ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/318416819

Implementation of techniques and OWASP security recommendations to avoid SQL and XSS attacks using J2EE and WS-Security

Conference Paper · June 2017

DOI: 10.23919/CISTI.2017.7975981

CITATIONS
0

READS
163

4 authors, including:

Daniel Guaman
Universidad Técnica Particular de Loja
8 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Danilo Rubén Jaramillo
Universidad Técnica Particular de Loja
17 PUBLICATIONS **16** CITATIONS
SEE PROFILE

Manuel Sucunuta
Universidad Técnica Particular de Loja
5 PUBLICATIONS **6** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Software and education View project

Computer vision View project

All content following this page was uploaded by Daniel Guaman on 01 March 2018.

The user has requested enhancement of the downloaded file.

Implementación de técnicas y recomendaciones de seguridad OWASP para evitar ataques de tipo inyección SQL, XSS utilizando J2EE y WS-Security

Implementation of techniques and OWASP security recommendations to avoid SQL and XSS attacks using J2EE and WS-Security

Daniel Guamán, Franco Guamán, Danilo Jaramillo, Manuel Sucunuta
Universidad Técnica Particular de Loja, Departamento de Ciencias de la Computación y Electrónica
Loja, Ecuador
{daguaman, foguaman, djaramillo, mesucunuta}@utpl.edu.ec

Resumen- **El presente trabajo contiene la implementación de técnicas y recomendaciones OWASP sobre un prototipo SOA desarrollado con J2EE. Para su diseño y codificación se utilizan ciertas especificaciones WS-Security, framework Metro, MVC como patrón arquitectónico, Facade y DAO como patrones de diseño. El prototipo fue validado a nivel de diseño, codificación y seguridad a través de herramientas tales como: Structural Analysis for Java, SonarQube, OWASP ZAP, Vega y Wireshark. Con el desarrollo del prototipo se comprueba que el uso de normas, recomendaciones y técnicas para escritura de código seguro en aplicaciones de software son necesarias para evitar vulnerabilidades; además, el análisis estático apoya en la identificación de brechas de seguridad y aspectos de calidad que muchas de las veces no son consideradas por los desarrolladores.**

***Palabras clave*— Metro Framework, OWASP, SOA, SQL Injection, WS-Security, XSS.**

Abstract— **This work contains the implementation of techniques and recommendations OWASP on a SOA prototype developed with J2EE. To its design and coding we used some WS-Security specifications, Metro framework, MVC as architectural pattern, Facade and DAO as design patterns. The prototype was validated in terms of design, coding and security through some tools such as: Structural Analysis for Java, SonarQube, OWASP ZAP, Vega and Wireshark. With the development of this prototype we prove that use of standards, recommendations and techniques for writing secure code in software applications are necessary in order to prevent vulnerabilities; besides, the static analysis supports to identify security breaches and quality aspects that many times are not considered by developers.**

***Index Terms*— Metro Framework, OWASP, SOA, SQL Injection, WS-Security, XSS.**

I. INTRODUCCIÓN

Internet ofrece el ambiente ideal para el despliegue de aplicaciones web diseñadas y construidas con la finalidad de realizar actividades como intercambio, visualización y transacciones de datos e información; dichas aplicaciones se construyen tomando como referencia aspectos importantes del desarrollo de software como arquitectura, tecnologías, infraestructura, pero se descuidan aspectos de seguridad y calidad haciendo que las aplicaciones se vean amenazadas.

SOA (Service Oriented Architecture) permite integrar diversos tipos de aplicativos mediante servicios web, los cuales deben cumplir requisitos y estándares de seguridad que garanticen su despliegue. SOA ha adquirido gran importancia debido al cumplimiento de características como: flexibilidad, fiabilidad, interoperabilidad, portabilidad, seguridad, mantenibilidad, entre otros. Los ingenieros de software enfocan su trabajo en realizar un adecuado análisis y diseño de aplicaciones de software; sin embargo, en ciertas ocasiones dejan pasar por alto aspectos y recomendaciones de seguridad dados por OWASP, los mismos que se deben implementar a nivel de diseño, codificación o implementación y así reducir el riesgo de amenaza permanente y directa sobre este tipo de aplicaciones o servicios.

Existen algunos tipos de ataques que pueden ocurrir sobre aplicaciones o servicios expuestos en la web entre los que destacan Inyección SQL, Cross-site Scripting (XSS) los mismos que están catalogados como el primer y tercer tipo de ataques más comunes. El presente trabajo tiene como finalidad a través de la construcción de un prototipo, tomar como referencia las recomendaciones de OWASP y con ello validar que con su aplicación se puede minimizar vulnerabilidades y asegurar las aplicaciones web. Para la construcción del prototipo se utiliza JAVA EE, estilo arquitectónico SOA, especificaciones WS-Security, Framework Metro, patrón arquitectónico MVC, patrones de diseño Facade y DAO y herramientas automatizadas para realizar las pruebas y validación.

En la sección II del documento se aborda la base teórica necesaria para la construcción del prototipo. En la sección III se expone la metodología con la cual se diseñó y codificó el prototipo y finalmente en los apartados IV y V se exponen los resultados y conclusiones obtenidos del presente trabajo.

II.BASE TEÓRICA

A. Seguridad de software.- La seguridad contempla un alto grado de criticidad; pues, desde el punto de vista de software, los datos son un bien clave a nivel organizacional que deben ser asegurados desde todos los aspectos [1],[2]. Además, se consideran como un conjunto de recursos destinados a lograr que los activos de una organización sean confidenciales, íntegros, consistentes y disponibles a sus usuarios, autentificados por mecanismos de control de acceso y sujetos a auditora.

Existen tres aspectos a tomar en cuenta para el aseguramiento de la información: *1. Confidencialidad:* Asegura que la información no pueda estar disponible, ni descubierta por terceros. *2. Disponibilidad:* Garantiza la seguridad de la información, hardware y software, para ser accedido en cualquier momento. *3. Integridad:* Condición de seguridad que certifica que la información ha sido insertada, actualizada y eliminada por usuarios autorizados.

Según [3] existen implicaciones considerables dentro de una organización como producto de la falta de estandarización y uso de normativas de seguridad durante un proceso de desarrollo de software. Específicamente el aseguramiento en la actualidad no se planea, sino se ejecuta al ultimar detalles de implementación de software.

B. OWASP.- Es un proyecto de código abierto dedicado a estudiar y combatir vulnerabilidades de seguridad en software web [4]. OWASP gestiona varios proyectos de seguridad; entre los más importantes OWASP Top 10 [5]; el mismo contiene los 10 tipos de ataque más influyentes hacia la seguridad de aplicaciones. El objetivo principal de este proyecto es proveer técnicas básicas de prevención y protección de ataques informáticos a nivel de desarrollo de software.

C. WS-Security.- En los servicios web (WS) la autenticación y el nivel de seguridad del mensaje (message-level security) se pueden lograr a través del uso de WS-Security. La especificación WS-Security [6] define un conjunto de extensiones para servicios SOAP con el fin de proporcionar integridad, confidencialidad y disponibilidad de la información, mediante el cifrado, firmado y soportando varios formatos de tokens de seguridad. WS-Security es flexible y está diseñado para ser utilizado como base en la construcción de modelos de seguridad entre los que se incluye PKI, Kerberos, y SSL[7], con lo cual provee soporte para múltiples tokens de seguridad, dominios de confianza, formatos de firmado, y tecnologías de encriptación [8]. El objetivo de WS-Security es la seguridad a nivel de intercambio de mensajes SOAP messages y está sometido a la estandarización dada por OASIS Web Services Security Technical Committee. La especificación WS-Security define nuevas extensiones SOAP (message headers) para proporcionar autenticación mensaje a mensaje y prevenir la repetición de los mismos a través de marcas de tiempo [9], así como confidencialidad e integridad de extremo a extremo implementando XML Encryption y XMLDSig respectivamente en un ambiente de Web Services.

D. Recursos de implementación.- Dentro del ámbito de seguridad, para el presente trabajo se analizan técnicas y normativas de aseguramiento de software a nivel de diseño arquitectónico, código fuente y base de datos, las mismas que se aplican al prototipo. Con este propósito se identifican dos metodologías que permitirán el aseguramiento del prototipo; dichas metodologías son autenticación y cifrado de datos de hipertexto, JAAS [10] y HTTPS [11] respectivamente. Para el tema de codificación y base de datos se implementarán recomendaciones dadas por OWASP [5] con la finalidad de aplicar, por ejemplo, llamadas a procedimientos almacenados e implementación de consultas parametrizadas. Estas dos técnicas contribuyen potencialmente al blindaje de debilidades relacionadas con Inyección SQL [12]. Las llamadas a procedimientos almacenados corresponden a la forma de extracción de datos de la base de datos; estos son pequeños programas con lógica propia que interactúan directamente con el motor de base de datos, mientras que las consultas parametrizadas contribuyen a la separación de la cadena de consulta SQL del parámetro de especificación.

Para la construcción del prototipo se utiliza framework's y Api's que permiten una codificación segura.

- *Metro,* framework para la implementación de servicios web SOAP, posee una pila de tecnologías y proyectos como JAX-WS [13] y WSIT [14] que permiten la interoperabilidad y la implementación de WS seguros.
- *JAX-WS (Java API for XML Web Services),* API para la implementación de servicios web con el protocolo SOAP. JAX-WS facilita el desarrollo de WS ayudando a ocultar la complejidad de los mensajes SOAP.
- *WSIT (Web Service Interoperability Technologies), e*s la encargada de la interoperabilidad, uno de los principios de SOA. La seguridad soportada por WSIT es una implementación de OASIS (Reference architecture foundation for Service Oriented Architecture) para el aseguramiento de los mensajes SOAP.
- *Ws-Security,* estándar para implementar seguridad a WS SOAP, provee un modelo unificado para utilizar mecanismos de seguridad que permita autenticación, confidencialidad e integridad de los datos de los servicios web. Entre las especificaciones definidas dentro de WS-Security e implementadas en el prototipo constan:
- *SAML (Security Assertions Markup Language [15])*, especificación basada en XML para el intercambio de información mediante la autenticación, asegurando la integridad y confidencialidad de los datos a través de aserciones integradas del WS-Security Policy.
- WS-Security Policy [16]: Afirmaciones de políticas de seguridad basadas en el marco WS-Policy, donde se definen las políticas, limitaciones y requerimientos de seguridad en el documento WSDL del servicio web.

Con el objetivo de identificar, validar y cuantificar los resultados de aplicación de recomendaciones OWASP sobre el

prototipo, se utilizan herramientas y tecnologías que apoyan en esta actividad. Para validar la arquitectura del prototipo se utiliza:

- *Structural Analysis for JAVA* [17], para la validación arquitectónica de la solución se implementó algunas características de la herramienta seleccionada, la misma que realiza un escaneo profundo al archivo compilado (.jar) de la aplicación; adicionalmente, evalúa la arquitectura de software, codificación y su estabilidad.
- *Sonargraph Architect*, considerada herramienta de análisis estático que permite validar el modelo de arquitectura de software que puede ser verificada y ejecutada en modo autónomo o con plugins IDE de Eclipse o IntelliJ, aporta al usuario la capacidad de entender la estructura del sistema desarrollado, muestra la interacción que existe entre componentes, clases, y otros artefactos de programación que son parte del desarrollo de software.

Para la validación del código fuente se utiliza:

- *SonarQube,* es una plataforma de código abierto usada por los equipos de desarrollo para controlar la calidad del código [18]. Cubre siete ejes principales de calidad de software entre los que se encuentra la arquitectura y diseño de software; la validación de calidad se basa en un análisis donde se aplica el método SQALE [19].

Para validar aspectos de seguridad implementados en el prototipo se utiliza:

- *SoapUI* [20], [21] es una herramienta especializada en análisis de seguridad de servicios SOAP y REST; su funcionamiento involucra la explotación de distintas vulnerabilidades mediante la utilización de parámetros de entrada hacia los WS.

III.METODOLOGIA DE TRABAJO

Para el desarrollo del presente trabajo se procedió a la construcción de un prototipo basado en la arquitectura SOA y uso de Ws-Security, en donde se aplican recomendaciones dadas por OWASP, seguidamente se definió el modelo de datos e identificadores que representen los recursos a exponer. La implementación de los WS se limitó al verbo HTTP GET que especifica una acción de consulta y a implementaciones propias de JAVAEE [22]. Para las fases de implementación del patrón de diseño, control de aseguramiento lógico y vulnerabilidad se utilizó lo siguiente:

- Patrón de diseño arquitectónico: *Facade*, provee una lógica sencilla, lo que involucra que la programación sea estructurada y que permita la abstracción entre capas, de manera que si una de ellas requiere cambios, no sea necesario modificar el resto, con lo cual se aplica criterios de mantenibilidad y modificabilidad [23], [24]. *Dao*, es la encargada de controlar el acceso a los datos actuando de intermediario entre ésta y la aplicación además es utilizada para gestionar la persistencia de los datos ya que abstrae y encapsula el acceso a la base de datos.
- Control de seguridad lógica: *Autenticación*, se utilizó las características de HTTP y HTTPS. Los servicios web son expuesto en Internet para que sean consumidos desde cualquier dispositivo móvil o aplicación web, es por ello que se busca a través de este control garantizar el acceso a los datos. Además se implementó WSIT, el cual provee mecanismos de autenticación SAML *Authorization Vouches with Certificates*, con ello se resguarda la información mediante aserciones SAML con tokens de autenticación del cliente que está avalado con credenciales. El cifrado y firmado como parte del WSIT posee mecanismo de protección de información a nivel de operación y de mensajes donde se especifican las partes que se requieren proteger la confidencialidad (cifrado) y la integridad (firmado) del mensaje SOAP.
- Vulnerabilidad: *Inyección SQL y XSS*, corresponden al primer y tercer tipo de ataques más común a aplicaciones web según el Top Ten de OWASP. Además es importante detallar que no solo OWASP hace referencia a los ataques de inyección como los más comunes; si no, organizaciones como WASC (Web Application Security Consortium) y CVE International (Common Vulnerabilities and Exposures) las cuales también clasifican a los ataques de estos tipos entre los primeros lugares.

Las fases del diseño del prototipo, codificación, despliegue se exponen a continuación.

a. Diseño del prototipo

Arquitectura: Como se puede observar en la Figure 1, se toma como referencia la arquitectura planteada por JAX-RS [22][25] en el componente de aplicación. A nivel arquitectónico se planteó una de las metodologías de seguridad *Autenticación JAAS*. El flujo arquitectónico dispuesto por el diseño planteado, es similar a una arquitectura Cliente-Servidor donde el cliente inicia una petición y el servidor determina una respuesta.

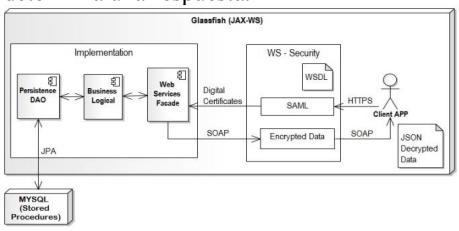

Figure 1 Diseño Arquitectónico

Cómo se puede observar, el diseño arquitectónico está formado por dos partes:

- Servicio Web: Es donde se encuentran el acceso a los datos, la lógica de negocio y lo concerniente a la seguridad, contiene los siguientes componentes:
  - Facade Servicio Web: Encargada de implementar la interfaz a la aplicación, donde se invocan los sub métodos para acceder a los datos mediante la capa de lógica de negocio.
  - Lógica de negocio del servicio: Encargada de proveer a la capa de Facade Servicio Web de todos los métodos que podrá utilizar cuando los requiera.
  - Persistencia: Encargada de gestionar el acceso a los datos para la aplicación. En este caso se utiliza JPA

mediante la implementación EclipseLink. Esta capa contiene todo lo referente a la lógica de acceso a los datos mediante procedimientos almacenados.

- Cliente del Servicio Web: Es el usuario que va a consumir el servicio web a través de un servlet en un navegador web. Este cliente para poder comunicarse correctamente con la aplicación del servicio web debe cumplir con el requerimiento de autenticación mediante aserciones SAML con certificados digitales.
- WS-Security: Corresponde al estándar a través del cual se implementa seguridad en web SOAP, en él se implementa SAML y aspectos de seguridad propios.

b. Codificación.- En esta sección, se utilizan recursos necesarios para construir el prototipo basado en SOA. Entre los recursos utilizados constan: JAX-RS, Data Persistence 2.5.2, Glassfish 4.1, org.glassfish.metro 2.3, SOAP Security 1.3.1. La construcción de los WS implicó la aplicación de las técnicas de seguridad expuestas por OWASP, construyendo métodos capaces de aportar la lógica necesaria a la aplicación SOA para resolver las peticiones del cliente. Para ello se codificó aplicando conceptos de programación orientada a objetos y el empleo del patrón de diseño estructural Facade [24], [26] buscando separar la programación en interfaces de comunicación que permitan la modificabilidad del prototipo.

c. Despliegue.- Previo al despliegue, se empezó configurando el control de seguridad lógico, ello implicó la especificación de roles y usuarios con acceso a los servicios web. Seguidamente se codificaron los archivos de configuración correspondientes al aseguramiento de la aplicación a nivel de servidor, para obtener un enlace entre software y servidor. Aquí se configuró el protocolo de comunicación HTTPS, autenticación y cabeceras de respuestas HTTP seguras [5].

- Autenticación: La autenticación se la realizó mediante aserciones SAML, que requieren configuración de certificados de lado del servidor y cliente, además es necesario la configuración de la clase SAML Callback Handler, mediante la herramienta WSIT.
- Firmado y Cifrado: WSIT además de los mecanismos de autenticación, también provee de protección de las operaciones y de los mensajes SOAP mediante el cifrado y firmado para la confidencialidad e integridad de los datos.
- Clickjack: Como medida de fortalecimiento para evitar ataques de tipo ClickJacking se implementa Clickjack, este tipo de ataques comúnmente pretende robar la identidad de un usuario con el fin de obtener el control de su máquina al hacer clic en una página web; su cadena de configuración corresponde a un filtro de seguridad implementado en la Clase ClickjackFilter, dentro del paquete de controladores, donde su lógica deniega este tipo de ataque.
- HTTPS y Autenticación: se configuró el protocolo de transporte seguro de HTTP con el fin de cifrar las conexiones y que éstas no sean interceptadas por terceros y para permitir implementar la lógica de autenticación. Dichas configuraciones permitieron la implementación de un protocolo de transporte seguro HTTPS para cifrar conexiones, además de su control de seguridad lógica, limitando así el acceso a usuarios no permitidos.

Como resultado de la compilación de las configuraciones aplicadas anteriormente a nivel de código, se obtiene un documento de descripción del servicio web SOAP (WSDL), en el cual se especifican las políticas, aserciones de autenticación, cifrado, firmado, operaciones, puertos, etc.

El WSDL es extenso, pero es necesario indicar que en él se muestran configuraciones, aserciones y políticas de seguridad entre las que constan: *sp:AsymmetricBinding* indica que el elemento de esta política está configurado para utilizar criptografía asimétrica de la clave pública durante el intercambio de mensajes, *sp:AlgorithmSuite* indica el algoritmo utilizado para la firma; para presente prototipo se utiliza el algoritmo Basic128, *sp:IncludeTimestamp* que marca el tiempo utilizado en cada mensaje, *sp:InitiatorToken* indica que la clave pública para firmar el mensaje sería con certificado X.509 y será enviado con cada mensaje desde el cliente al servidor y viceversa, *WssX509V3Token10*: indica el tiempo de ejecución para utilizar el certificado, *sp:OnlySignEntireHeadersAndBody* indica que la firma se realizará sobre toda la cabecera y el cuerpo del mensaje, *sp:SamlToken* indica que se utiliza aserciones de seguridad (SAML) como tokens de seguridad, sp:EncryptedParts *(mensaje de entrada)* y *(mensaje de salida)* indica las partes del mensaje que serán cifradas, en este caso se indica que se cifrará el Body (cuerpo) del mensaje, *sp:SignedParts (mensaje de entrada)* y *(mensaje de salida)*, indica las partes del mensaje a firmar; de igual forma se indica que se firmará el Body del mensaje. Mediante las políticas de seguridad del *WS-Security Policy* expuestas anteriormente se pueden definir los requerimientos y lineamientos que debe cumplir el cliente del servicio web para poder acceder al mismo de forma satisfactoria, por ejemplo, que se encuentren definidos las aserciones y los tokens de autenticación de SAML, los tipos de certificados utilizados, el encriptado y firmado de los mensajes.

IV.PRUEBAS Y RESULTADOS

En base a los conceptos y recomendaciones de codificación y seguridad implementadas en el prototipo SOA, se ejecutaron las validaciones necesarias desde el punto de vista de arquitectura, codificación y seguridad de WS, los resultados se exponen a continuación.

a. Arquitectura de software.- Para el análisis de la arquitectura implementada en el prototipo se utiliza Structural Analysis for Java como herramienta que a través de un análisis extrae las características del diseño del prototipo codificado, en este caso como resultado del análisis se obtuvo que el prototipo contiene un 91 % de estabilidad; este valor según los estándares de la herramienta expone que en aplicaciones que presentan más del 90 % de estabilidad se consideran confiables y presentan calidad deseada para software.

b. Código fuente.- Para esta segunda etapa de pruebas se evaluó la calidad del código escrito a través de la herramienta SonarQube; se realizaron algunas iteraciones para la

evaluación, dos de ellas se ubican en la TABLE 1; dentro de los problemas que generan deuda técnica en aspectos de seguridad constan la escaza implementación de estándares de programación, duplicaciones de código, mal uso de las estructuras de datos; cabe indicar que por cada iteración se realizaron los ajustes correspondientes para minimizar dichos problemas. Entre los problemas constan: bloques duplicados, parámetros del método, excepciones capturadas y variables for each no deben ser reasignados, los nombres de campo deben cumplir con una convención de nombres, los loggers deben ser "private static final" y deben compartir una convención de nombres, métodos no deben estar vacíos.

TABLE 1 PORCENTAJE DE DEUDA TÉCNICA EN CÓDIGO

Iteración	Líneas de código	Funciones	Problemas	Deuda Técnica
1	4857	303	1038	36 días
2	4899	306	240	7 días 4 horas
3	4860	303	102	7 días 4 horas
4	4716	303	3	1 hora

Aplicando los estándares y técnicas recomendadas por OWASP a nivel de código y realizando iteraciones para validar el código a través de herramientas se puede incrementar la calidad y seguridad del prototipo; ello se ve comprobado a través de la métrica SQALE que implementa SonarQube en sus análisis con la que se obtuvo una calificación de A y una deuda técnica del 1.0 %, lo cual justifica la calidad del prototipo escrito.

c. Seguridad.- En esta última etapa se evalúa la seguridad del prototipo el mismo que tiene como objetivo reducir los tipos de vulnerabilidades presentadas anteriormente y que son las que afectan a las aplicaciones web. Para la validación desde el punto de seguridad del prototipo se utilizan como herramientas Vega Subgraph, Owasp Zap y Wireshark que permiten extraer las alertas en el prototipo según el método de extracción de datos, vulnerabilidad y capa de transporte.

En la Figure 2 se observa un pico de alertas fuerte, en la iteración que corresponde a la vulnerabilidad XSS y SQL como metodología de extracción de datos. Estos resultados se presentan de forma general; el estudio de estos puntos se lo ve reflejado en las secciones posteriores divididas en Inyección sql y XSS.

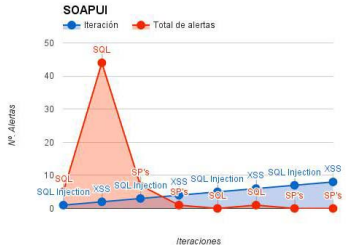

Figure 2 Pruebas de seguridad alertas de seguridad SOAPUI

A. d. Inyección SQL.- A partir de los resultados expuestos en la Figure 2, se subdividió las evaluaciones de seguridad, en base a la forma de extracción de datos: sentencias SQL y llamadas a procedimientos almacenados (SP's) escrito en la codificación del prototipo, y su protocolo de transporte configurado a nivel de servidor; a partir de aquí, las pruebas de inyección SQL reflejan los resultados expuestos en la Figure 3 *Pruebas de Seguridad | Resultados de alertas de Inyección SQL*

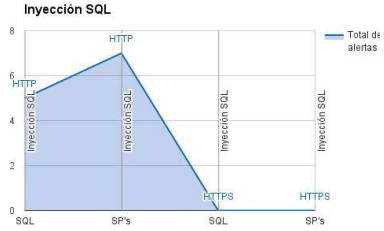

Figure 3 Pruebas de Seguridad | Resultados de alertas de Inyección SQL

Los resultados reflejan mayor cantidad de alertas de seguridad cuando se usa un protocolo de transporte no seguro. También se aprecia que los mayores problemas de seguridad corresponden a las dos formas de extracción de datos aplicadas; es decir, el aseguramiento de este tipo de vulnerabilidad viene dado por la escritura de código y se ve complementada con la aplicación del protocolo de transporte seguro HTTPS.

e. Cross Site Scripting (XSS).- En lo referente a XSS, se aplicó una validación similar a la vulnerabilidad anterior; se evaluó a partir de capa de transporte, vulnerabilidad, y forma de extracción de datos, dichos resultados se ven reflejados en la Figure 4.

Figure 4 Pruebas de Seguridad | Resultado de alertas de XSS.

En la Figure 4 se observa un pico de alertas cuando se usa el protocolo de transporte no seguro y sentencias SQL para la extracción de datos. Hay que recalcar que la técnica de llamadas a procedimientos almacenados brinda un eficiente grado de seguridad, incluso cuando se usa un protocolo de transporte no seguro. En cuanto a Inyección SQL, los problemas se ocasionan principalmente por falta de aplicación de estándares de programación segura. Es importante especificar que la aplicación de mecanismos de seguridad externos a nivel de servidor y comunicación de componentes primarios de arquitectura de software (Cliente-Servidor), aporta a la reducción de debilidades de software; es decir que el cifrado de la conexión aplicando estándares HTTPS reduce este impacto. Para su justificación se evaluó protocolo de

comunicación, forma de extracción de datos, cifrado del mensaje.

Otras de las alertas reportadas por las herramientas al momento de analizar el prototipo desde el punto de vista de seguridad y que fueron resueltas de forma iterativa son: Directory Transversal, X-Frame-Options Header Not Set, Web Browser XSS Protection Not Enabled, X-Content-Type-Options Header Missing, Page Fingerprint Differential Detected, Character Set Not Specified: Cuando no se especifica un conjunto de caracteres en la cabecera de la respuesta de tipo de contenido o en el cuerpo de la cabecera de la respuesta del navegador.

Para la analizar si el prototipo contiene vulnerabilidad de exposición de datos sensibles se utiliza la herramienta Wireshark, con la cual se procede a verificar el tráfico que existe al realizar una petición HTTP hacia el servicio. Además se implementó SAML para la parte de autenticación de los servicios web; en donde se evidencia que si no se realiza las respectivas configuraciones tanto en el cliente como en el servicio, el cliente no podrá acceder a los datos que retorne el servicio web. La aplicación cliente no puede acceder a la información debido a los siguientes errores:

- *Grave: WSS1500:* el manejador de nombres de usuario no se ha configurado correctamente mediante la devolución de llamada y es nulo.
- *Grave: WSS0216:* Se ha producido un error al utilizar CallbackHandler para UsernameCallback
- *Grave: WSS0217:* se ha producido un error al utilizar el método CallbackHandler.

Los errores antes expuestos pertenecen a errores de configuración de Autenticación y Autorización del SAML (Security Assertion Markup Language); éstos se resuelven con las respectivas configuraciones en el cliente y en el servicio, mediante el mecanismo se seguridad *SAML Sender Vouches with Certificates*.

V.CONCLUSIONES

Como parte de la implementación de recomendaciones y estándares se concluye que la especificación WS-Security permite utilizar mecanismos de seguridad de autenticación SAML con certificados digitales, así como mecanismos de cifrado y firmado provistos por la herramienta WSIT con el fin de proteger el transporte de los mensajes; además, es necesario utilizar el mecanismo de seguridad SAML Sender Vouches with Certificates, del lado del servidor y del cliente para que las peticiones del cliente sean validadas y consumidas mediante credenciales en formatos de tokens X.509 SAML por el servicio web. El uso del framework Metro permite la implementación de JAX-WS y WSIT para la comunicación a través de XML con el protocolo SOAP y la especificación WS-Security de forma automatizada y ocultando la complejidad de los mensajes SOAP. Con la implementación de WSIT se logró seguridad en la mensajería de los servicios web mediante especificaciones del WS-Security, mecanismos de autenticación de cifrado y de firmado de los mensajes, permitiendo integridad y confidencialidad de los datos. El uso de normativas OWASP como referencia para el aseguramiento de aplicaciones web, mitiga en gran medida la ausencia de estándares de seguridad establecidos para SOA. El implementar procedimientos almacenados como recomienda OWASP, garantiza el aseguramiento de información y principalmente previene ataques de tipo Inyección SQL; además aporta al rendimiento en relación a petición y respuesta de los servicios web SOA. Como parte del diseño y codificación del prototipo, se pudo comprobar que la principal forma de prevención de ataques de tipo XSS consiste en realizar una limpieza a los parámetros ingresados por el usuario final en los identificadores de recursos (URI), previo a la ejecución y resolución de la petición HTTP; además, el uso de canales seguros de comunicación como HTTPS es imprescindible dado que cifra las conexiones, ayudando así a prevenir el robo de información y violación de accesos lógicos. El proceso de validación por iteraciones permite ajustar el código fuente a los estándares de calidad recomendados por la herramienta SonarQube, la cual utiliza métricas y estándares de calidad basados en SQALE.

VI.REFERENCIAS

[1] G. Serme, A. S. De Oliveira, J. Massiera, and Y. Roudier, "Enabling message security for RESTful services," *Proc. - 2012 IEEE 19th Int. Conf. Web Serv. ICWS 2012*, pp. 114–121, 2012.

[2] G. McGraw, "Software Security," *Secur. Privacy, IEEE*, vol. 2, no. 2, pp. 80–83, 2004.

[3] M. Yag, "Integrando la Ingeniería de Seguridad en un Proceso de Ingeniería Software," no. September 2015, 2015.

[4] L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice," *Architecture*, vol. 2nd, no. JANUARY 2003, p. 528, 2003.

[5] "Owasp." [Online]. Available: https://www.owasp.org/index.php.

[6] a. Nadalin, C. Kaler, P. Hallam-Baker, R. Monzillo, and Others, "Web Services Security: SOAP Message Security 1.0," *Security*, no. March, p. 56, 2004.

[7] L. E. Moser, P. M. Melliar-Smith, and W. Zhao, "Building dependable and secure web Services," *J. Softw.*, vol. 2, no. 1, pp. 14–26, 2007.

[8] Y. Makino, S., Tamura, K., Imamura, T., & Nakamura, "WS-Security, Implementation and performance of Security," *Int. J. Web Serv. Res.*, vol. 1, pp. 58–72, 2004.

[9] M. Naedele, "Standards for XML and web services security," *Computer (Long. Beach. Calif).*, vol. 36, no. 4, pp. 96–98, 2003.

[10] "Authorization Service (JAAS) Reference Guide." [Online]. Available: http://docs.oracle.com/javase/1.5.0/docs/guide/security/jaas/JAASRefGuide.html.

[11] E. Rescorla, "Http over TLS," pp. 1–7, 2000.

[12] K. Wei and M. Muthuprasanna, "Preventing SQL injection attacks in stored procedures," *Aust. Softw. Eng. Conf.*, p. 8 pp.–198, 2006.

[13] Oracle, "JAX-WS."

[14] Oracle, "Metro."

[15] K. Lawrence, C. Kaler, and P. Hallem-baker, "Web Services Security : SAML Token Profile 1.1," *OASIS Stand.*, no. February, pp. 1–31, 2006.

[16] S. Track and W. Product, "WS-SecurityPolicy 1.2," *OASIS Stand.*, no. April, 2012.

[17] M. J. Munro, "Product metrics for automatic identification of 'bad smell' design problems in Java source-code," *Proc. - Int. Softw. Metrics Symp.*, vol. 2005, no. Metrics, pp. 125–133, 2005.

[18] J. Pablo, O. Delgado, J. Pablo, and O. Delgado, "Análisis de seguridad y calidad de aplicaciones (Sonarqube) Análisis de seguridad y calidad de aplicaciones (Sonarqube)," 2015.

[19] J.-L. Letouzey, "The SQALE method for evaluating Technical Debt," *2012 Third Int. Work. Manag. Tech. Debt*, pp. 31–36, 2012.

[20] S. Lim, "Method of Application Driven QoS Service in Open Service Platform based on RESTful Web Services," pp. 632–633, 2014.

[21] T. Fertig and P. Braun, "Model-driven Testing of RESTful APIs,"

pp. 1497–1502, 2015.
[22] M. Hadley and P. Sandoz, “JAX-RS: Java™ API for RESTful Web Services,” *Oracle Corp.*, p. 96, 2008.
[23] “Patrones de Diseño.” [Online]. Available: http://msdn.microsoft.com/es-es/library/bb972240.aspx.
[24] J. Shuai and M. Huaxin, “Design Patterns in Object Oriented Analysis and Design,” *IEEE xplore*, pp. 326–329, 2011.
[25] D. Coward, “Java ™ API for WebSocket,” in *Java ™ API for WebSocket*, 2013, p. 43.
[26] M. R. Dale and C. Izurieta, “Impacts of Design Pattern Decay on System Quality,” 2014.

View publication stats