Desarrollo de Servicios Web 1

Tema Nº11 y N°12:Web Service Security

Indicador de logro Nº11 y N°12:Aplica Security para Web Service a través de las tecnologías de autenticación por usuario y contraseña.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº11 y N°12:**

WEB SERVICE SECURITY

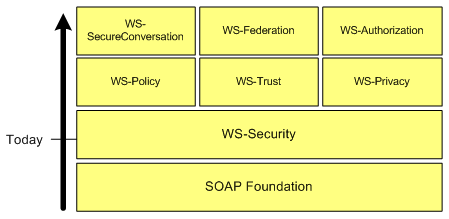
**Subtema 1:**

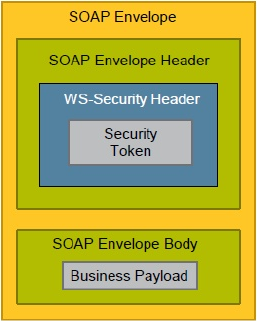
WS-SECURITY

**MARCO TEÓRICO**

Web Services Security, o WS-Security. Expone servicios en un entorno que requiere protección de mensajes para que no sean alterados ni leídos en tránsito, o en situaciones en las cuales el remitente debe ser identificado positivamente. El término "WS-Security" se utiliza generalmente para referirse a un grupo de especificaciones que manejan cifrado y firmas digitales, lo que le permite crear una aplicación segura.

Se debe tener una comprensión básica de SOAP, y por extensión, se necesita una comprensión básica de XML.





**La necesidad de tener seguridad**

La especificación SOAP básica no establece la protección de mensajes, y deja esas especificaciones extendidas. El problema yace en la naturaleza de una aplicación de servicios web. En la mayoría de los casos, tratamos con SOAP sobre HTTP, lo que significa que cada mensaje debe pasar por uno o más nodos intermedios, cualquiera de los cuales puede leer y/o alterar un mensaje. Y eso suponiendo que la solicitud SOAP misma fuera directa. En algunos casos, los mensajes SOAP están diseñados específicamente para viajar a través de más de un modo antes de llegar a su destino final.

El resultado final es que necesitamos evitar que alguien que no sea el destinatario deseado lea información confidencial, a fin de evitar la interceptación. Además, necesitamos una manera de evitar que alguien que no sea el remitente deseado envíe mensajes, a fin de evitar el acceso no autorizado.

La especificación SOAP brinda un medio para agregar información de seguridad – agregamos información a un elemento de Encabezado del Sobre – pero no especifica qué debe ser la información. Para solucionarlo, necesitamos WS-Security.

**Las partes de WS-Security**

Hay tres grandes problemas en la protección de intercambios de mensajes SOAP, y WS-Security brinda respuestas para todos ellos, aunque no directamente. Es, de hecho, una especificación que habla no sobre cómo proteger el mensaje, sino cómo hacer saber al destinatario que se ha protegido el mensaje. Para realizar la protección real, WS-Security referencia especificaciones adicionales. Veamos cómo funciona eso.

El primer problema es identificar y autenticar al cliente. Debido a que hay muchas maneras diferentes de crear tokens de seguridad, WS-Security no especifica ningún medio particular, sino que define cómo se deben transferir los tokens de seguridad dentro de los mensajes SOAP. En otras palabras, hace saber al destinatario cómo extraer tokens de seguridad del mensaje para su procesamiento.

El segundo problema es garantizar la integridad del mensaje. WS-Security utiliza firmas digitales para eso, empleando la especificación de Firma XML en lugar de inventar algo nuevo. La Firma XML es una recomendación W3C que incluye un mecanismo para firmar documentos XML de manera digital.

El tercer problema es mantener el mensaje seguro contra la interceptación mientras está en tránsito. Una vez más, WS-Security utiliza otro estándar W3C, esta vez Cifrado XML, que brinda un mecanismo para cifrar documentos XML.

**La meta de WS-Security**

Es permitir que las aplicaciones construyan intercambios seguros de mensajes SOAP. Esta especificación está destinada a proporcionar un conjunto flexible de mecanismos que puedan ser utilizados para construir una amplia gama de protocolos de seguridad; en otras palabras, esta especificación intencionadamente no describe explícitamente protocolos de seguridad, sino que define las primitivas para poder hacerlo.

Como con cualquier protocolo de seguridad, WS-Security por sí sólo no garantiza la seguridad y se debe aplicar un gran esfuerzo para garantizar que los protocolos de seguridad construidos con WS-Security no resulten vulnerables cuando se enfrentan a un amplio abanico de ataques. Como ya se ha dicho, el Lenguaje de Seguridad de Servicios Web (WS-Security) soporta una amplia variedad de modelos de seguridad. La siguiente lista identifica los requisitos clave que han conducido el desarrollo de la especificación:

* Múltiples tokens de seguridad para autenticación y autorización.
* Múltiples dominios de seguridad.
* Múltiples tecnologías de seguridad.
* Seguridad a nivel de mensaje de extremo a extremo de la comunicación y no tan sólo seguridad punto a punto a nivel de transporte.

**Servicios Web seguros en Metro**

Como alternativa a CXF y Axis 2, encontramos Metro, este framework comprende los estándares para servicios web Java:

* JAX-WS (Java API for XML Web Services): API de Java para servicios web SOAP básicos.
* JAXB (Java Architecture for XML Binding): API de Java para vincular datos XML y objetos Java.
* WSIT (Web Services Interoperability Technologies): Plataforma Java que da soporte a las extensiones SOAP (WS-\*), incluyendo tecnologías para servicios web seguros (WS-Security y WS-Trust).

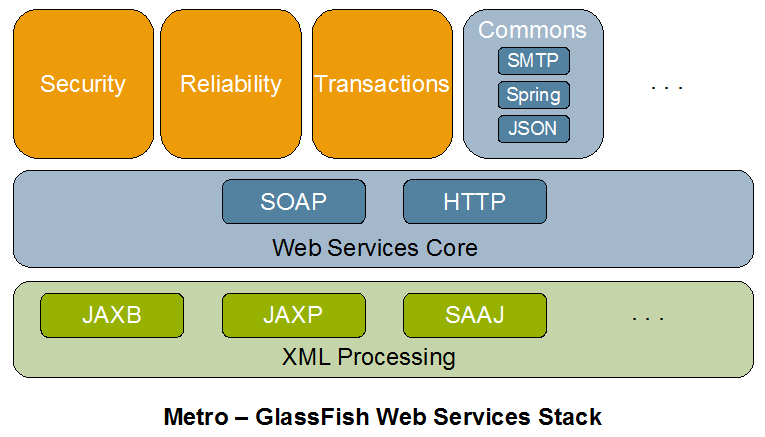
Metro se ha diseñado entorno a las tecnologías estándar desarrolladas por Sun (JAX-WS, JAXB y WSIT), mientras que Axis2 queda abierto a utilizar diferentes tecnologías. Por ejemplo, en Axis2 para vincular datos Java-XML podemos optar por tecnologías distintas de JAXB. Sin embargo, a pesar de contar con esta flexibilidad, debemos tener en cuenta que si utilizamos JAX-WS y JAXB dentro de Axis2 encontraremos bastantes limitaciones, entre ellas el no poder utilizar WS-Security.

En Axis2 necesitamos instalar módulos adicionales para utilizar las tecnologías de segunda generación (WS-\*), mientras que el propio motor de Metro ya integra a todas ellas. Esto hace a Metro más fácil de configurar, pero menos flexible.

En cuanto al despliegue, los servicios web Axis2 se instalan como módulos independientes (AAR) dentro de la aplicación web de Axis2, mientras que en Metro debemos crear nuestro propio módulo web (WAR) que incluya el servicio. Si desplegamos a un servidor distinto a Glassfish deberemos incluir las librerías de dicha pila dentro de nuestro WAR o en los directorios de librerías comunes del servidor, y además ficheros de configuración adicionales.

Aunque Metro puede ser instalado en cualquier servidor de aplicaciones, es especialmente interesante utilizarlo en un entorno Netbeans-Glassfish. Metro forma parte de Glassfish, por lo que no será necesario añadir librerías ni configuración adicional a dicho servidor.

Además, si como entorno de desarrollo utilizamos Netbeans tendremos a nuestra disposición una interfaz gráfica para configurar los servicios web, con la que de forma sencilla podremos crear servicios web seguros sin necesidad de tratar con ficheros de configuración directamente. Vamos a centrarnos ahora en esta forma de crear servicios web seguros.



**1. MATERIALES**

Para la experiencia a realizar se requiere lo siguiente:

1. EQUIPO

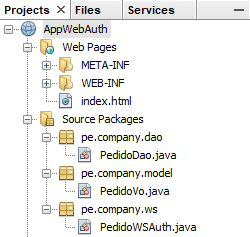
Se requiere tener un equipo de cómputo (PC o laptop).

1. HERRAMIENTAS

* Java Developer Kit (JDK) version 8.x
* NetBeans IDE
* Apache TomCat 8.x.x
* Metro 2.0
* SoapUI 5.5

**2. PROCEDIMIENTO**

Se tiene el proyecto Web:



Clase VO:

package pe.company.model;

import java.util.Date;

public class PedidoVo

{

private Integer id\_pedido;

private Date fecha\_pedido;

private String cliente;

private String vendedor;

private String producto;

private Integer cantidad;

private Double importe;

public PedidoVo() {}

public PedidoVo(Integer id\_pedido, Date fecha\_pedido, String cliente, String vendedor,

String producto, Integer cantidad, Double importe) {

this.id\_pedido = id\_pedido;

this.fecha\_pedido = fecha\_pedido;

this.cliente = cliente;

this.vendedor = vendedor;

this.producto = producto;

this.cantidad = cantidad;

this.importe = importe;

}

//Getters and Setters

}

Clase Dao:

package pe.company.dao;

import java.util.\*;

import pe.company.model.PedidoVo;

public class PedidoDao

{

//BD virtual

private static final Map<Integer,PedidoVo> *itemsPedido*=new HashMap();

static {

PedidoVo pedido1=new PedidoVo(110036,new GregorianCalendar(1990,0,30).getTime(),

"Ace International","Tom Snyder","Montador",9,22500.0);

PedidoVo pedido2=new PedidoVo(112961,new GregorianCalendar(1989,11,17).getTime(),

"J.P. Sinclair","Sam Clark","Bisagra Izqda",7,31500.0);

PedidoVo pedido3=new PedidoVo(112963,new GregorianCalendar(1989,11,17).getTime(),

"Acme Mfg.","Bill Adams","Articulo Tipo 4",28,3276.0);

PedidoVo pedido4=new PedidoVo(112968,new GregorianCalendar(1989,9,12).getTime(),

"Jones Mfg.","Dan Roberts","Articulo Tipo 4",34,3978.0);

*itemsPedido*.put(pedido1.getId\_pedido(),pedido1);

*itemsPedido*.put(pedido2.getId\_pedido(),pedido2);

*itemsPedido*.put(pedido3.getId\_pedido(),pedido3);

*itemsPedido*.put(pedido4.getId\_pedido(),pedido4);

}

public PedidoDao() {}

//método buscar por ID

public PedidoVo findById(Integer id\_pedido) {

PedidoVo pedido=*itemsPedido*.get(id\_pedido);

return pedido;

}

//otros métodos

//code...

}

Clase Web Service:

package pe.company.ws;

import java.util.List;

import java.util.Map;

import javax.annotation.Resource;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

import javax.xml.ws.WebServiceContext;

import javax.xml.ws.handler.MessageContext;

import pe.company.dao.PedidoDao;

import pe.company.model.PedidoVo;

@WebService(serviceName="PedidoWSAuth")

public class PedidoWSAuth {

@Resource

private WebServiceContext webServiceContext;

//instancia al Dao

private PedidoDao pedidoDao=new PedidoDao();

private String username="";

private String password="";

@WebMethod(operationName="buscar")

public PedidoVo buscar(@WebParam(name="id\_pedido") Integer id\_pedido)

{

PedidoVo pedido=new PedidoVo();

//contexto para extraer mensaje de cabecera

MessageContext messageContext=webServiceContext.getMessageContext();

Map requestHeader=(Map)messageContext.get(MessageContext.*HTTP\_REQUEST\_HEADERS*);

List usernameList=(List)requestHeader.get("USERNAME");

List passwordList=(List)requestHeader.get("PASSWORD");

if((usernameList!=null)&&(passwordList!=null))

{

username=(String)usernameList.get(0);

password=(String)passwordList.get(0);

}

//verificando credenciales

if((username.equals("admin"))&&(password.equals("d123"))) {

pedido=pedidoDao.findById(id\_pedido);

}

else

{

pedido.setFecha\_pedido(null);

pedido.setCliente("Sin acceso");

pedido.setVendedor("Sin acceso");

pedido.setProducto("Sin acceso");

pedido.setCantidad(0);

pedido.setImporte(0.00);

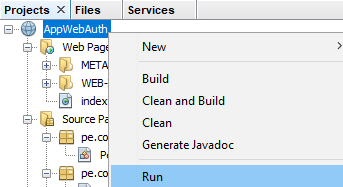
}

return pedido;

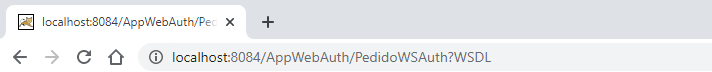
}

}

Ejecución del proyecto:



Luego se pasa a ver el WSDL:



<definitions

xmlns:wsu=*"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"*

xmlns:wsp=*"http://www.w3.org/ns/ws-policy"*

xmlns:wsp1\_2=*"http://schemas.xmlsoap.org/ws/2004/09/policy"*

xmlns:wsam=*"http://www.w3.org/2007/05/addressing/metadata"*

xmlns:soap=*"http://schemas.xmlsoap.org/wsdl/soap/"*

xmlns:tns=*"http://ws.company.pe/"*

xmlns:xsd=*"http://www.w3.org/2001/XMLSchema"*

xmlns=*"http://schemas.xmlsoap.org/wsdl/"*

targetNamespace=*"http://ws.company.pe/"* name=*"PedidoWSAuth"*>

<types>

<xsd:schema>

<xsd:import namespace=*"http://ws.company.pe/"*

schemaLocation=*"http://localhost:8084/AppWebAuth/PedidoWSAuth?xsd=1"* />

</xsd:schema>

</types>

<message name=*"buscar"*>

<part name=*"parameters"*

element=*"tns:buscar"* />

</message>

<message name=*"buscarResponse"*>

<part name=*"parameters"* element=*"tns:buscarResponse"* />

</message>

<portType name=*"PedidoWSAuth"*>

<operation name=*"buscar"*>

<input wsam:Action=*"http://ws.company.pe/PedidoWSAuth/buscarRequest"*

message=*"tns:buscar"* />

<output wsam:Action=*"http://ws.company.pe/PedidoWSAuth/buscarResponse"*

message=*"tns:buscarResponse"* />

</operation>

</portType>

<binding name=*"PedidoWSAuthPortBinding"* type=*"tns:PedidoWSAuth"*>

<soap:binding

transport=*"http://schemas.xmlsoap.org/soap/http"* style=*"document"* />

<operation name=*"buscar"*>

<soap:operation soapAction=*""* />

<input>

<soap:body use=*"literal"* />

</input>

<output>

<soap:body use=*"literal"* />

</output>

</operation>

</binding>

<service name=*"PedidoWSAuth"*>

<port name=*"PedidoWSAuthPort"*

binding=*"tns:PedidoWSAuthPortBinding"*>

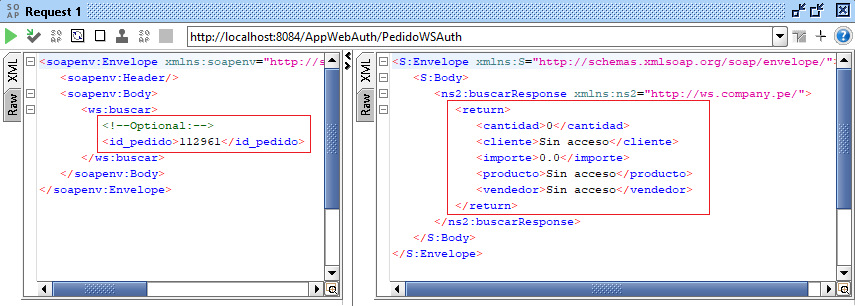
<soap:address location=*"http://localhost:8084/AppWebAuth/PedidoWSAuth"* />

</port>

</service>

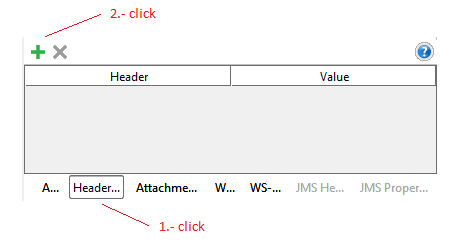
</definitions>

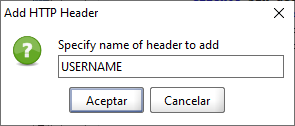
Hacemos la prueba con SoapUI:

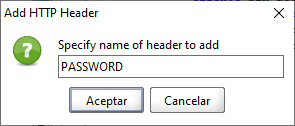


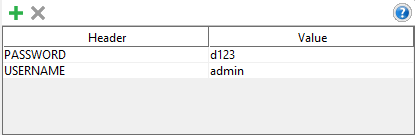
Nota: Se observa que no se tiene permiso

Para poder acceder se tiene que poner en el Header:

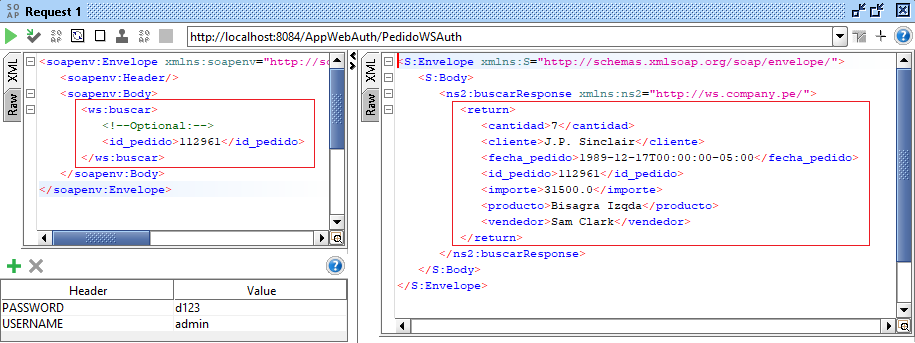


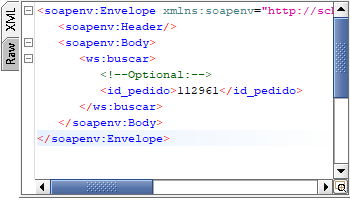


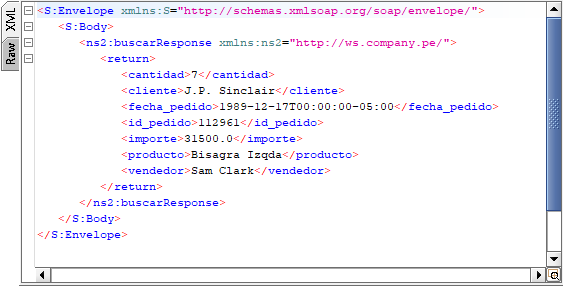




Ejecutamos el SopaUI nuevamente:







Nota: Se observa el acceso al servicio con las credenciales apropiadas

**ACTIVIDAD VIRTUAL:**

1. **CUESTIONARIO TÉCNICO**

Revisa y analiza el tema desarrollado en la presente sesión, luego responde las siguientes preguntas propuestas:

* ¿Qué es WS-Security?
* ¿Qué ventajas WS-Security proporciona?
* ¿En qué caso aplica el uso de WS-Security? Describa con un ejemplo.
* Realice una aplicación cliente Desktop o Web que consuma el servicio desarrollado en esta guía
* Mediante NetBeans IDE, Base de datos, Apache Tomcat y WS-Security crear web service con autenticación por usuario y contraseña. Realizar pruebas unitarias.

1. **ENLACES DE REFERENCIA**

* http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/211
* https://www.dsi.uclm.es/descargas/technicalreports/DIAB-05-01-2/Seguridad\_en\_Servicios\_Web.pdf
* http://www.jtech.ua.es/ayto/sws/sesion11-apuntes.html

1. **CONCLUSIONES DE LA EXPERIENCIA**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_