicación (*site*)).

Como se observa en la figura 4.3 una sesión engloba varios escaneos y cada uno de ellos contiene una serie de alertas y se asocia a una sola aplicación. Cada máquina puede ubicar varias aplicaciones.

4.3.2. Modelado de la base de datos

A partir del modelo anterior de objetos del dominio es posible obtener de forma sencilla el modelo relacional. En éste se define cómo se guardará la información en las

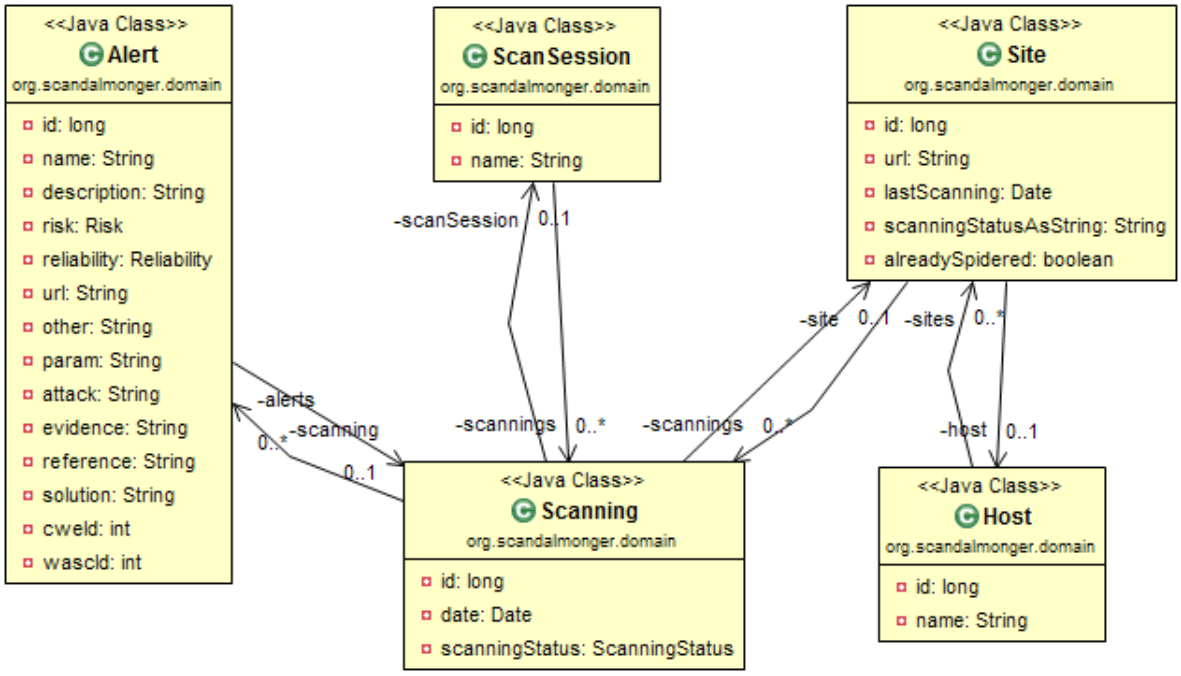

Figura 4.3: Modelo de objetos del dominio

tablas de la base de datos y las relaciones entre las mismas. El gráfico de la figura 4.4 muestra el modelo relacional para la aplicación.

4.3.3. Acceso a datos

El acceso y la actualización de la información persistente se realiza de forma transparente al sistema de almacenamiento de datos desacoplando la lógica de acceso a datos de la lógica de negocio.

Para conseguir este propósito, se ha creado una clase general que implementa las operaciones CRUD de acceso a datos (*Create*, *Read*, *Update* y *Delete*) siguiendo el patrón *Data Access Object (DAO)*. De esta manera, las clases DAO específicas para cada tabla no tienen que implementar las operaciones CRUD básicas y sólo definen operaciones específicas. Además, Hibernate permite desacoplar la implementación de los DAOs del SGBD elegido, que puede ser cualquier SGBD relacional, tan solo requiere modificar algunos parámetros de configuración. Con este enfoque, se puede hacer *plug-*

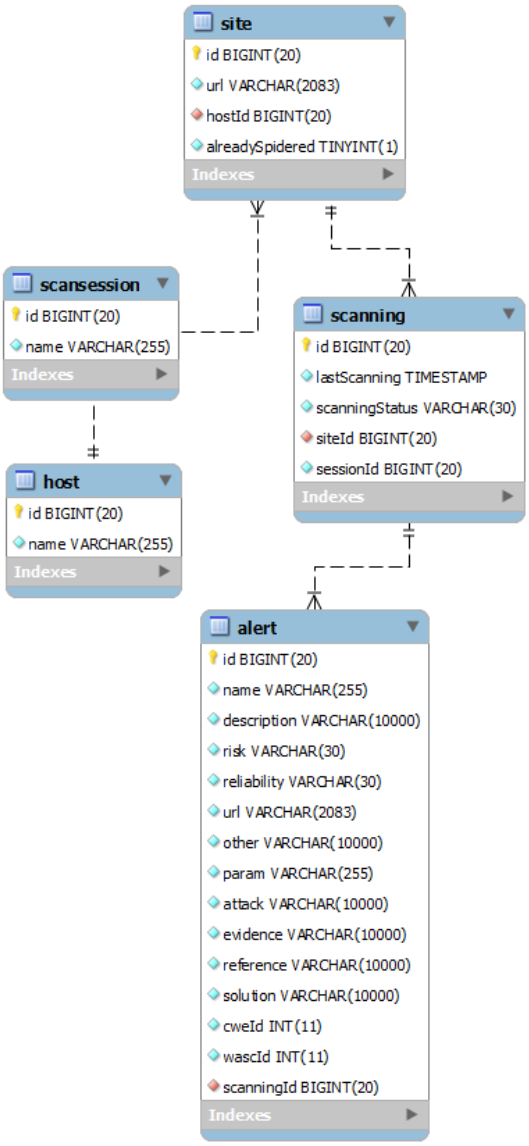

Figura 4.4: Diagrama relacional

and-play en el modelo, es decir, no es necesario recompilarlo en el caso de que se sustituya la base de datos, tan sólo hay que modificar la configuración de la aplicación.

La figura 4.5 muestra la implementación de los DAOs existentes en la aplicación.

4.3.4. Servicios de la capa modelo

La capa modelo proporciona una serie de servicios que implementan los casos de uso de la aplicación (descritos en la sección 3.2.2) y proporcionan soporte al controlador y la vista. Se ha implementado un servicio por cada grupo de casos de uso.

Las funcionalidades que implementan los servicios se ejecutan en un entorno transaccional, con el fin de que el sistema sea asegure la consistencia de datos en la BD, realizando *rollbacks* (deshacer los cambios) en caso de que ocurra un error grave (p.ej: caída de la base de datos).

Para acceder a las funcionalidades de ZAP (escaneo activo, *spider* y gestión de sesiones) mediante el API Rest, se han implementado una serie de clientes Rest que realizan peticiones al API y reciben una respuesta en JSON (también podría ser en XML) para su posterior deserialización en un objeto.

La figura 4.6 muestra los servicios existentes en la aplicación.

4.4. Descripción de la capa controlador

4.4.1. El patrón Front Controller en Spring MVC

El patrón Front Controller [24] define un componente único *(Servlet Front Controller)* responsable de procesar las peticiones HTTP. Además, centraliza funciones como la selección de la vista, seguridad, gestión de errores, etc. y delega otras, como el procesamiento de la lógica de negocio (figura 4.7).

En Spring MVC, el componente centralizado es la clase *DispatcherServlet*, que delega la petición en el controlador correspondiente, esta correspondencia la obtiene la clase *MappingHandler*. El *MappingHandler* relaciona la URI de la petición entrante con un controlador. Estas clases realizan una acción determinada encapsulando llamadas al modelo (a través de una servicio) y seleccionan la vista adecuada creando una

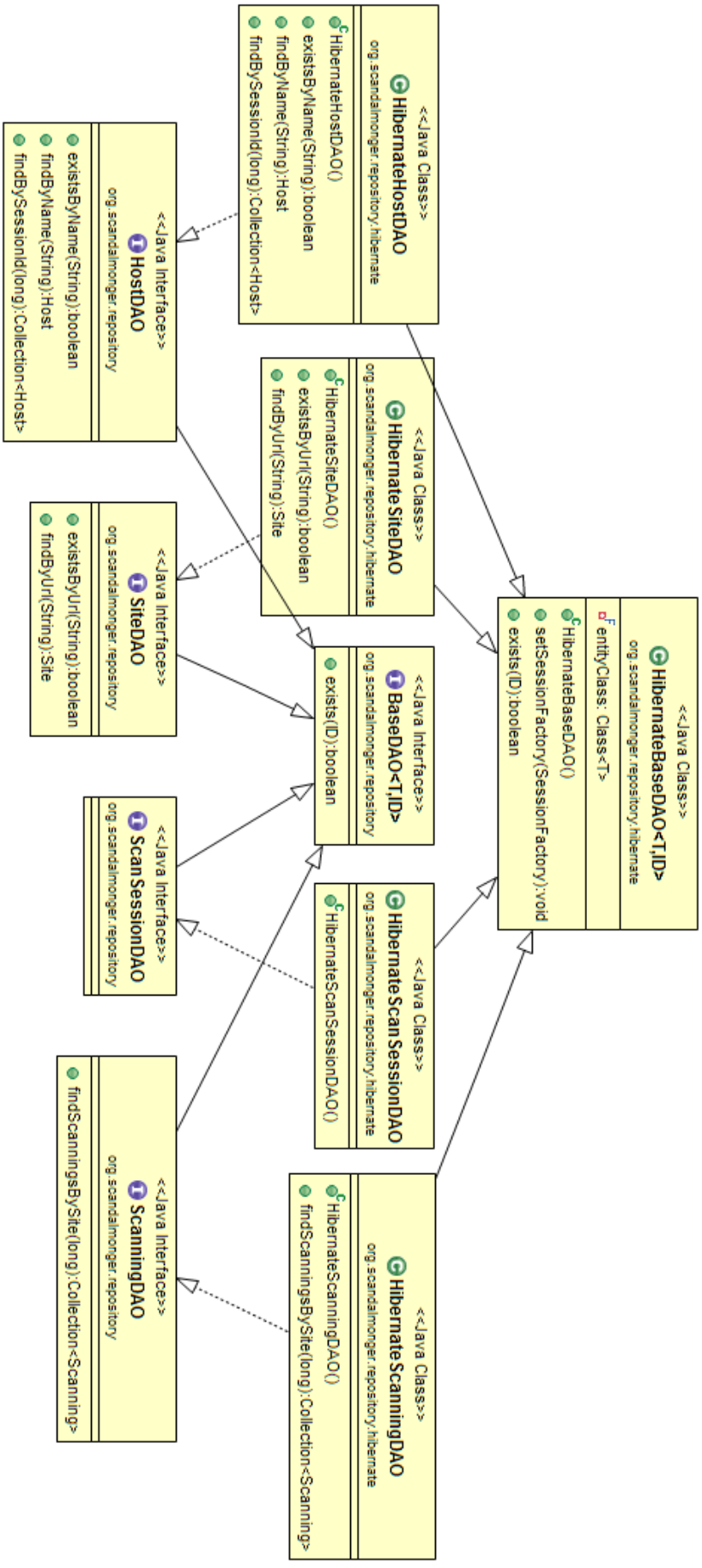

Figura 4.5: Clases que intervienen en el acceso a datos

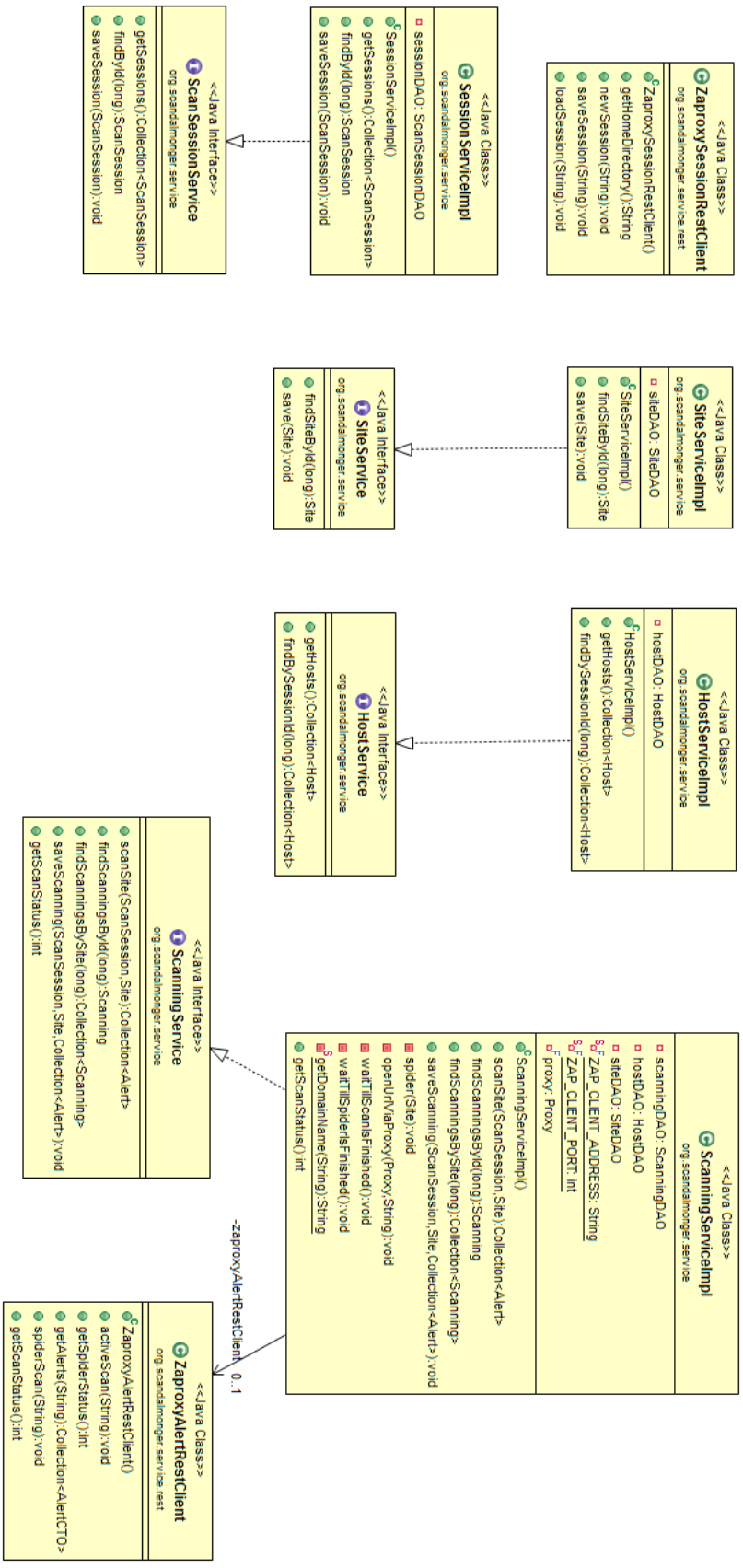

Figura 4.6: Servicios de la capa modelo

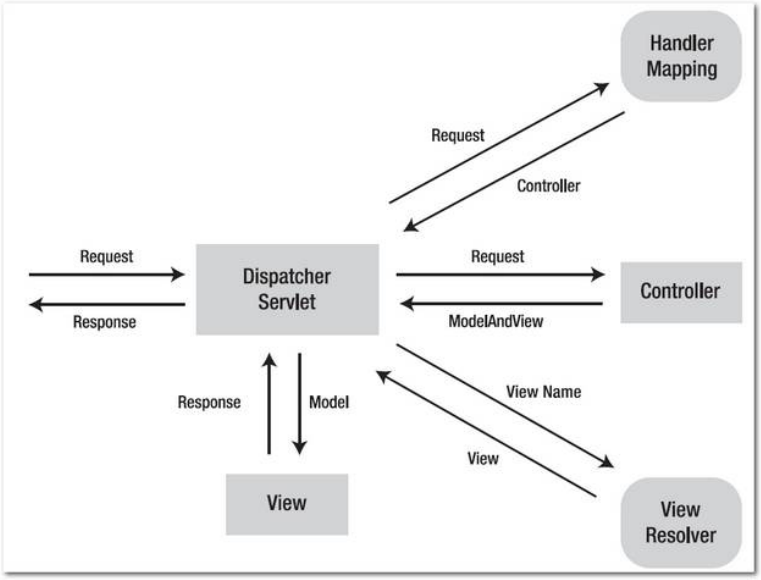

Figura 4.7: Arquitectura de Spring MVC

respuesta.

La figura 4.8 muestra los controladores existentes en la aplicación.

4.5. Descripción de la capa vista

4.5.1. Internacionalización

Una aplicación proporciona soporte para internacionalización (abreviado *i18n*) cuando se visualiza correctamente en distintos idiomas y en diferentes países o regiones.

Spring MVC proporciona soporte para internacionalización, consiste en un mecanismo de traducción basado en ficheros de mensajes de forma que la aplicación esté disponible en varios idiomas. Cada fichero corresponde a un idioma y contiene una serie de pares clave valor. El fichero de mensajes se selecciona dependiendo del objeto *Locale*, definido por Spring MVC, que representa una asociación de idioma y país y que se incluye en la sesión de usuario. De esta forma, cada usuario conectado a la aplicación puede tener un objeto *Locale* distinto.

Esta aplicación sólo se encuentra disponible en castellano, pero proporciona soporte

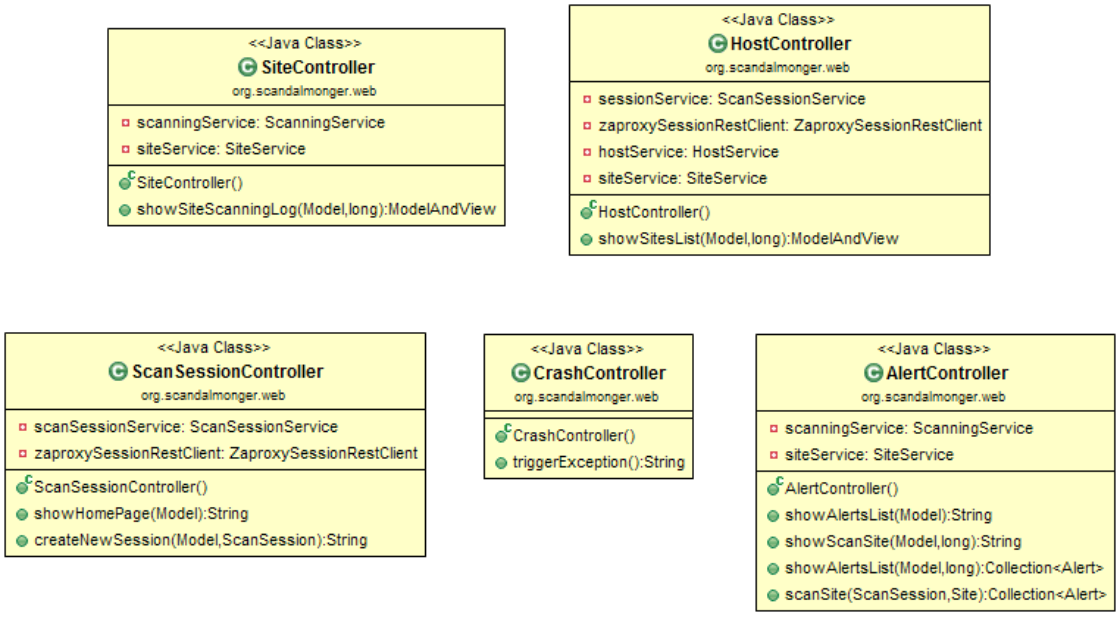

Figura 4.8: Controladores de la capa controlador

de internacionalización para definir otros idiomas sin realizar cambios en la estructura.

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones

El análisis de las vulnerabilidades que puedan tener las aplicaciones es clave para la seguridad de todo el sistema ya que una única vulnerabilidad puede ser un punto de entrada en el sistema para un atacante.

Existen multitud de herramientas de análisis de vulnerabilidades, algunas de las cuales se han descrito en el capítulo 2 . La herramienta desarrollada es multiplataforma, sencilla y fácil de utilizar y aunque delega en otra herramienta externa para el análisis de vulnerabilidades también podría utilizar una implementación propia.

Al inicio del proyecto se establecieron los objetivos que debía cumplir la aplicación, tanto a nivel general como a nivel de funcionalidades. La aplicación desarrollada permite monitorizar la seguridad de los sistemas mediante el análisis periódico de las vulnerabilidades existentes en las aplicaciones web que albergan.

Además, el uso del patrón arquitéctonico MVC, que permite mantener separada la implementación de cada capa y el uso de otros patroñes de diseño y tecnologías, hace que la aplicación resulte fácil de mantener y sea fácilmente escalable.

Gracias a este proyecto se han aprendido nuevas tecnologías (Spring MVC, Thymeleaf, JQuery) y se ha experimentado con varias herramientas de análisis de vulnerabilidades, algunas de las cuales se han estudiado en profundidad (Zed Attack Proxy y Paros).

5.2. Mejoras futuras

En esta sección se proponen posibles ampliaciones o mejoras referidas tanto a las funcionalidades de la aplicación como a las tecnologías utilizadas.

- Diseño e implementación de una librería de análisis de vulnerabilidades propia y hecha a medida, para sustituir la delegación en herramientas externas y evitar así las limitaciones de la misma o bien facilitar las dos opciones, para que el usuario pueda elegir.
- Generación de informes y estadísticas con las incidencias de cada tipo de vulnerabilidad.
- Generación de informes y estadísticas con datos sobre la frecuencia de los escaneos.
- Permitir la configuración de la herramienta pudiendo especificar las reglas a utilizar en los escaneos, urls a ignorar, el tiempo entre peticiones, el número de hilos a utilizar en el proceso, etc.
- Pemitir la personalización de la herramienta (número de elementos mostrados por defecto, elementos favoritos, accesos rápidos, etc.).

Apéndice A

Manual de usuario

A.1. Introducción

SCANdalmonger (juego de palabras en inglés entre *scan* (escanear) y *scandalmonger* (chismoso)) es una aplicación que permite realizar escaneos de aplicaciones web propias (aunque estén alojadas en máquinas diferentes) con el objetivo de detectar vulnerabilidades web y monitorizar la seguridad del sistema.

A.2. Instalación

Para instalar la aplicación siga los siguientes pasos:

1. Primero, es necesario crear una nueva base de datos. La aplicación es independiente del tipo de base de datos, por lo que no tiene que ser una base de datos en concreto. Después de elegir una base de datos, ejecutar el *script schema.sql* incluido en la aplicación, para crear la base de datos y las tablas que utiliza la aplicación. Este *script* se encuentra en se encuentra en el directorio *\WEB-INF\classes*, dentro del archivo comprimido *scandalmonger-1.0.0.war* en el que se empaqueta la aplicación. En el mismo directorio se encuentra el archivo de propiedades *data-access.properties* que es necesario configurar para que la aplicación pueda conectarse a la base de datos.

2. Después, hay que desplegar el archivo *scandalmonger-1.0.0.war* en un contenedor

de aplicaciones Java EE, por ejemplo Apache Tomcat [11] y configurar la fuente de datos (*datasource*) en el mismo. Es importante utilizar un puerto distinto al que utiliza Zed Attack Proxy (que por defecto es el 8080), por ejemplo, el 8181.

3. También es necesario instalar la herramienta Zed Attack Proxy (versión 2.2.2) que se encuentra disponible en la página `https://code.google.com/p/zaproxy`.

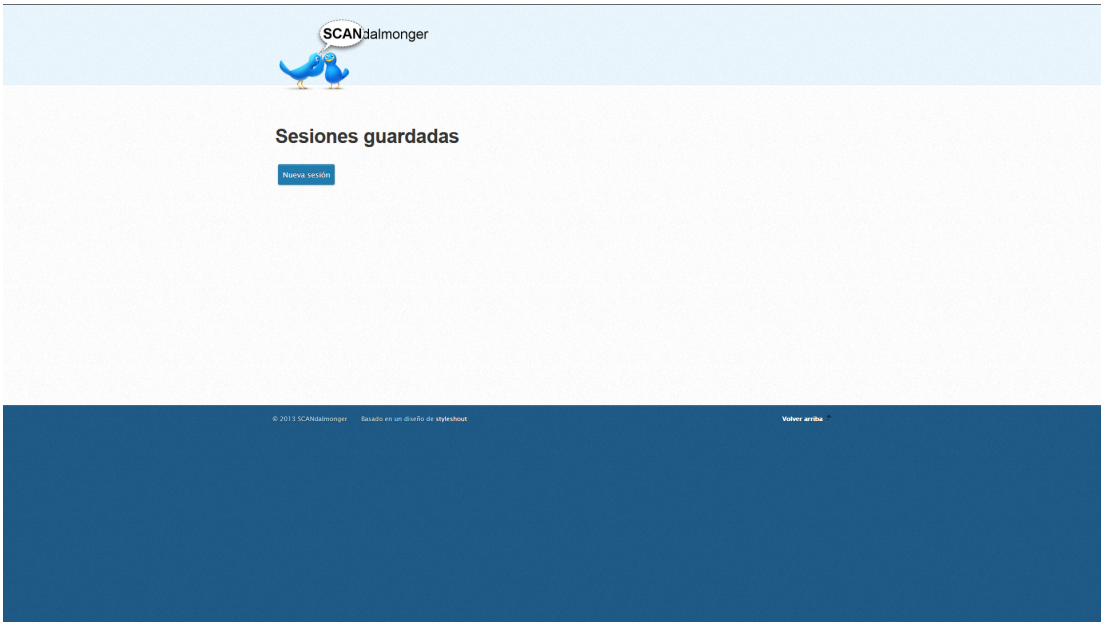

Figura A.1: Pantalla principal de SCANdalmonger

A.3. Acceso a la aplicación

Antes de acceder a la aplicación hay que arrancar Zed Attack Proxy. Aunque se puede abrir en modo gráfico, es recomendable arrancarla en modo demonio, ya que si se va a trabajar con SCANdalmonger, ya proprociona un entorno gráfico. Para arrancar Zed Attack Proxy en modo demonio, desde una consola, hay que ejecutar el siguiente comando:

```
zap.bat -daemon (en Windows)
```

`zap.sh -daemon (en Linux)`

A la aplicación se accede desde un navegador, escriendo en la barra de direcciones: `http://localhost:8181/scandalmonger/`

A.4. Sesiones

Al entrar en la aplicación se muestra una lista de las sesiones de escaneo existentes. Una sesión de escaneo agrupa todas las máquinas y aplicaciones que se han escaneado en algún momento durante esa sesión y las vulnerabilidades (o alertas) generadas.

A.4.1. Creación de una nueva sesión

Para empezar a trabajar con SCANdalmonger es necesario entrar en una sesión, como todavía no existe ninguna hay que crear una nueva. Si se pulsa en el botón *Nueva sesión* se muestra un cuadro de dialógo en el que hay que introducir un nombre para la nueva sesión A.2, por ejemplo, Sesión 08-01-2014.

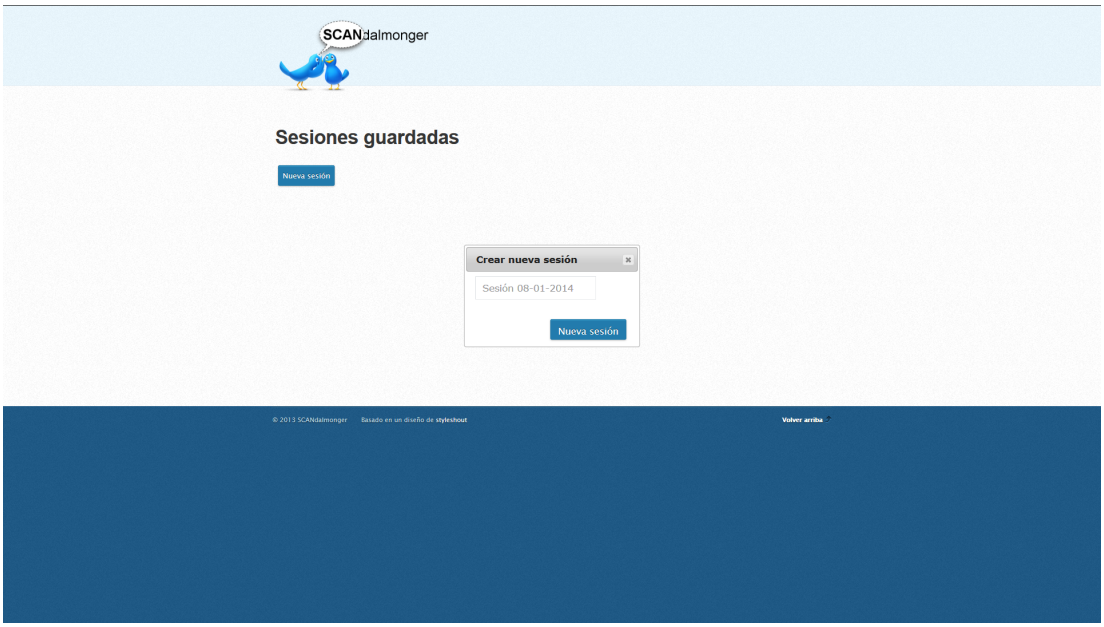

Figura A.2: Cuadro de dialógo para la creación de una nueva sesión

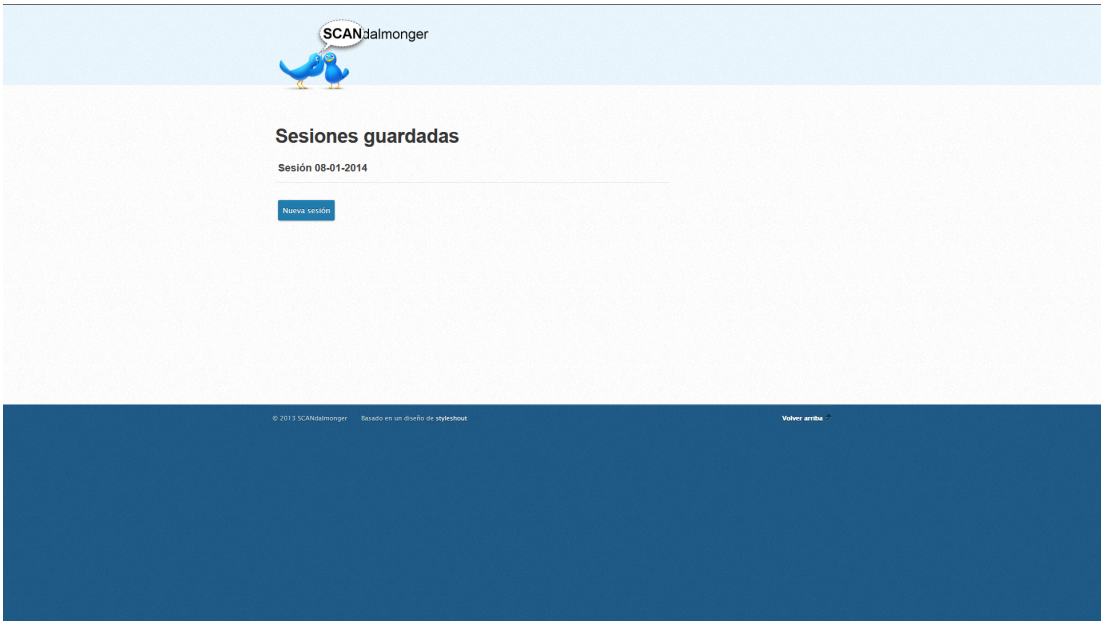

Figura A.3: Nueva sesión Sesión 08-01-2014 creada

A.5. Escaneo de aplicaciones

Al pulsar sobre el nombre de la nueva sesión creada (figura A.3), se muestra la pantalla en la que irán apareciendo los nombres de las máquinas y aplicaciones (o sitios) que se vayan escaneando.

A.5.1. Escanear una nueva aplicación o sitio

Si se pulsa en el botón *Nuevo sitio* se mostrará la pantalla de análisis de vulnerabilidades. En el campo de texto hay que introducir la url de la aplicación a escanear. Podria ser por ejemplo, la aplicación Hacme Casino A.4, que es una aplicación vulnerable diseñada específicamente para realizar tests de penetración. El resultado del análisis se puede ver en la figura A.5.

A.5.2. Detalle de una vulnerabilidad

En cada fila de la tabla se muestra una vulnerabilidad, si se pulsa en el enlace *Detalles* de una fila, se mostrará una ventana con los detalles de esa vulnerabilidad

Figura A.4: Aplicación Hacme Casino

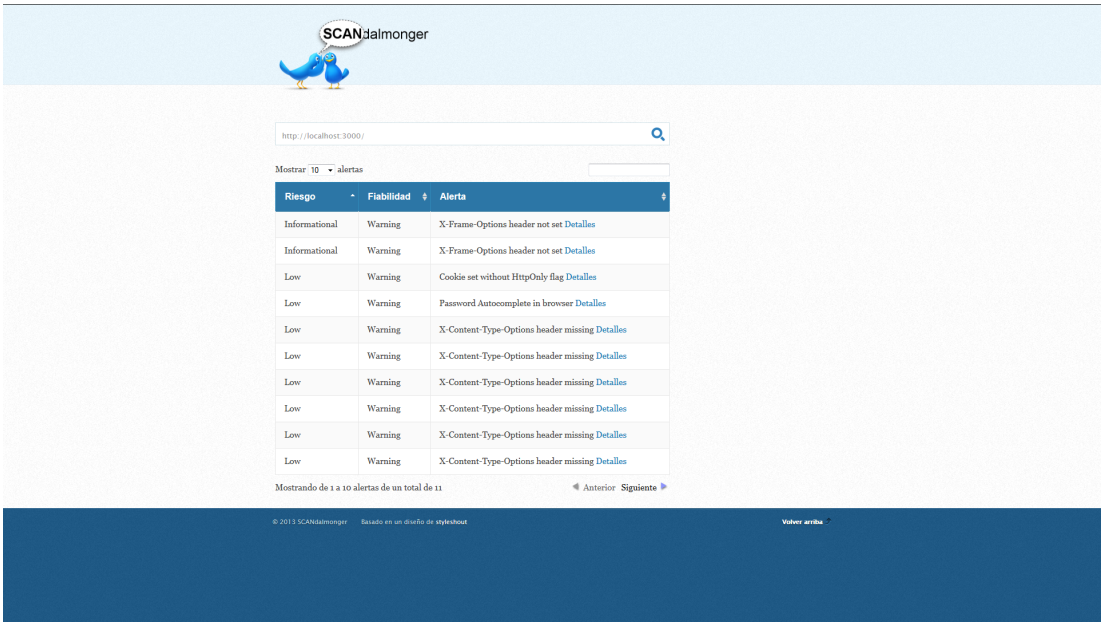

Figura A.5: Vulnerabilidades detectadas en la aplicación Hacme Casino

(ataque, evidencia, referencias, solución, etc.).

A.5.3. Lista de máquinas y sitios escaneados

Al volver a la pantalla anterior, se mostrarán las máquinas y aplicaciones escaneadas previamente (localhost), su estado y la fecha del último análisis. El estado de la aplicación se representa gráficamente por un semáforo, si hay alguna vulnerabilidad de riesgo alto el semáforo se mostrará de color rojo, si hay alguna de riesgo medio, será amarillo y si hay alguna de riesgo bajo, un aviso o no hay ninguna, será verde. El estado de la máquina se representa de la misma manera, si hay alguna aplicación con alguna vulnerabilidad de riesgo alto el semáforo se mostrará de color rojo, si hay alguna con alguna vulnerabilidad de riesgo medio, será amarillo y si hay alguna con alguna vulnerabilidad de riesgo bajo, un aviso o no hay ninguna, será verde.

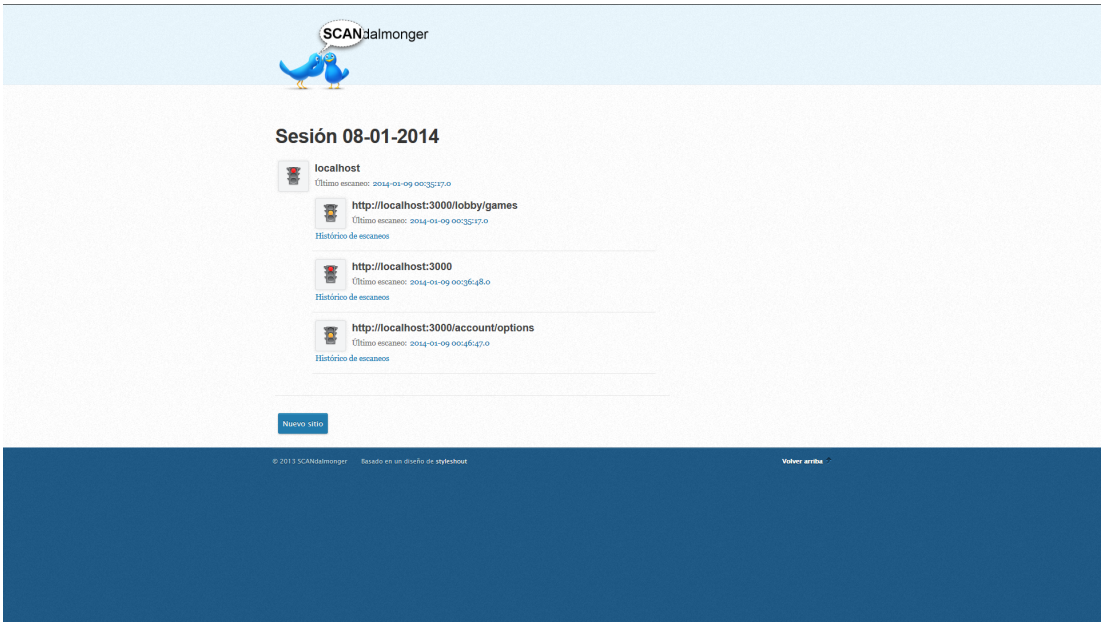

Figura A.6: Lista de máquinas y sitios escaneados

A.5.4. Histórico de escaneos

Debajo de cada sitio escaneado se muestra un enlace *Histórico de escaneos* que muestra una lista de todos los análisis realizados hasta el momento de ese sitio y las

vulnerabilidades que tenían en esos instantes. Esto permite estudiar la evolución de la seguridad de la máquina.

Figura A.7: Histórico de escaneo

Glosario

API

Application Programming Interface. Definición de clases, interfaces y métodos de una librería que permite su uso sin tener conocimiento de su implementación.

Base de datos relacional

Es una base de datos que cumple con el modelo relacional, en la que todos los datos se almacenan y son accedidos por medio de relaciones o tablas.

Cookie

Par clave-valor que se almacena en el navegador y se incluye en la petición HTTP para dotar de estado a este protocolo.

CSS

Cascading Styles Sheet. Hoja de estilos en cascada. Estándar del W3C para la definición del aspecto visual de una página web.

Framework

Conjunto de clases, interfaces y APIs que facilitan el desarrollo y/o la implementación de una aplicación.

JavaBeans

Componentes reusables para el desarrollo de aplicaciones Java. Son objetos serializables que contienen atributos privados y métodos get/set para el acceso y modificación de sus atributos.

JNDI

Java Naming and Directory Interface. API que proporciona un conjunto de interfaces para acceder a un servicio de nombres y directorios.

Open source

Es el término con el que se conoce al *software* cuyo código fuente es público y modificable en función de una licencia (GNU, GPL, LGPL MIT, Apache, etc.).

ORM

Mapeo objeto/relacional. Mecanismo de conversión de objetos Java persistentes a tablas de una base de datos relacional.

SRS

Software Requirements Specification. Descripción completa del comportamiento del sistema *software* a desarrollar. Contiene una serie de requisitos funcionales (casos de uso) y no funcionales o complementarios (requisitos que imponen restricciones en el diseño o la implementación y estándares de calidad.)

URI

Uniform Resource Identifier. Cadena corta de caracteres que identifica un recurso normalmente accesible en una red o sistema (servicio, página, documento, dirección de correo electrónico, etc.).

W3C

Word Wide Web Consortium. Organización encargada de desarrollar tecnologías interoperables, especificaciones, guías, *software* y herramientas para la web.

Wrapper

Término utilizado comúnmente para hacer referencia al patrón Wrapper o patrón Adaptador.

XHTML

eXtensible HyperText Markup Language. Lenguaje de etiquetas estandarizado para la creación de documentos para la web basado en HTML y conforme a XML.

Bibliografía

[1] Sommerville, I., *Software Engineering*. International Computer Science, Addison Wesley, eighth ed., June 2006.

[2] Fundation, O., "Owasp top 10 - 2013," *OWASP*, 2013.

[3] "Vega." `http://www.subgraph.com/products.html`.

[4] "Nikto." `http://www.cirt.net/nikto2`.

[5] "Nmap." `http://nmap.org/`.

[6] "Wikto." `http://research.sensepost.com/tools/web/wikto`.

[7] "Wapiti." `http://wapiti.sourceforge.net/`.

[8] "Paros." `http://www.parosproxy.org/`.

[9] "Owasp zed attack proxy project." `https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project`.

[10] "Java ee at a glance." `http://www.oracle.com/technetwork/java/javaee/overview/`.

[11] "Apache tomcat." `http://tomcat.apache.org/`.

[12] "Jetty." `http://www.mortbay.org/jetty-6/`.

[13] "Postgresql." `http://www.postgresql.org/`, 2008.

[14] "Mysql." `http://www.mysql.com/`, 2008.

[15] "Oracle." `http://www.oracle.com/`, 2008.

[16] "Spring framework." `http://www.springframework.org/`, 2008.

[17] "Thymeleaf." `http://www.thymeleaf.org/`.

[18] "Javaserver pages technology." `http://java.sun.com/products/jsp/`.

[19] "Jquery." `http://jquery.com/`.

[20] Lee, J., "Test framework comparison," *The server side*, July 2005.

[21] "Testng." `http://testng.org/`.

[22] "Jtiger." `http://jtiger.org/`.

[23] "Maven." `http://maven.apache.org/`.

[24] *Core J2EE Patterns*, 2008. 2001-2002 Sun Microsystems.