Incorporando seguridad a las componentes de Interfaz de Usuario del framework JSF (JAVA Server Faces) con soporte para clientes heterogéneos.

Autor:Pablo José Iuliano[1]

Directores: Francisco Javier Díaz[1], Claudia Queiruga[1].

{piuliano,jdiaz,cqueiruga}@info.unlp.edu.ar

Carrera de Grado: Licenciatura en Informática.

[1] LINTI (Laboratorio de Investigación en Nuevas Tecnologías Informáticas), Facultad de Informática, UNLP. La Plata, Argentina

Abstract. Este trabajo propone una solución open source, llamada TouchFacesSecure, a la falta de seguridad en las aplicaciones Faces [1]. TouchFacesSecure es una extensión del módulo Mobile TouchFaces [2] de la distribución libre PrimeFaces [3] que provee mecanismos de seguridad para la prevención de posibles ataques contra la aplicación.

Keywords. faces; seguridad; componentes de interfaz de usuario; web móvil; open source; software libre

1 Introducción

Actualmente se está transitando una fase migratoria en cuanto a la tendencia tecnológica predominante, esta migración va del desktop con acceso a la Web al Web mobile. Se predice que en los próximos 5 años habrá más usuarios conectados a Internet a través de dispositivos móviles que con PCs tradicionales [4]. La siguiente figura grafica las proyecciones que se manejan en cuanto a la migración del Web tradicional al Web mobile:

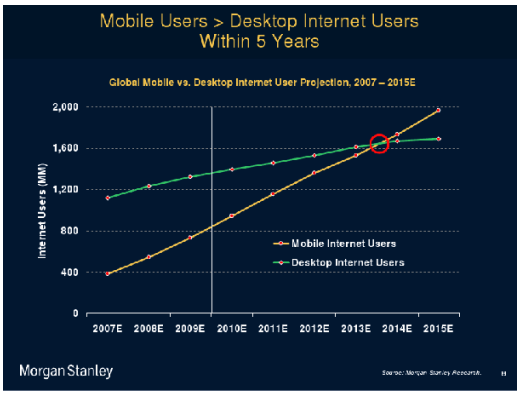

Fig. 1. : Proyección de usuarios de Internet *mobile* vs. los *desktop*. Extraído de *"Mary Meeker: Mobile Internet Will Soon Overtake Fixed Internet"* [4].

En este sentido, se proyecta que para el 2011 las descarga de aplicaciones móviles escalen hasta llegar a los 9.88 millones y 16.2 millones para el 2013 [5]. La Figura 2, grafica la cantidad de descargas de este tipo de aplicaciones que hubo en los años 2008, 2009, 2010 y las proyecciones que se esperan los años venideros:

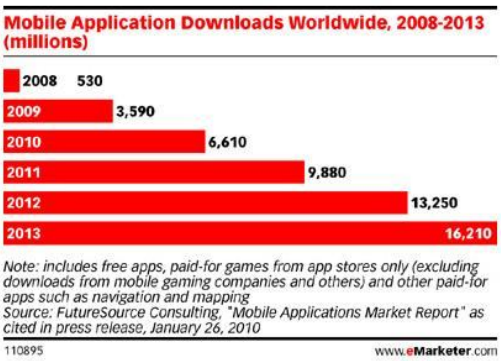

Fig. 2. Descargas de aplicaciones *mobile*. Extraído de *"Chart of mobile application downloads"* [5].

Debido a la gran proliferación de las aplicaciones móviles y su gran disponibilidad hacen que los riesgos de seguridad se incrementen en forma considerable. Muchos de estos programas manejan información personal del usuario; así un ataque exitoso que explota una vulnerabilidad de seguridad podría tener como consecuencia que el atacante se haga de todos los datos personales del usuario del dispositivo. Cabe mencionar que toda la información a la cual tendría acceso el atacante no sería la de la organización,

como sucedía en los 90s, sino que es la información personal del usuario móvil. Esta condición única de los entornos móviles hace que los riesgos de seguridad tomen una considerable, muy preocupante y nueva dimensión.

JAVA Server Faces (JSF o Faces) [1] es el framework estándar para construcción de aplicaciones Web en JAVA incorporado en la distribución JEE de Sun [6]. Sin embargo, el tratamiento de la seguridad en las aplicaciones JSF, no es una de sus fortalezas.

Este trabajo intenta abordar la problemática del tratamiento de la seguridad de las aplicaciones web convencionales y móviles desarrolladas en Faces.

2 JAVA Server Faces

JAVA Server Faces (JSF o Faces) [1] es un framework para la construcción de aplicaciones Web escritas en JAVA. JSF fue desarrollado por la JAVA Community Process (JCP, JSR 252) y forma parte de la especificación de JAVA Enterprise Edition (JEE) a partir de la versión 5.0.

2.1 Fundamentos de JAVA Server Faces

JAVA Server Faces provee un conjunto de componentes de interfaz de usuario predefinidos (botones, hipervínculos, checkboxes, etc.), un modelo para la creación de componentes customizados y la posibilidad de procesar en el servidor los eventos generados en el cliente.

Faces introduce la funcionalidad de programar librerías de componentes de interfaz de usuario propias. Esta es una forma concreta y muy potente de lograr código reusable de interfaz de usuario basado en web.

La arquitectura extensible de **Faces** permite que se puedan incorporar nuevas componentes con sus correspondientes *renderers*, validadores y conversores. Los *renderers* son entidades desacopladas de las componentes que resuelven la vista en dispositivos específicos bajo la consigna de "pantalla neutral". Los validadores y conversores proveen saneamiento de datos y conversión de tipos respectivamente.

2.2 Distribuciones Faces para clientes web

Actualmente existe un conjunto de distintas librerías open source implementadas en JSF que tienen como objetivo proporcionar soluciones web para clientes heterogéneos. A continuación se detallarán las distintas iniciativas:

- **Ericsson Mobile JSF Kit - Java.net Mobile JSF**: El kit de desarrollo Mobile JSF, comenzado por Ericsson [6] y continuado por

java.net [7] tiene como objetivo permitir que las aplicaciones que se dibujaran de distinta forma según el cliente.

- **Apache MyFaces Trinidad:** Apache MyFaces Trinidad [8] es una librería JSF que incluye una gran cantidad de componentes de calidad Enterprise con soporte para clientes móviles. El dibujado de los componentes es similar a la provista/implementada por Mobile JSF.
- **Mobile TouchFaces de PrimeFaces**: PrimeFaces [3] es una librería open source cuyo objetivo principal es ofrecer un conjunto de componentes ricos para web. PrimeFaces [3] posee un módulo llamado Mobile TouchFaces [2], el cual contiene un conjunto de componentes para crear aplicaciones para clientes móviles. Uno de los objetivos de TouchFaces es ser una suite de componentes ligera, que sea compatible con otras suites de componentes JSF ya existentes y que el código Javascript que genera no sea intrusivo.

3 Seguridad en Aplicaciones Web

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 [10].

El proyecto OWASP (Open Web Application Security Project) [10] 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 ha publicado un listado con las diez vulnerabilidades más comunes y críticas encontradas en los desarrollos web. Este listado se utilizó para seleccionar las vulnerabilidades que están dentro del ámbito de interés de este trabajo:

- Secuencia de Comandos en Sitios Cruzados (XSS).
- Fallas de Inyección.
- Vulnerabilidades de Falsificación de Petición en Sitios Cruzados (CSRF).
- Revelación de Información y Gestión Incorrecta de Errores.

4 Pruebas de vulnerabilidades de seguridad sobre una aplicación JSF open-source construida con PrimeFaces

A fin de analizar la efectividad de la solución que propone este trabajo, se utilizó como base la aplicación prime-phonebook que se encuentra publicada en el sitio oficial de la distribución PrimeFaces [3]. Se adaptó la versión

original para que funcionara también en ambientes móviles, ya que sólo funcionaba con clientes web tradicionales.

4.1 Adaptación y arquitectura de prime-phonebook-mobile

El proceso de adaptación comenzó por reemplazar los componentes para clientes web tradicionales por los del módulo Mobile TouchFaces [2]. El resultado de este rediseño es una aplicación que soporta tanto clientes web tradicionales como móviles, a la que denominaremos "prime-phonebook-mobile". En la siguiente figura se muestra la arquitectura resultante de la modificación de la aplicación original.

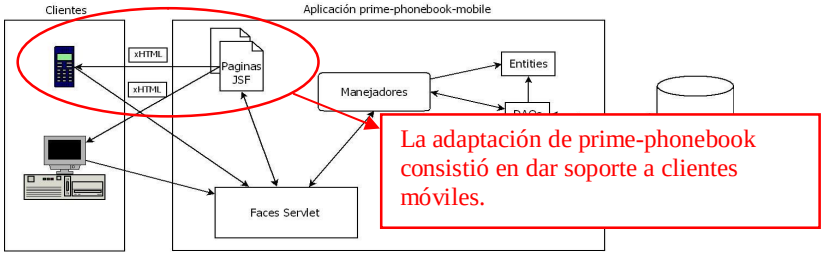

Fig. 3. Arquitectura de prime-phonebook-mobile

4.2 Ataques contra prime-phonebook-mobile

La aplicación prime-phonebook-mobile es particularmente vulnerable a un ataque de XSS. Al analizar cómo se muestra la información en el sistema, queda de manifiesto que los parámetros que se ingresan se muestran en la página de respuesta tal cual fueron ingresados. Con lo cual si se inyecta código a través de los mismos, éste se ejecutará luego de recibir la página desde el servidor. Esto último es ilustrado a continuación.

- Si se ingresa en el parámetro *firstname* el código Javascript que se muestra a continuación, el resultado es la exposición del identificador de la sesión del usuario (almacenado en una cookie) y de esta manera un atacante podría hacerse pasar por el usuario validado consistiendo en un robo de identidad. Además de la posibilidad de manipular la información sensible que el dueño de la sesión tiene almacenada en la aplicación:

```
<script>document.getElementById("content").innerHTML =
document.cookie</script>
```

Fig. 4. Ingreso de código XSS.

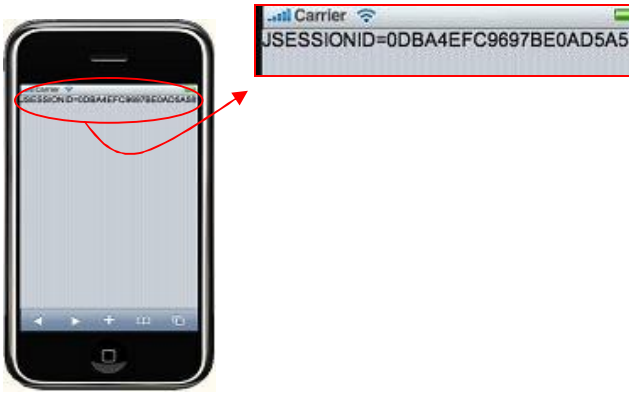

Fig. 5. Resultado de la ejecución

Es posible también realizar de manera exitosa un ataque de inyección SQL a prime-phonebook-mobile, por la falta de validación de los parámetros de entrada como antes se mencionó. Un ejemplo de esto último se presenta en la eliminación de un contacto de la aplicación de estudio, ya que es posible inyectar una cadena maliciosa de tal manera que cuando se atienda el requerimiento se ejecute el código malicioso. El efecto del ataque anterior es la eliminación de todos los contactos del usuario. Las Figuras 6, 7 y 8, muestran cómo se lleva a cabo este proceso.

Fig. 6. Estado inicial.

Fig. 7. Inyección de código

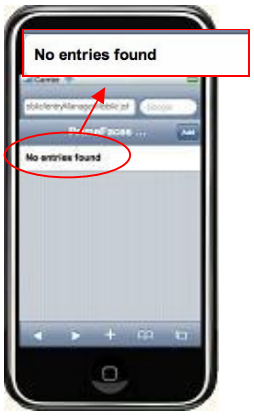

Fig. 8. Estado Resultante.

prime-phonebook-mobile no valida que los requerimientos sean generados a través de sus páginas. Esto se debe a que las peticiones autorizadas están basadas únicamente en credenciales que son automáticamente presentadas como la cookie de sesión si el usuario está actualmente conectado a la aplicación.

La falsificación de petición en sitios cruzados permite a un atacante realizar acciones sobre la aplicación de forma indirecta a través del engaño del usuario validado. Las Figuras 9, 10, 11 y 12, muestran que la concreción de un ataque de CSRF sobre la aplicación de estudio.

Fig. 9. Aplicación en el estado inicial.

Fig. 10. Aplicación que contiene el link de CSRF.

Fig. 11. Resultado del *link* malicioso.

Fig. 12. Estado final de la aplicación.

Al generarse un requerimiento sobre prime-phonebook-mobile, toda la información sensible queda expuesta en el código retornado por el servidor. Así con sólo inspeccionar la página resultado se obtiene fácilmente la información relacionada con la base de datos, con lo cual constituye una grave falla en cuanto a la confidencialidad de los datos. La Figura 13 grafica lo antes mencionado.

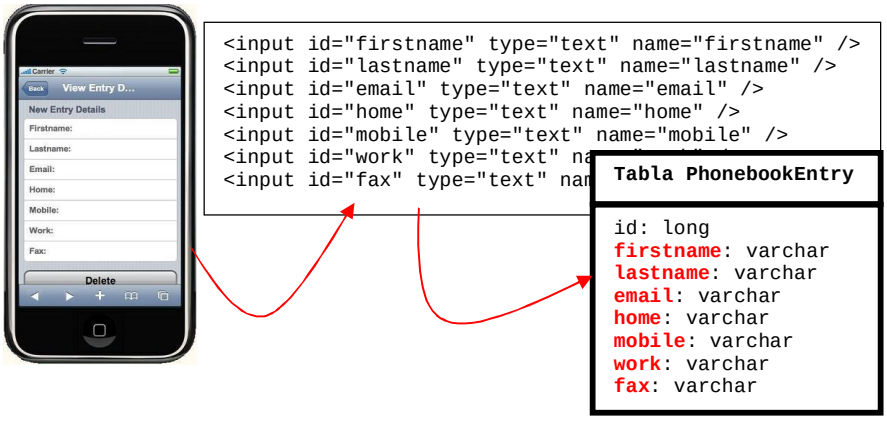

Fig. 13. Deducción de la estructura de la base de datos a través del código retornado.

5 TouchFacesSecure: una extensión de TouchFaces con soporte para seguridad

Como mencionamos anteriormente, la construcción de una librería de extensión de Mobile TouchFaces [2] que encapsule en sus componentes mecanismos de securitización, la cual denominaremos TouchFacesSecure, es la estrategia que este trabajo propone para proveer seguridad a las aplicaciones **Faces**.

5.1 Librerías de soporte utilizadas en la construcción de TouchFacesSecures

En esta sección se describirán las tecnologías open source utilizadas como soporte para la construcción de los componentes seguros. Estas se detallan a continuación:

- **Bouncy Castle**: Es una implementación JAVA de algoritmos criptográficos desarrollados por la organización Legion of the Bouncy Castle [11], la cual está organizada como una API liviana adecuada para el uso en cualquier ambiente web.
- **ValidatingHttpRequest de OWASP**: Es la solución propuesta por OWASP [10] para la problemática de la entrada no validada. La clase ValidatingHttpRequest [12] realiza el proceso de validación de la entrada de datos, el cual dispara una excepción si dicho proceso falla.

5.2 Arquitectura de TouchFacesSecure

La Figura 14 ilustra la arquitectura de TouchFacesSecure, la interacción de sus componentes principales y detalla los módulos que conforman una aplicación **Faces** construida a partir de TouchFacesSecure.

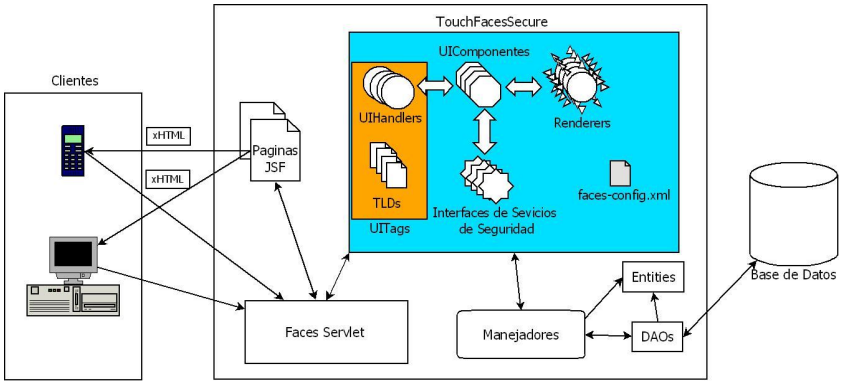

Fig. 14. Arquitectura de TouchFacesSecure.

5.3 La estrategia de solución de TouchFacesSecure

A continuación se describirá en detalle la funcionalidad y las características más importantes de los componentes que conforman la librería.

Application Seguro.
Hay ciertos tipos de scripts que se ejecutan del lado del cliente, éstos acceden y manipulan información de la sesión del usuario que se encuentra almacenada en la cookie. Esta situación puede conducir a que se realicen eventuales ataques de XSS y por ello se debe prevenir. Sumado al hecho que Application es el contenedor de todos los componentes que conforman las páginas JSF, se decidio reimplementar Application para incorporar un mecanismo que prevenga la ejecución de estos tipos de *scripts*.

Implementación.
El componente Application Seguro asigna el atributo HttpOnly en el header del response HTTP. El atributo antes mencionado tiene por finalidad prevenir que la información de la cookie sea accedida y manipulada por scripts que ejecutan del lado del cliente.

InputText Seguro.
El componente estándar InputText es susceptible a eventuales ataques de XSS y de inyección de SQL como se demostró anteriormente. Esta situación se presenta debido a que InputText no posee ningún tipo de validación por defecto que tome en cuenta el ingreso de datos pensados para intentar concretar los dos ataques antes mencionados. Otro problema observado es que este componente estándar no posee ningún mecanismo de ocultamiento de información para los datos que ingresa el desarrollador en ciertos atributos críticos del tag, como por ejemplo el *name* cuando se está confeccionando la página en la cual estará inserto.

Implementación.
El desarrollador al utilizar InputText Seguro especifica los atributos propios de cualquier componente InputText estándar al momento del diseño de la pagina JSF. Antes de ser enviada al cliente, la información crítica que especificó el desarrollador se cifra.

El cifrado se aplica sobre el atributo *name* del tag, ya que dependiendo de su contenido podría dar lugar a una eventual deducción parcial de la estructura de la base de datos. Esta última condición se da frecuentemente debido a que los desarrolladores son muy propensos a incluir información de las estructuras de las tablas de la base de datos que son manipuladas en las páginas donde se ubican estos componentes.

La manera en que InputText Seguro resuelve el cifrado de la información es a través de clases que implementan la interfase Encryptor. A continuación se muestra la definición de la interfase Encryptor:

```
package tesis.tool.seguridad.encryptionToolkit;
public interface Encryptor {
  public byte[] encrypt(byte[] msg);
  public java.lang.String encrypt(String msg);
  public byte[] decrypt(byte[] msg);
  public java.lang.String decrypt(String msg);
}
```

La clase del componente InputText Seguro provee confidencialidad de datos de dos modos diferentes:

- Por defecto: El componente provee un Encryptor por defecto. Éste utiliza la API de Bouncy Castle [10] y en particular un algoritmo RSA de clave pública - clave privada.
- Especificada por el usuario: El desarrollador tiene la posibilidad especificar el algoritmo de encriptación a utilizar, creando o utilizando una clase que implementa Encryptor que usa el algoritmo de cifrado deseado.

La segunda medida de seguridad implementada busca evitar la concreción de ataques XSS y de inyección de SQL. Esto se logra mediante la utilización de una clase llamada XSSSQLInjectionToolkit, que es una adaptación de la clase ValidatingHttpRequest [12] publicada en el sitio de OWASP [10].

El componente InputText Seguro permite que el usuario especifique el mensaje de error a desplegar cuando se detecta una entrada inválida. Para esto último, se define un atributo llamado xssSqlInjectionDetectedMsg donde se proporciona el texto del mensaje de error a mostrar.

Otro mecanismo flexible que se incorpora es la posibilidad de asignar un *handler* para las ocasiones en que se registran excepciones de seguridad web. Estas excepciones son disparadas cuando se detecta, por ejemplo, una entrada de datos que intenta efectuar un XSS o una inyección de SQL. Los *handlers* manejan excepciones del tipo WebSecurityException y para que una clase pueda ser considerada un *handler* debe implementar la interfase WebSecurityExceptionHandler. Esta interfaz define un sólo método llamado exceptionThrowed(), el cual es utilizado para informar que una situación anómala en cuanto a la seguridad de la aplicación se ha producido. Esto permite que el usuario tenga control sobre dichas situaciones y tomar alguna acción en consecuencia.

Label Seguro.

Las etiquetas que son utilizadas en la aplicación de estudio, mantienen una relación con los componentes para las cuales son especificadas. Esta relación se alcanza a través del atributo *for*, en éste se especifica el nombre del componente al cual se asociará. De esta forma, la confidencialidad en la aplicación está comprometida, ya que cuando se dibujan las etiquetas, los nombres de los componentes que tienen asociados se revelan.

Implementación.

Label Seguro utiliza mecanismos de confidencialidad análogos a InputText Seguro. Así, el componente puede usar una clase particular que implemente Encryptor o dejar que el componente utilice un Encryptor por default para ocultar la información crítica.

CommandLink y CommandButton Seguros.

Los componentes diseñados para disparar acciones en una aplicación PrimeFaces han sido seleccionados como punto de validación para evitar ataques del tipo CSRF. Esta decisión se sustenta en el hecho que tanto CommandLink como CommandButton son los responsables de propagar los eventos a los listeners que los atienden. Por esta razón y por la naturaleza propia de CSRF resulta adecuado situar la validación en este tipo de componentes.

Implementación.
El desarrollador al utilizar CommandLink Seguro o CommandButton Seguro para generar acciones notará que al inspeccionar el código de una página en donde se usan, además de dibujarse el button o link, también se generó un token aleatorio el cual es almacenado en el cliente para su posterior verificación.

Al igual que InputText Seguro, estos *commands* permiten personalizar el la generación de tokens, los mensaje de error y los *handler*s que atenderán las excepciones de seguridad web. Así mismo se provee una implementación por defecto para todos los anteriores.

La generación de tokens aleatorios en estos componentes está pensada de tal manera que las clases encargadas de esta tarea sean intercambiables entre sí. Sólo deben cumplir la condición de implementar la interfase CSRFTokenManageable.

La primera acción que se produce cuando el cliente web activa un evento con alguno de estos dos componentes, es la validación de los tokens. Esta validación constata que el token enviado por el cliente coincida con el que está almacenado en el servidor. De esta manera se garantiza que todas las acciones realizadas provengan de la aplicación y no como resultado de un ataque de CSRF exitoso.

5.4 Modo de utilización de TouchFacesSecure

La librería de componentes seguros presentada persigue dos objetivos adicionales, los cuales se presentan a continuación:

Enfoque 1: Fácil migración.
Se parte de la existencia de una aplicación JSF que no trata fallas de seguridad y se tiene como objetivo lograr una aplicación que traten dichas vulnerabilidades, en el menor tiempo posible, con el menor costo y de forma razonablemente sencilla (estas dos últimas valoraciones referidas a la recodificación de lo ya implementado). Los pasos a seguir para alcanzar este objetivo son los siguientes:

1. Copiar en el directorio WEB-INF/lib de la aplicación el archivo correspondiente a la librería de componentes seguros.
2. Cargar la librería de en las páginas donde se utilizarán los componentes.
3. Reemplazar los prefijos de los componentes estándares por el prefijo de los que son implementados en la librería segura.

Enfoque 2: Funcionalidades Personalizables Seguras.
Este enfoque permite obtener una aplicación segura y flexible, entendiéndose por flexibilidad la facilidad para implementar notificaciones al usuario y mecanismos de seguridad intercambiables. Los pasos 1, 2 y 3 son los mismos que en el Enfoque 1, además se deben realizar las siguientes acciones:

4. Establecer qué valores específicos deben tomar los atributos: `encryptor`, `xssSqlInjectionDetectedMsg`, `tokenManager`, `csrfDetectedMsg` y `webSecurityExceptionHandler`.
5. El valor del último atributo del punto 4 debe implementar la interfase WebSecurityExceptionHandler para que los componentes tengan un medio para avisarle que la aplicación está siendo atacada.
6. Incorporar un mecanismo de *polling* que permita notificar al usuario cuáles fueron los ataques que registró la aplicación.

Se decidió mostrar el uso de los componentes bajo estos dos enfoques, modificando la aplicación de estudio para alcanzar dicho propósito.

6 Tratamiento de ataques en las aplicaciones securitizadas con TouchFacesSecure

En esta sección se presentan las dos aplicaciones producto de la reimplementación de prime-phonebook-mobile con las librerías de extensión TouchFacesSecure. La primera de las dos aplicaciones, llamada prime-phonebook-mobile-segura-facil-migracion, fue codificada siguiendo el Enfoque 1 explicado anteriormente. Mientras que la segunda, denominada prime-phonebook-mobile-segura-funcional-personalizable, se implementó siguiendo el Enfoque 2. Tanto prime-phonebook-mobile-segura-facil-migracion como prime-phonebook-mobile-segura-funcional-personalizable serán sometidas a los mismos ataques que fueron realizados a prime-phonebook-mobile y luego se presentarán los resultados registrados de cada unos de ellos.

Las fallas de XSS e inyección SQL están presentes en prime-phonebook-mobile. Si se vuelven a ejecutar los dos ataques anteriores, pero esta vez en las aplicaciones aseguradas a través del uso de la librería de componentes seguros, el resultado se muestra gráficamente en las siguientes figuras:

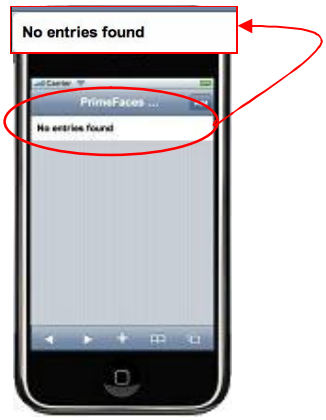

Fig. 15. Estado final de prime-phonebook-mobile-segura-facil-migracion.

Fig. 16. Notificación del intento de ataque XSS a prime-phonebook-mobile-segura-funcional-personalizable.

Otra manera de vulnerar la aplicación **Faces** de estudio original, es realizar una falsificación de petición en sitios cruzados. Pero si volvemos a intentar el mismo método de ataque, esta vez contra prime-phonebook-mobile-segura-facil-migracion y prime-phonebook-mobile-segura-funcional-personalizable el resultado es notablemente diferente al mostrado anteriormente. Las siguientes imágenes grafican lo antes mencionado:

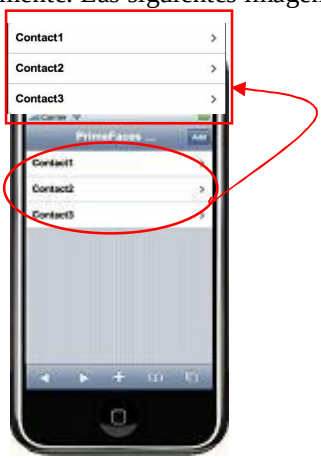

Fig. 17. Estado final de prime-phonebook-mobile-segura-facil-migracion luego del intento de ataque CSRF.

Fig. 18. Notificación de prime-phonebook-mobile-segura-funcional-personalizable luego del intento de ataque CSRF.

prime-phonebook-mobile hace un manejo deficitario de la confidencialidad de los datos sensibles. En TouchFacesSecure al tratar de recabar información mediante la inspección del código retornado en las soluciones securitizadas, se obtiene lo siguiente:

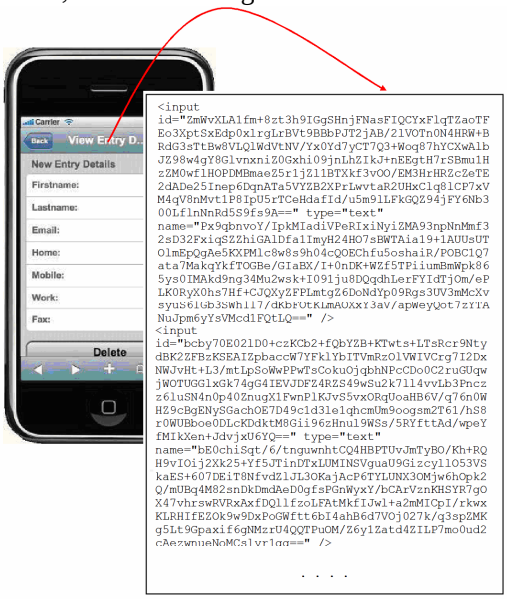

Fig. 19. Inspección del código **HTML** con la información sensible encriptada.

Es evidente que la información útil que se puede recabar es nula.

7 Conclusiones

Debido a la incesante aparición de nuevas tecnologías y frameworks de desarrollo web en JAVA, éstos 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; por su parte JAVA Server Faces [1] no ha sido la excepción, es por ello que se pueden encontrar tantas aplicaciones **Faces** vulnerables a distintos tipos de ataques en la web.

La solución propuesta consiste en generar una librería de componentes seguros open source denominada TouchFacesSecure, que al ser utilizada por nuestra aplicación de estudio sana (o al menos mitiga) las vulnerabilidades que estén dentro de nuestro interés asegurando de este modo la aplicación.

Se comprobó que los componentes que integran la librería tienen dos objetivos estructurales adicionales al de proveer seguridad a una aplicación

Faces, los cuales son: alcanzar una migración de una aplicación JSF estándar a una segura con el mínimo costo y proveer flexibilidad en los mecanismos de seguridad utilizados. El primero de ellos logrado al respetar los nombres de los tags de los componentes en conjunción con los nombres, tipos y cantidad de atributos que cada uno de los componentes estándar define. El segundo también se alcanzó, esta vez utilizando un conjunto de interfases que proveyeron la abstracción necesaria para que las implementaciones de los mecanismos de seguridad que utilizan los componentes fueran fácilmente intercambiables entre sí. Esta íntercambiabilidad de los mecanismos de seguridad impacta directamente en la robustez y adaptabilidad de la solución propuesta, lográndose disponer de un medio para afrontar las cuestiones críticas de seguridad como lo son la obsolescencia de los algoritmos criptográficos y los riesgos computacionales asociados a los mismos.

Utilizar una estrategia de mitigación de vulnerabilidades Web en aplicaciones JSF mediante la construcción de componentes Faces seguras logra que la solución sea centralizada, extensible, distribuible y reutilizable. Todos estos aspectos son altamente deseables en cualquier tipo de solución de software.

8 Referencias Bibliográficas

1. JSF Technology: http://java.sun.com/javaee/javaserverfaces/
2. TouchFaces:http://97.107.138.40:8080/prime-showcase/touch/index.jsf
3. PrimeFaces Suite: http://www.primefaces.org/
4. Mary Meeker: Mobile Internet Will Soon Overtake Fixed Internet, http://gigaom.com/2010/04/12/mary-meeker-mobile-internet-will-soon-overtake-fixed-internet/
5. Chart of mobile application downloads, http://fonegigsblog.com/2010/04/07/chart-of-mobile-application-downloads/
6. JEE, http://java.sun.com/javaee/technologies/.
7. Ericsson http://www.ericsson.com/
8. Java.net Java Communications Community http://java.net/
9. Apache MyFaces Trinidad http://myfaces.apache.org/trinidad/index.html
10. The Open Web Application Security Project, http://www.owasp.org/
11. Legion of the Bouncy Castle http://www.bouncycastle.org/
12. How to add validation logic to HttpServletRequest http://www.owasp.org/index.php/How_to_add_validation_logic_to_HttpServletRequest