Incorporando seguridad a las componentes de interfaz de usuario del framework JSF (JAVA Server Faces)

Javier F. Diaz[1], Claudia A. Queiruga[1] y Pablo J. Iuliano[1],

[1] LINTI (Laboratorio de Investigación en Nuevas Tecnologías Informáticas), Facultad de Informática, La Plata, Argentina.
{jdiaz, claudiaq, piuliano}@linti.unlp.edu.ar

Abstract. El framework de desarrollo JSF (JAVA Server Faces) es uno de los más popularmente usados en el desarrollo de aplicaciones Web. Este incorpora la novedad de permitir construir "componentes a medida" o "componentes customizadas", aunque la implementación inadecuada desemboca en aplicaciones con fallas de seguridad. En este trabajo se intenta presentar un camino para proveer seguridad a las aplicaciones desarrolladas en JAVA Server Faces.

1 Introducción

Internet ha cambiado la manera en que las organizaciones e individuos dirigen sus negocios. Al mismo tiempo reformuló el concepto de seguridad informática. Hoy en día, las aplicaciones Web deben pensase en función de proteger los activos de la organización y el de los usuarios de las mismas. Con lo cual, este tipo de aplicaciones deben garantizar de algún modo cierto nivel mínimo de seguridad.

Java Server Faces (JSF o Faces) es el framework estándar para construcción de aplicaciones Web en JAVA incorporado en la distribución de JEE de Sun y actualmente está sumando muchos adeptos [1] [2]. Éste permite construir aplicaciones Web JAVA basadas en componentes de interfaz de usuario. La idea básica de JSF es escribir aplicaciones Web JAVA de la misma manera que se escriben aplicaciones de escritorio [3] usando librerías como SWING y AWT [4]. Sin embargo, a pesar de su poder de extensibilidad y las facilidades que provee JSF para incorporar seguridad en el desarrollo de las aplicaciones son extremadamente acotadas.

El desarrollo de este artículo se enfocará en analizar un conjunto de vulnerabilidades clásicas de las aplicaciones Web, y en particular aplicaciones desarrolladas con Faces proponiendo una estrategia de solución para las mismas. Específicamente la estrategia de solución será la construcción de "componentes a medida" o "componentes customizados" que implementen acciones que mitiguen una o varias de las vulnerabilidades analizadas. Este tipo de estrategia provee centralización, localización y reutilización de las soluciones implementadas.

En la sección 2 de este artículo se describen los fundamentos de Java Server Faces en cuanto al desarrollo de aplicaciones Web. En la sección 3 inicialmente se abordan

conceptos generales de seguridad en la Web y se identifican las vulnerabilidades Web que están dentro del ámbito de interés de este artículo, utilizándose como fuente de información las publicaciones del sitio del proyecto OWASP [5]. La sección 4 y 5 plantearán un escenario que ilustrará la problemática en cuestión, la solución propuesta y finalmente se presentarán las conclusiones.

2 Fundamentos de JAVA Server Faces

JAVA Server Faces (JSF) comúnmente llamado Faces, es un framework JAVA estándar la construcción de interfaces de usuario server-side. Fue desarrollado por el Java Community Process (JCP, JSR 252) [2] y forma parte de la especificación de Java Enterprise Edition (JEE) versión 5.0 [6]. La idea básica de Faces es escribir aplicaciones Web de la misma manera que se escriben aplicaciones de escritorio, tal como se haría con herramientas populares como Microsoft Visual Basic [3] [7] [8], PowerBuilder [3] [7] [9] y Borland Delphi [3] [7] [10]. Una de las ventajas más importantes de JSF es la posibilidad de desarrollar aplicaciones Web al estilo Rapid Application Development (RAD) permitiendo construir rápidamente aplicaciones a partir de un conjunto de componentes de GUI reusables. Así se obtienen desarrollos más productivos en un lapso de tiempo menor, en conjunción con la ventaja de poder construir interfaces de usuario complejas para web de forma razonablemente sencilla.

JSF provee un conjunto de componentes de interfaz de usuario predefinidos (botones, hipervínculos, checkboxes, etc.) un modelo para la creación de componentes de interfaz de usuario customizadas y una técnica para procesar en el servidor los eventos generados en el cliente a partir de la interacción del usuario con la aplicación. Incluso permite sincronizar una componente UI con un objeto de valor (VO) erradicando una cantidad considerable de código tedioso y repetitivo.

Java Server Faces es una abstracción de alto nivel de la API de Servlet. Las aplicaciones construidas con Faces usan el protocolo HTTP para la comunicación entre la parte cliente y la parte servidora y diversas tecnologías para la "Vista" como por ejemplo JSP. El framework no restringe la "Vista" solo a JSP, también se podrían utilizar tecnologías como XML/XSLT, motores de templetizado, HTML, XHTML [11], WML [12] o algún otro tipo de lenguaje que entiendan tanto el cliente como el servidor. Sin embargo, todas las implementaciones de Faces proveen integración con JSP, aunque sea en forma muy básica.

JSF es considerado un framework de aplicaciones Web, debido a que resuelve la mayoría de las tareas tediosas y repetitivas involucradas en el desarrollo web permitiendo al programador enfocarse en tareas más desafiantes como por ejemplo diseñar la lógica de negocio. Unas de las funcionalidades claves de Faces es soportar el patrón de diseño MVC para web o Model 2 [13] [14], el cual separa el código de la presentación del de la lógica de negocio. Sin embargo a diferencia de otros frameworks MVC para web basados en peticiones, Faces usa una solución focalizada en componentes de interfaz de usuario y en el manejo de eventos generados por ellas. El estado de una componente es guardado en el servidor cuando el cliente solicita una nueva página (o genéricamente una vista) y recuperado al retornar la respuesta.

JAVA Server Faces introduce la posibilidad de programar librerías de componentes de interfaz de usuario propias. Esta es una forma concreta y muy potente de desarrollar conjuntos de clases que interactúan juntas para lograr código de interfaz de usuario basado en Web, reusable. La potencia del framework radica en el modelo de las componentes y en la arquitectura extensible que permite que se pueda incrustar extensiones a las ya existentes, mediante los modelos UI, renderers, validadores y conversores [15].

JSF es un framework orientado principalmente al desarrollo de aplicaciones Web y no provee mecanismos nativos para dotar de seguridad a las aplicaciones que se desarrollan con él. Esto conduce a que el desarrollador inexperto o desinteresado por cuestiones de seguridad, construya aplicaciones Faces que sean fácilmente vulneradas. La alternativa a esto último es que el desarrollador se provea él mismo de los mecanismos que no provee el framework para construir aplicaciones con cierto nivel de seguridad.

3 Seguridad en Aplicaciones Web JAVA

Las aplicaciones Web JAVA al igual que las aplicaciones de escritorio que aspiran a ser seguras deberían resolver los siguientes ítems:

- Integridad
- Autenticación
- Autorización
- Confidencialidad

La integridad [16] [17] se refiere a la necesidad que la información que manejan las aplicaciones sea protegida contra posibles alteraciones indeseadas o corrompida de algún modo, ya sea en forma intencional o no.

Autenticación y autorización [16] [17] son dos conceptos que generalmente se implementan juntos, la autenticación determina que un usuario sea quién dice ser y la autorización verifica que la acción que intenta realizar esté permitida. De este modo, es necesario asegurar la identidad del usuario antes que realice cualquier operación en el sistema. Actualmente existe una extensión de Java llamada Java Authentication and Authorization Service (JAAS) que resuelve la problemática antes descripta [16] [17] [18].

La confidencialidad [16] [17] apunta a que los datos o información valiosa no sean divulgados o accedidos por personas o usuarios ajenos al grupo que tiene permitido el acceso.

La integridad y confidencialidad [16] [17] puede ser alcanzada a través de la utilización de alguna herramienta criptográfica, particularmente para JAVA, las APIs de Java Cryptography Architecture (JCA) y Java Cryptography Extension (JCE) [19] [20] resuelven los aspectos de criptográficos en la plataforma. Más allá de esto último, actualmente ni las soluciones JEE, ni ninguno de los frameworks más popularmente utilizados para los desarrollos Web en Java (Struts, Struts 2, Java Server Faces, etc.) prevén soluciones para resolver la problemática de la integridad y confidencialidad. Esta carencia se ve reflejada en la Figura 1.

Figura 1 - FUNCTIONALITIES OFFERED BY JAVA WEB FRAMEWORKS. Extraído de theserverside.com del artículo "Enterprise JAVA Community: Are JAVA Web Applications Secure?"

El desarrollador debe proveer la integridad y confidencialidad programáticamente a través del uso de las APIs criptográficas de Java (JCA y JCE) [19] [20].

La seguridad en las aplicaciones Web puede definirse de varias maneras dependiendo del punto de vista. Un enfoque muy popular para proveer seguridad es identificar las posibles vulnerabilidades de seguridad de las aplicaciones y qué medidas habría que tomar para erradicarlas o en su defecto mitigarlas de alguna manera. Para la tarea de identificación de las posibles vulnerabilidades en las que podría incurrir una aplicación Web, se utilizó como referencia la información que se encuentra en el sitio del proyecto OWASP [4].

El proyecto OWASP (Open Web Application Security Project) [4] es una comunidad dedicada a colaborar con organizaciones para el desarrollo, venta y mantenimiento de aplicaciones Web confiables, según ellos mismos publican en su sitio.

OWASP también ha publicado un listado con las diez vulnerabilidades más comunes y críticas encontradas en los desarrollos Web. El listado de las diez vulnerabilidades más comunes se ha modificado desde su versión original del 2004 y la última actualización data del 2007[21]. Esta última fue la que se utilizó para seleccionar las vulnerabilidades que están dentro del ámbito de interés de este artículo:

A1 – Secuencia de Comandos en Sitios Cruzados (XSS)
A2 – Fallas de Inyección
A6 – Revelación de Información y Gestión Incorrecta de Errores
A8 – Almacenamiento Criptográfico Inseguro
A9 – Comunicaciones Inseguras

Esta clasificación contiene una mezcla de ataques, vulnerabilidades y contramedidas. Por ello se podría volver a organizar los puntos anteriores de la siguiente manera:

- Ataques de "Phishing" que pueden explotar cualquiera de estas vulnerabilidades, particularmente XSS (A1).
- Violaciones de Privacidad debido a una validación insuficiente, reglas de negocio y comprobaciones de autorización débiles (A2, A6).
- Robo de identidad debido a insuficientes o no existentes controles criptográficos (A8 y A9).
- Toma de control de sistemas, alteración de datos o destrucción de datos a través de inyecciones (A2).
- Pérdida de reputación a través de la explotación de cualquiera de estas vulnerabilidades.

En el sitio del proyecto OWASP [4], se detallan un conjunto de soluciones para el desarrollo de aplicaciones Web en JAVA. Dichas soluciones están orientadas a la autorización, autenticación, validación de datos de entrada y protección de XSS. Toda esta información intenta acercar al desarrollador mecanismos de seguridad para lograr aplicaciones Web seguras.

En la actualidad existen distintas soluciones que intentan solucionar algunas de las vulnerabilidades anteriormente descriptas en aplicaciones web JAVA. Entre ellas se destacan la implementación de medidas de seguridad a través de servlets filtros [6], otras mediante el archivo descriptor de la aplicación web (web.xml) y otra basándose en programar la seguridad en forma separada usando tecnología de aspectos [22] y luego compilarla con la aplicación.

4 Componentes JSF Seguros

Unas de las vulnerabilidades menos considerada y tratada es la exposición de información sensible o privilegiada en desarrollos Web. En particular es interés del artículo encontrar mecanismos que permitan tratar esta vulnerabilidad sobre Faces. Es por ello que tomaremos dicha vulnerabilidad para intentar sanearla o al menos mitigarla, planteando una aplicación que exponga datos privados y un camino de solución para ello.

4.1 Planteo de la problemática a tratar

Para graficar la problemática de seguridad que involucra la falta de confidencialidad [16] [17], analicemos el siguiente escenario: supongamos que una

aplicación desarrollada con JSF tiene por funcionalidad realizar búsquedas sobre una tabla de una base de datos de una organización. Existen varios criterios de búsqueda, los cuales son ingresados a través de varios campos de entrada HTML, y el resultado final es mostrado en pantalla.

Una vez que hemos fijado la funcionalidad de la aplicación, si analizamos en detalle poniendo especial énfasis en la seguridad y privacidad de la información, se pone de manifiesto que la exposición o no de información privada queda a criterio de la metodología que utilice el desarrollador para la codificación de la aplicación. Esta última afirmación se sustenta en el hecho que toda la información sensible de la estructura de la tabla podría quedar expuesta en el código HTML si el desarrollador utiliza para identificar los campos de entrada HTML los nombres de los campos de la tabla, técnica popularmente usada entre los desarrolladores. Así con solo inspeccionar el código fuente de la página que retorna el servidor como resultado del requerimiento realizado por el cliente, se obtendría fácilmente la estructura de la tabla en su totalidad. De este modo se estaría dejando abierta la posibilidad que se infiera o deduzca la estructura de la base de datos, con lo cual esta información podría ser utilizada para realizar algún eventual ataque, como por ejemplo la utilización de la técnica de SQL Injection. También es importante resaltar que la eventual inferencia de la estructura de la base de datos de la organización deja disponible información privada que atenta directamente contra los intereses de la misma

La situación descripta anteriormente en términos muy generales, se ha plasmado en una aplicación concreta cuyo objetivo es realizar la búsqueda antes mencionada sobre una tabla llamada persona. Dicha tabla posee campos llamados nombre, apellido, edad y documento, estos son utilizados para almacenar la información relevante de la persona.

Mediante una página JSF se ingresan los valores que servirán para realizar la búsqueda sobre la tabla persona.

Extracto 1: Código JSF de la página que realiza la búsqueda. Se puede apreciar que los campos de búsqueda comparten los mismos nombres con los campos de la tabla persona.

```
<h:inputText id="nombre"/>
<h:inputText id="apellido"/>
<h:inputText id="edad"/>
<h:inputText id="documento"/>
```

Cuando la página JSF sea requerida por un navegador Web de un cliente, se expondrá la estructura de la tabla persona en el código HTML, el cual es devuelto al cliente que generó el requerimiento. Con solo inspeccionar el HTML que retorno el servidor, se tendrá acceso a información que no debería ser publica.

Extracto 2: Código HTML del lado del cliente donde se muestra los nombres de los campos de la tabla en texto plano.

```
<input id="j_id_id1:nombre" type="text"
name="j_id_id1:nombre"/>
```

```
<input id="j_id_id1:apellido" type="text"
name="j_id_id1:apellido"/>

<input id="j_id_id1:edad" type="text"
name="j_id_id1:edad"/>

<input id="j_id_id1:documento" type="text"
name="j_id_id1:documento"/>
```

El desarrollo de una librería de componentes JSF seguras, que considere la problemática descripta en el ejemplo anterior, es la solución o mitigación que propone este trabajo. Las componentes de la librería utilizan las APIs criptográficas de Java (JCA y JCE) [19] [20] para ocultar los valores de las etiquetes ingresados por el desarrollador en los atributos de los componentes. De este modo se alcanzaría la confidencialidad en los datos en las aplicaciones Faces que utilizaren dicha librería.

4.2 Utilización de encriptación simétrica

Para implementar el ocultamiento de la información sensible que deben llevar a cabo las componentes se podría utilizar encriptación simétricas de datos. Esto significa que la misma clave se utilizará para realizar el cifrado y descifrado de los datos [19] [20]. Con lo cual dicha llave deberá ser almacenada en el servidor con las medidas de seguridad que correspondan. Para llevar acabo este tipo de encriptación, se deberá seleccionar uno o más de los algoritmos de encriptación más populares y que son soportados por las APIs JCA y JCE [16] [17]. Dichos algoritmos se describen a continuación:

- DES (Digital Encryption Standard): utiliza una llave de 56 bits. En 1999 logró ser violado y por ello lo convierte en un método de encriptación poco seguro [19] [20]
- 3DES (Three DES o Triple DES): aplica tres veces el proceso DES con tres llaves diferentes de 56 bits. La importancia de esto es que si alguien puede descifrar una llave, es casi imposible poder descifrar las tres y utilizarlas en el orden adecuado. Hoy en día es uno de los algoritmos simétricos más seguros [23] [24]
- IDEA (International Data Encryption Algorithm): trabaja con llaves de 128 bits. Realiza procesos de shift y copiado y pegado de los 128 bits, dejando un total de 52 sub llaves de 16 bits cada una. Aún no es aceptado como un estándar, aunque no se le han encontrado debilidades aún [23] [24]
- AES (Advanced Encryption Standard): aún no es un estándar, pero es de amplia aceptación a nivel mundial. Es uno de los más seguros [23] [24]
- RC5: este sistema es el sucesor de RC4, que consistía en hacer un XOR al mensaje con un vector que se supone aleatorio y que se desprende de la clave, mientras que RC5 usa otra operación, llamada dependencia de datos, que aplica sifts a los datos para obtener así el mensaje cifrado [23] [24]

- Blowfish: este algoritmo fue diseñado por Bruce Schneier y hace un cifrado por bloques con claves de longitud variable desde 32 a 448 bits (en múltiplos de 8). La implementación de este algoritmo es de dominio pública [23] [24].

4.3 Utilización de encriptación

Alternativamente, al momento de implementar el proceso de encriptación de los datos que maneja la componente se podría optar por la encriptación asimétrica o de clave pública. Esto involucrará que se utilice una clave pública para realizar el cifrado y un clave privada para poder descifrar los datos cifrados primeramente con la clave pública. La clave pública podrá ser distribuida libremente, pero la clave privada deberá ser almacenada en el servidor y protegida adecuadamente. En este tipo de esquema el método más utilizado es RSA [23] [24].

4.4 Utilización de encriptación seleccionada por el desarrollador

Como última opción, se podría delegar al desarrollador la selección del método de encriptación, siendo este asimétrico o simétrico y que la componente utilice alguno de los algoritmos mencionados anteriormente para lograr el tipo de cifrado seleccionado. Otra solución sería que se ingrese el algoritmo de encriptación elegido, de un conjunto de algoritmos que entienda la componente. De esta manera se obtiene una componente más flexible y posibilita al desarrollador tener un rol más activo en la definición de la seguridad de la aplicación.

4.5 Implementación de las Componentes Seguras

Al momento de implementar cada uno de los componente de la librería se define un conjunto de atributos que el desarrollador debería especificar o no para su correcto uso. Algunos atributos cifran los valores que se le ingresa y otros muestran los valores ingresados en texto claro. Además, se definen los renderers (uno o varios) que resuelven el mostrado de la componente, si esta se debe mostrar con distintos tipos de lenguajes como por ejemplo HTML, XHTML [11] o WML [12] (estos dos últimos muy utilizados en plataformas móviles) entonces se definirá un renderer por cada tipo de lenguaje. Alternativamente se podría delegar el renderizado a la propia componente, con lo cual solamente se podrá utilizar con un solo tipo de lenguaje para el mostrado de la misma. En el momento de codificar el o los renderers, es necesario analizar las necesidades de encriptación por renderer ya que cada renderer podría estar orientado a una plataforma cliente en particular con necesidades de seguridad diferentes. Los diferentes tipos de implementaciones criptográficas y las elecciones de los algoritmos que los implementan determinarán el nivel de seguridad que se proveerá a la información. Es por ello que esta decisión podría ser buena idea delegarla al desarrollador (Ver secciones 4.2, 4.3 y 4.4).

Retomando el escenario planteado en 4.1, ahora tenemos definida una librería de componente segura con las cualidades descriptas anteriormente. Dentro de dicha

librería se encuentra definida una componente segura que se llama InputSecure la que utilizaremos para resolver la problemática de exponer información privilegiada.

La componente InputSecure toma dos atributos que son el identificador y el nombre, los cuales guardan información para realizar la búsqueda sobre la tabla persona y por ello dicha información se cifrará al instante de ser evaluada la componente. La utilización de la componente y la definición de los valores de los atributos antes mencionados se muestran en el Extracto 3.

Extracto 3: Código JSF donde se muestra la utilización de la componente segura InputSecure.

```
<input:InputSecure ident="nombre" name="nombre"/>
<input:InputSecure ident="apellido" name="apellido"/>
<input:InputSecure ident="edad" name="edad"/>
<input:InputSecure ident="documento" name="documento"/>
```

Ahora bien, cuando todas las InputSecure sean evaluadas y dicho resultado sean retornados al cliente, los valores especificados en los atributos name e ident estarán cifrados logrando el objetivo de proteger la información sensible. Extracto 4 grafica lo antes mencionado.

Extracto 4: Código HTML que resulta una vez que se evaluaron las componentes seguras InputSecure. Notar que los identificadores y los nombres de los campos de búsqueda se muestran cifrados.

```
<input type="text" name="vRr2qJO7oHg="
id="iZXE/Tfp0OU=" />
<input type="text" name="B4mM8RCJ9mYpGMZgFAumoQ=="
id="aKiUmSj1DS5ffftO1aTgnw==" />
<input type="text" name="UBfrhXepzwA="
id="idNk1EJ2/NQ=" />
<input type="text" name="Vf9eWMOAOxFrvr8MQ1nshQ=="
id="5YGmOALH0EIofDL7WIKfXA==" />
```

5 Conclusiones

Debido a la incesante aparición de nuevas tecnologías y frameworks de desarrollo Web en Java, estos han enfocado sus esfuerzos en simplificar la construcción de aplicaciones ricas y mucho más interactivas, relegando a un segundo plano los aspectos básicos de seguridad. Y Java Server Faces no ha sido la excepción, es por ello se pueden encontrar tantas aplicaciones Faces vulnerables a distintos tipos de ataques en la Web.

Es por ello que al utilizar una estrategia de mitigación de vulnerabilidades Web en aplicaciones JSF mediante la construcción de componentes Faces seguras, se logra que la solución sea centralizada, extensible, distribuible y reutilizable. Todos estos aspectos son altamente deseables en cualquier tipo de solución de software. Por otro

lado, simplifica la construcción de aplicaciones Web ya que el desarrollador podría utilizar las componentes seguras con cierto nivel de complejidad fácilmente, sin tener que construirla él mismo o construyéndola una vez y luego reutilizarla en los desarrollos futuros.

Referencias Bibliográficas

1. JavaServer Faces Technology, http://java.sun.com/javaee/javaserverfaces/
2. JSRs: JAVA Specification Requests. JSR 252: JAVAServer Faces 1.2, http://jcp.org/en/jsr/detail?id=252
3. Giulio Zambon with Michael Sekler. Beginning JSP,™ JSF,™ and Tomcat™ Web Development
4. Java SE Desktop Overview, http://java.sun.com/javase/technologies/desktop/
5. The Open Web Application Security Project, http://www.owasp.org/
6. J2EE, http://java.sun.com/javaee/technologies/
7. Introducción a JSF Java Server Faces, http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=IntroduccionJSFJava
8. Getting Started with Visual Basic, http://msdn.microsoft.com/en-us/vbasic/bb466159.aspx
9. PowerBuilder - 4GL RAD Application Development and Design Software Solution - Sybase Inc., http://www.sybase.com/products/modelingdevelopment/powerbuilder
10. RAD Application Development Software | Delphi from Embarcadero Technologies, http://www.embarcadero.com/products/delphi/
11. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition), http://www.w3.org/TR/xhtml1/
12. Wireless Markup Language, http://es.wikipedia.org/wiki/Wireless_Markup_Language
13. Model 2, http://en.wikipedia.org/wiki/Model_2
14. Understanding JavaServer Pages Model 2 architecture, http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html
15. Kito D. Mann. JAVAServer Faces in Action
16. Enterprise JAVA Community: Are JAVA Web Applications Secure?, http://www.theserverside.com/tt/articles/article.tss?l=AreJAVAWebApplicationsSecure
17. Designing Enterprise Applications with the J2EETM Platform, Second Edition., http://JAVA.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/security/security5.html
18. JAVATM Authentication and Authorization Service (JAAS) Reference Guide for the J2SE Development Kit 5.0., http://JAVA.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html
19. JAVATM Cryptography Architecture API Specification & Reference, http://JAVA.sun.com/j2se/1.5/docs/guide/security/CryptoSpec.html
20. JAVATM Cryptography Extension (JCE) Reference Guide for the JAVATM 2 Platform Standard Edition Development Kit (JDK) 5.0, http://JAVA.sun.com/j2se/1.5.0/docs/guide/security/jce/JCERefGuide.html
21. Las 10 vulnerabilidades de seguridad más críticas en aplicaciones Web. Versión 2007 en español, http://www.owasp.org/index.php/Diez_Mayores_2007
22. Bellante y Chanfreau, Tesis de grado “Librería genérica de aspectos para tratar las vulnerabilidades de seguridad mas criticas en aplicaciones Web”.
23. Encriptación, http://www.textoscientificos.com/redes/redes-virtuales/tuneles/encriptacion
24. Java Security Packages using JCA/JCE, http://www.javabeat.net/tips/60-java-security-packages-using-jcajce.html