# JAAS (SEGURIDAD)

Echemos un vistazo a las formas de proteger las aplicaciones Java EE. En este capítulo, debemos ver cómo funciona la seguridad de Java EE; luego, ¿cómo definimos la autenticación y la autorización dentro del mundo Java EE; usar módulos de inicio de sesión para autenticar a los usuarios; y también definir roles de aplicación y restricciones de seguridad para asegurarse de que solo los usuarios debidamente autorizados puedan acceder a los recursos relevantes.

Intentamos definir la seguridad en Java EE declarativamente. Pero a veces, es posible que también tengamos que hacerlo mediante programación. Así que veremos ambos enfoques: enfoque declarativo y programático. Al final de la sesión, también veremos un conjunto muy diferente de problemas de seguridad relacionados con los servicios web. Porque verá, los servicios web, los servicios SOAP... tienen sus propios estándares de seguridad. Bueno, básicamente porque no están necesariamente escritos en Java. Tipo de servicio de servicios web, cuya implementación puede variar. No es necesariamente solo Java.

Y por lo tanto, la estandarización de la seguridad para el servicio web tiene que trascender solo el mundo de Java y ser más universal. Así que no es solo Java EE. Podría ser un conjunto más amplio de especificaciones que debemos considerar con respecto a los servicios web. Pero comencemos con la seguridad de Java EE.

Para implementar la seguridad en las aplicaciones Java EE, podríamos usar la API de servicios de autenticación y autorización de Java. La autenticación es un proceso de establecimiento de la identidad de la persona que llama: quienquiera que nos invoque, o sea quien dice ser. Así que asegúrese de que entendamos quiénes son estas personas que llaman. La autorización es establecer los permisos para que la persona que llama realice acciones en nuestro sistema: permitirles o no permitirles acceder a los recursos y realizar llamadas.

Además de eso, los servidores Java EE también se pueden configurar para proteger las comunicaciones de transporte, los protocolos de transporte. Y hay dos cosas que podría considerar configurar en esta situación. Y es confidencialidad e integridad. La confidencialidad significa que está cifrando y descifrando paquetes de datos, los paquetes de red que entran y salen de su servidor para protegerlos del acceso no autorizado. Integridad significa que está verificando si el paquete no se corrompió en la transmisión, si aún es válido. ¿Es el mismo que se envió?

La API de servicios de autenticación y autorización de Java le permite realizar una configuración declarativa de las propiedades de seguridad. Esto es parte de la configuración de su servidor. Ahora, hay una parte de la configuración que debe realizar en el servidor de aplicaciones real. Y hay una parte que debe hacer dentro de su aplicación.

En el servidor de aplicaciones, debe configurar el proveedor de seguridad JAAS. Un proveedor de seguridad es un módulo conectable que va a la API de la aplicación JAAS y permite que su servidor acceda a servidores de seguridad que pueden estar ubicados externamente. Algunos servicios Java EE pueden proporcionar la parte del servicio de seguridad de la configuración del servidor. Por ejemplo, el servidor WebLogic tiene un servidor LDAP incorporado directamente como parte del mismo WebLogic.

Pero incluso en ese caso, técnicamente hablando, el resto del servidor Java EE accede a eso a través del módulo del proveedor de seguridad conectable, incluso si se trata de un servidor de seguridad recursivo que está utilizando y que está integrado directamente en el servidor Java EE. Por supuesto, en grandes sistemas de producción, probablemente usaría un servicio de seguridad externo, no sé, todo tipo de servicio de directorio de Internet que guarda información sobre sus usuarios, grupos a los que desea acceder desde su entorno. Así que eso lo hacen los administradores del servidor. Ellos son los responsables de esa parte.

Ahora, como desarrollador, su trabajo es configurar la seguridad dentro de la aplicación Java EE. Puede hacerlo con anotaciones o puede hacerlo con descriptores de implementación XML. Entonces serían archivos como web.xml o ejb-jar.xml o anotaciones dentro de sus beans o dentro de sus componentes web. Los bits de la configuración que necesita crear incluyen un módulo de inicio de sesión, que forma parte de la parte web de la aplicación. Y el módulo de inicio de sesión es el mecanismo a través del cual le gustaría desafiar a las personas que llaman para que produzcan la autenticación.

¿Entonces que te gustaría hacer? ¿Quieres que presenten usuario y contraseña? ¿Cómo exactamente? ¿Quieres... no sé, presentar algún tipo de firma digital, validar eso? Podrían ser diferentes medios de garantizar que la persona que llama es genuina. Nombre de usuario, contraseña es probablemente el más obvio. Pero, de nuevo, es tu elección. Ese es su módulo de inicio de sesión: cómo procesa exactamente la información de autenticación. Bien, registra el módulo de inicio de sesión con el archivo web.xml. Es algo que tiene que hacer en un descriptor de implementación.

Ahora, el siguiente bit, roles de aplicación y roles de seguridad, puede hacerlo en descriptores de implementación como web.xml o ejb-jar.xml o mediante anotaciones, como prefiera. Estos están diseñados para proporcionar una descripción de los roles de seguridad de la aplicación. Piensa en ello de esta manera. ¿Sabe quiénes serán los usuarios reales en un sistema futuro? Realmente no. Como desarrollador, no tiene conocimiento sobre los usuarios y grupos del mundo real dentro de su servidor de seguridad. El administrador del servidor probablemente lo sepa. Como desarrollador que está desarrollando el código, probablemente no esté realmente al tanto.

Entonces, lo que hace en su lugar es inventar roles de seguridad con una anotación DeclareRoles de la nada. Simplemente diga, creo que en mi código necesito inventar el rol A, B, lo que sea: cliente, empleado, los roles que necesite. Así que solo inventa estos roles. Cualquier restricción de seguridad que desee especificar, la describe como una restricción que limita el acceso a ciertos recursos en un contenedor web que serán URL web, servlets, páginas. En un contenedor EJB los recursos serían Enterprise JavaBeans, sus métodos.

Entonces, cierto recurso, limita el acceso a ese recurso, a esa URL o a ese bean, solo a los miembros de estos roles lógicos. Dicho esto, debe ser miembro de un rol de empleado para acceder a esa URL. Tienes que ser miembro del rol de cliente para llamar a ese bean, o algo así. Multa.

Ahora, hay un pequeño problema. Tenemos roles de aplicación. Tenemos restricciones a las que se hace referencia en estos roles de aplicación, pero no están relacionadas con usuarios y grupos del mundo real. Por lo tanto, debemos asignar los roles de aplicación que ha inventado, como desarrollador, como parte del código de su aplicación, a usuarios y grupos del mundo real que el administrador de su servidor mantiene en el servidor de seguridad, al que accedemos a través del proveedor de seguridad.

Y eso se hace con archivos descriptores de implementación específicos del servicio. En un escenario de WebLogic, será WebLogic XML o WebLogic ejb-jar.xml. Bueno, y podrían ser descriptores diferentes en JBoss o WebSphere o Tomcat o lo que sea. Dependiendo del servidor que estés usando, habrá diferentes descriptores XML para cada uno de estos casos. Pero todos tienen un descriptor equivalente, en el que puede tomar un nombre de función que está utilizando dentro de su archivo WAR o JAR y asignarlo a grupos o usuarios del mundo real que tenía en su servidor de seguridad.

Y por lo tanto, cuándo el usuario iniciará sesión en el sistema, utilizando cualquier módulo de inicio de sesión que haya configurado. Así que el usuario inicia sesión. El usuario pasará a ser miembro de ciertos grupos. Y luego este usuario, estos grupos se asignan a algunos roles de aplicación. Y ese mapeo luego se traduce en permiso para que ese usuario acceda a cualquier recurso, según cómo los roles declarados se relacionan con las restricciones de seguridad, según la configuración que proporcionó, ya sea a través de anotaciones o descriptores XML.

Bueno, este es el flujo de solicitud de autenticación y autorización. Una persona que llama hace la llamada a nuestro servidor Java EE, envía o solicita. Eso puede desencadenar inmediatamente la invocación de un módulo de inicio de sesión. Podría ser que el usuario llame explícitamente a algún tipo de página de inicio de sesión. Podría ser que el usuario esté tratando de acceder a otra URL, que está restringida por la restricción de seguridad. Entonces hay una restricción. Si el usuario aún no ha iniciado sesión, el servidor activará el proceso de inicio de sesión; básicamente, enviará al usuario al módulo de inicio de sesión que ha proporcionado.

El usuario proporcionará información de autenticación: nombre de usuario, contraseña, lo que sea. Eso se verificará a través de la API de JAAS a través del proveedor de seguridad contra el servidor de seguridad externo real. Y eso concluye el proceso de validación de las credenciales proporcionadas por el usuario. Una vez hecho esto, podemos proceder a resolver la autorización. Derecha.

Ahora, solo una referencia muy rápida a la terminología. En lugar de decir usuario, la API de JAAS hace referencia a un término denominado principal de seguridad. La entidad de seguridad puede ser un usuario. Pero verá, el invocador que llamó a su aplicación puede no ser un ser humano, ¿verdad? Podría ser otra aplicación. Entonces, principal es un término más generalizado, que dice que es la persona que llama a quien usted ha autorizado.

La entidad de seguridad es una identidad de seguridad conocida. En un caso simple, sí, es el usuario. En un caso más complejo, esa podría ser alguna aplicación. Tal vez ellos, en lugar de la contraseña del nombre de usuario, usen una firma digital. Cualquiera que sea la forma en que se haga, en ese momento, ya habrá autenticado a la persona que llama. Así que sabes quiénes son.

Ahora está comprobando si tienen acceso a un recurso en particular. ¿Tienen permiso, de acuerdo con la configuración de restricciones de seguridad, para llamar a la página, URL, bean, lo que sea que estén tratando de llamar? Y si lo hacen, se concede el acceso al recurso. Y pueden proceder con una invocación y obtener su respuesta del servidor. Bueno, si no lo hacen, obtendrán un error.

Bien, puede haber dos tipos de errores en este proceso. Una es que no pueden iniciar sesión en primer lugar. Así que será un error de autenticación. Pero otro tipo de error podría ser que hayan iniciado sesión correctamente, de acuerdo. Simplemente no tienen permiso para ir a una URL en particular. Y en ese caso, habrá un error de autorización, dependiendo de lo que salió mal en ese proceso. O de lo contrario obtendrán el recurso.

Ahora, configuración del módulo de inicio de sesión. Eso se hace en un archivo web.xml. Y aquí hay varias posibilidades en cuanto a lo que podría ser el módulo de inicio de sesión: HTTP básico, resumen, formulario, cliente, mutuo. El más común es probablemente la forma. Esa es probablemente la forma de autenticación más utilizada, que es exactamente lo que muestra el ejemplo aquí.

Entonces, un formulario básicamente significa que construyes una página HTML personalizada: tu página, sea lo que sea. Y podría ser simplemente HTML. No tiene que ser dinámico, en realidad. De todos modos, la página contiene un formulario simple que se envía al servlet, llamado j\_security\_check. No escribes ese servlet. Ya está escrito para ti. Y se envía a los parámetros j\_username, j\_password, tan simple como eso. Entonces eso realiza la autenticación por usted.

En un archivo web.xml, todo lo que necesita hacer es decir, este es el tipo de autenticación que me gustaría usar, autenticación de formulario. Y luego configure la ubicación de esa página de inicio de sesión. Ah, y por supuesto, si el usuario tiene problemas para iniciar sesión, entonces qué tipo de página de error mostrarle al usuario: páginas personalizadas, todo lo que quieras, en realidad.

Ahora, esto de aquí, que es el reino de la seguridad, ¿qué es eso? Vas al sitio web. Usted proporciona un nombre de usuario y una contraseña. Tienes éxito. Está bien. Así que ha sido autenticado-- ¡ta-da! Vas a otra página en ese mismo sitio. ¿Necesita proporcionar un nombre de usuario y contraseña de nuevo? No si estas páginas están en el mismo reino.

Realm es básicamente una etiqueta contra la cual se memoriza el token de autenticación del usuario. Entonces, básicamente, cuando se autentica con éxito, el navegador recibe el token del servidor que dice sí, está bien. Se quien eres. La próxima vez que me llames, preséntame esa ficha.

La próxima vez que el navegador llame al servidor, presente ese token de autenticación, el servidor solo está verificando, no está revalidando todo nuevamente. No, solo verifica si el usuario está accediendo a recursos dentro del mismo reino. Si el usuario accede a los recursos dentro del mismo dominio, el token sigue siendo válido. El usuario puede continuar, siempre que sus restricciones de seguridad permitan que ese rol acceda al recurso. Si el usuario intenta acceder a una página que se encuentra en otro reino, la página de inicio de sesión se activará nuevamente.

Básicamente, solo describir el reino es describir un grupo de recursos para los cuales una sola página de inicio de sesión servirá. Si es otro grupo de recursos donde necesita volver a autenticarse, está bien. Crea otro reino. Tal vez quieras hacer eso, no lo sé. Depende Tal vez haya un tipo adicional de área extra segura del sitio web donde requiera que los usuarios realicen un inicio de sesión adicional. Y ese podría ser el mensaje: le pediría que cree otra configuración de dominio. Así que esa fue probablemente la forma más común.

Permítanme comentar rápidamente sobre algunas otras formas. HTTP básico: no sé si lo has visto. Pero es cuando accede al sitio web, y aparece en su pantalla, un pequeño cuadro de diálogo que dice, ingrese nombre de usuario, contraseña. Eso es todo. Eso es lo básico para ti. Bueno, podrías usar eso, supongo.

Además de la autenticación de formulario, existen otras formas de autenticación que puede utilizar. Por ejemplo, HTTP básico, probablemente lo hayas visto en algunos sitios web. Está accediendo al sitio web y aparece un cuadro de diálogo que dice, ingrese el nombre de usuario y la contraseña. Resumen HTTP: muy similar a eso. El mismo cuadro de diálogo aparece en el sitio web. Pero se intenta transmitir la contraseña de forma codificada.

Hay un problema con estos enfoques. No son muy utilizados. Y, de hecho, algunos servidores de aplicaciones ni siquiera admiten el resumen. Y déjame explicarte cuál es el problema. Si está accediendo a un sitio web a través del protocolo HTTP, entonces está transmitiendo cualquier cosa desde su cliente a su servidor en texto sin formato. Entonces, ¿de qué sirve preguntar por una contraseña si la vas a transmitir a través de la red en texto abierto? Eso no es seguro, ¿verdad?

Entonces, en lugar de intentar cifrar parcialmente esa solicitud, como lo hace el resumen, en realidad vale la pena configurar HTTPS Listener. Entonces, si toda su conversación a nivel de protocolo está encriptada, no solo HTTP, sino HTTP con SSL, entonces está bien. Proporcionará su nombre de usuario y contraseña como parte de su autenticación. Y se transmitirán por la red de forma encriptada, como cualquier otro tráfico que vaya de ese cliente al servidor. Entonces, realmente, solo configurar el módulo de inicio de sesión por sí solo no es lo suficientemente seguro. Debe configurar la seguridad a nivel de transporte y el HTTPS Listener también para que esta sea una forma segura de autenticación.

Otra opción es la autenticación del cliente, donde podemos pedirle al cliente que nos presente un certificado de seguridad, una firma digital, que podríamos haber registrado con nuestro archivo de billetera SSL del lado del servidor, y básicamente decir en qué firmas SSL confiamos, o cómo traducir a nuestros usuarios reales. O puede ser algo que usará para la autenticación de aplicaciones, en lugar de usuarios, donde el nombre de usuario y la contraseña parecerán menos aplicables y la firma digital probablemente tendrá más sentido. Por lo tanto, es algo que las personas suelen usar cuando configuran la seguridad, por ejemplo, cuando una aplicación, en lugar de una persona, necesita decirle al servidor que es original. Al igual que una aplicación móvil, tal vez, al iniciar sesión en el servidor en nombre de un usuario, podría usar el módulo de autenticación del cliente.

La autenticación mutua significa que está transmitiendo la autenticación en ambos sentidos. Le estás preguntando al cliente sobre la información de autenticación, como la firma digital. Pero también le estás presentando uno al cliente, intercambiando las firmas digitales entre el servidor y el cliente en ambas direcciones. Estos son una especie de módulos de inicio de sesión predefinidos que vienen listos para usar.

Una cosa interesante es que también puede crear módulos de inicio de sesión personalizados, por lo que si no está satisfecho con esta lista de opciones. Y esta es la clave de cómo aborda la creación de módulos de inicio de sesión personalizados. En realidad es bastante simple. Todo lo que necesita hacer es usar un par de métodos en un objeto de solicitud. Entonces puedes hacerlo desde cualquier servlet que quieras. Y los métodos son inicio de sesión y cierre de sesión. Y ahí tienes Entonces, básicamente, puede decidir mediante programación cómo va a iniciar sesión en un usuario y cuándo va a cerrar la sesión del usuario.

Puede usar una combinación de enfoque declarativo y programático para la configuración del módulo de inicio de sesión. Y, de hecho, esto es algo que se le pedirá que haga como parte de su ejercicio. Allí utilizará un módulo de inicio de sesión de formulario, que se define de forma declarativa; no necesita llamar a eso, para iniciar sesión con sus usuarios. Pero luego se le indicará que use el método de cierre de sesión para cerrar la sesión de los usuarios explícitamente en algún momento de esa aplicación. Es una buena idea proporcionar un mecanismo de cierre de sesión explícito para que lo use el usuario, porque hace que la aplicación sea más segura.

Ahora, echemos un vistazo a la forma de declarar roles de aplicación. Así que recuerde, los roles de la aplicación básicamente los inventa usted, el desarrollador, como parte de la configuración de su aplicación. En primer lugar, puede utilizar cualquiera de las anotaciones, como esta anotación de declaración de funciones, o el archivo web.xml o ejb-jar.xml, para describir las funciones de seguridad.

Más tarde, cuando necesite asignar estos roles de seguridad a usuarios y grupos del mundo real, tomará uno de los descriptores específicos del servidor; bueno, para el servidor WebLogic, será WebLogic XML o WebLogic ejb-jar.xml. -- y asigne el rol lógico-- nombre del rol. Aquí está... empleado, por ejemplo. Ese es solo uno de estos roles. Asignarlo a lo que quieras. Estos son sus usuarios del mundo real o grupos del mundo real. Tal vez haya un grupo del mundo real llamado empleados, o gerentes, o alguien llamado Joe, que será miembro de esa función de seguridad.

Los roles de seguridad se utilizan luego en las restricciones de seguridad. Estos son algunos ejemplos de restricciones de seguridad. El ejemplo de la izquierda muestra un servlet. Así que tienes la restricción de seguridad, la restricción de seguridad del servlet. Y básicamente dice, para acceder a ese servlet, debe ser miembro de un rol de empleado, roles permitidos. Derecha.

Ahora, podría hacer lo mismo definiendo la restricción de autenticación en un archivo web.xml. Por lo tanto, puede asignarlo a una URL o un patrón de URL. Como puede ver, podría haber varias URL diferentes allí. Y debe ser miembro de un determinado rol para acceder a estas URL.

Con Enterprise JavaBeans, algo similar: declarar roles, roles permitidos. Entonces restringes el acceso al bean. Y también puede hacerlo en un archivo ejb-jar.xml. Así que es lo mismo, en realidad, de cualquier manera que quieras hacerlo.

Una cosa más: esta configuración PermitAll, DenyAll, estas anotaciones. Lo que hacen es que te permiten decir que cierto método o métodos en ese bean, porque puedes usarlo en un nivel de bean o puedes usarlo en un nivel de método. Entonces, la anotación PermitAll significa que aunque generalmente el bean dice que debe ser miembro de un rol de empleado para acceder a los métodos en ese bean, pero un método particular llamado find en realidad permite que cualquiera lo llame. Así que lo anulas en un nivel de método.

Y puedes hacerlo al revés. Por ejemplo, puede prohibir el acceso, DenyAll, en una anotación de uso DenyAll, digamos a nivel de bean. Y luego bajas al nivel de un método particular. Y luego dices, bueno, para acceder a ese método, puedes ser miembro de ciertos roles. Entonces puede permitir roles permitidos en un método determinado. Y puedes hacerlo de forma muy selectiva, a nivel de método por método. O podrías hacerlo por todo el frijol. Por lo tanto, estas anotaciones podrían aplicarse tanto a nivel de bean como de método.

Si desea manejar la seguridad de forma programática, en lugar de mediante declaración, ciertamente puede hacerlo. Intenta hacer que la configuración de seguridad sea lo más declarativa posible. Es más cómodo, a través de las anotaciones oa través de los archivos de configuración XML. Eso depende de usted. Pero cuanto más declarativo es, más conveniente es manejarlo.

Sin embargo, hay ciertos casos en los que es posible que desee procesar la seguridad de forma dinámica y programática. Y aquí hay algunos ejemplos de por qué querrías hacerlo y luego cómo lo vas a hacer. Digamos, imagine que desea saber quién es su usuario real, como en el nombre de usuario, no lo sé, por razones de seguimiento de auditoría. Imagine que necesita registrar en la tabla quién ha creado o actualizado el registro, por ejemplo, qué usuario ha realizado la última actualización. Así que necesitas ese registro de auditoría.

Verá, el usuario de la base de datos es un poco inútil para usted, porque el usuario de la base de datos es una cuenta genérica configurada para la fuente de datos. ¿Que necesitas? Necesita un usuario de API de autenticación y autorización de Java, una definición de usuario de Java, para rastrear y registrar en esa tabla. Bueno, allá vas. Puede que tenga que comprobar quién es ese usuario.

Y hay un método muy, muy simple que podría usar en un Enterprise JavaBean. Se llama getCallerPrincipal, getName. Sí. Y luego, en una API de servidor, eso es getUserPrincipal, getName. Para que puedas averiguar quién es ese usuario. Además, hay un método llamado isUserInRole o isCallerInRole, que verificará mediante programación si el invocador es miembro de un determinado rol.

¿Por qué querrías hacer eso? ¿Qué hay de malo con los roles permitidos, porque solo puedes hacerlo declarativamente? Bueno, aquí hay un pensamiento. ¿Qué pasa si estás accediendo a una página web? Permites que ese usuario acceda a la página. Está bien. La página contiene un enlace hacia otra página. Y el usuario no puede ver esa otra página. Entonces, si un usuario intenta hacer clic en el enlace y seguirlo, obviamente obtendrá un error, porque la restricción de seguridad impide que el usuario vea esa otra página.

Pero ¿qué hay de ver el enlace? La primera página a la que accede el usuario está perfectamente bien. El usuario puede verlo. Entonces, el usuario puede ver un elemento en una página, digamos ese enlace, por ejemplo, que en realidad, se supone que el usuario no debe ver, porque no podrá usarlo de ninguna manera. Podría ser parte de una página que requiere ocultarse para un usuario, o tal vez atenuar algunos campos, según quién sea el usuario. Y obviamente, la restricción de seguridad declarativa no es suficiente, porque la restricción de seguridad declarativa opera en el nivel de una URL. Y eso no es suficiente para restringir el acceso a un fragmento dentro de una página.

Ahí vas. Tal vez ese fragmento dentro de una página tenga que estar restringido con un poco de código programático. Está comprobando si el usuario es miembro de un determinado rol. Y en base a eso, permita que el usuario vea el elemento o lo atenúe o lo oculte.

Y se puede hacer algo similar en el mundo Enterprise JavaBeans, donde tiene el método isCallerInRole, que le permite restringir el acceso a parte del algoritmo. Lo cual es un poco extraño, porque en realidad, en Enterprise JavaBean, puede dividir el método en dos, con bastante facilidad, y tener un control de anotaciones de seguridad declarativas y acceso a cualquiera de los métodos. Pero digamos, por el bien del argumento, si necesita averiguar, dentro del método EJB, quién es ese usuario desde la perspectiva de los roles de los que es miembro, bueno, isCallerInRole, una simple verificación booleana, hará el truco.

Ahora, vamos a hablar de un tema ligeramente diferente. Vamos a hablar sobre la seguridad de los servicios web. La seguridad de los servicios web es un tema diferente, porque trasciende el entorno Java EE. Los servicios web pueden estar escritos en otros lenguajes, no necesariamente en Java.

Ahora, con respecto a los servicios web, hay dos tipos: recuerde, REST y SOAP. Si es un servicio REST, es esencialmente solo un identificador de un protocolo HTTP. Por lo tanto, cualquier seguridad que haya definido para sus servlets como un componente de manejo de HTTP aún se aplica al servicio REST. En otras palabras, no hay necesidad de inventar funciones adicionales o API por el bien de los servicios REST. Podrían funcionar perfectamente con Java EE, restricciones de seguridad de Java, restringiendo el acceso a las URL a las que se asignan estos servicios REST.

Con los servicios SOAP va a ser más difícil porque, lamentablemente, los servicios SOAP no están necesariamente basados ​​en Java y, por lo tanto, no se pueden estandarizar solo a través de Java EE. Entonces, hay un conjunto de estándares relacionados con los servicios SOAP que trascienden el mundo de Java. Y es muy importante, porque los servicios SOAP no necesariamente utilizan el protocolo de transporte HTTP. Pueden utilizar otros transportes. Y puede que no esté necesariamente implementado en Java. Y por lo tanto, sus propiedades de seguridad tienen que ser propias, de su propia estandarización.

Hay un grupo, un consorcio de empresas llamado grupo OASIS, que es un guardián estándar para todo lo relacionado con los servicios SOAP. Y parte de sus estándares es un conjunto de estándares relacionados con la seguridad de los servicios, a saber, WS-Policy, SecurityPolicy, WS-Security, etcétera. Hay bastantes estándares diferentes que define el grupo OASIS. Entonces, el estándar de seguridad es parte de ese panorama más amplio.

Echemos un vistazo a qué es exactamente lo que nos permite hacer el estándar de seguridad de servicios web. Bien, imagina un mensaje: un mensaje SOAP. Ese mensaje SOAP, estoy seguro, se transmite sobre algún protocolo de transporte. Y sí, podríamos haber protegido el mensaje al nivel de un transporte. Digamos que se encuentra dentro de un paquete HTTP, ¿verdad? HTTP, y estoy seguro de que podríamos haber cifrado ese paquete HTTP.

Pero si recibe ese paquete HTTP en un servidor, ¿qué va a hacer primero? Su servidor lo descifrará, ¿verdad? Está pasando un límite de HTTPS Listener. El paquete se va a descifrar. Entonces, está ingresando a su aplicación ese paquete SOAP en forma no cifrada.

Y si necesita manejarlo y tal vez pasarlo a otro servicio, bueno, ya no está encriptado. Y está completamente a merced de la capacidad de otro servidor para cifrar el transporte. ¿Y controlas otro servicio externo? Realmente no. Si es un servicio externo, realmente no sabes qué tan segura es su conexión de transporte.

Entonces, lo que puede hacer es ignorar el protocolo de transporte e incorporar tokens de seguridad directamente en los paquetes SOAP, y también encriptar los bolsillos SOAP total o parcialmente. Para que puedas firmarlos. Y puedes encriptarlos. Y a diferencia del bolsillo de transporte, que se descifra automáticamente, los paquetes SOAP no se descifran automáticamente. Debe decir conscientemente que desea descifrar ese paquete.

Entonces, ¿qué tal si... imagina que tu usuario está llamando a un servicio web, alguna aplicación, llamando a un servicio web? Y está pasando información sobre... bueno, una tarjeta de crédito. Así que está este pedido y contiene información de la tarjeta de crédito. Lo que puede hacer con SOAP es que puede tomar esa parte del cuerpo de un mensaje SOAP que contiene la tarjeta de crédito y codificar solo esa parte. Y no tienes que descifrarlo. Eso significa que un mensaje se puede descifrar en general, pero mantener partes cifradas en él, luego pasar a través de su aplicación de manera segura sin que la aplicación sepa lo que hay dentro. Así que eso es realmente una especie de protección adecuada de esa información.

El mecanismo exacto que usaría para implementar tokens de seguridad o cifrados depende de usted. Hay varios mecanismos diferentes que puede usar: certificados X.509, tokens SAML, Kerberos, etcétera. Hay bastantes mecanismos diferentes. Y nuevamente, eso se reduce al punto de que los servicios web se diseñan como un tipo de componentes portátiles y universales que no dependen de ningún lenguaje de programación o entorno en particular. Por lo tanto, deben admitir una variedad de posibles tokens de seguridad.

En Java, si desea cifrar, firmar digitalmente o descifrar mensajes SOAP, dado que los mensajes SOAP son XML, puede utilizar el cifrado XML y las API de firma XML. Son una especie de API de bajo nivel que manejan la incorporación de tokens de seguridad, la validación de tokens de seguridad, el cifrado y descifrado de cualquier dato XML, incluidos, obviamente, los paquetes SOAP.

Ahora, no hay un estándar Java EE sobre cómo se hace eso. Java EE le proporciona las API. Hay un estándar SOAP, cómo se hace eso, relacionado con el protocolo SOAP. Pero Java EE no nos brinda soporte listo para usar para cifrar automáticamente o firmar digitalmente o descifrar o validar paquetes SOAP. Tienes que codificarlo.

Sin embargo, existen herramientas de terceros que pueden hacer este trabajo, entre ellas, digamos, por ejemplo, un producto de Oracle llamado OWSM, Oracle Web Services Manager. Pero eso es un producto patentado. Eso no es parte de la especificación Java EE 7. Es algo que debe instalar y conectar a su servidor adicionalmente. Y eso podría serle de gran ayuda en la automatización o en el proceso de protección de los servicios web. De lo contrario, con el cifrado XML y las API de firma XML, tendrá que escribir ese código que cifra, valida o descifra sus paquetes SOAP usted mismo.

Entonces, al final de la sesión, analizamos un par de partes importantes de la seguridad en nuestra aplicación. Uno era la seguridad general de Java EE. Y el otro era la seguridad de los servicios web. En general, Java es el mundo de la seguridad, donde estamos hablando de autenticación, autorización, protección de las comunicaciones y el hecho de que puede hacerlo tanto de forma declarativa como programática.

Bueno, en términos de sus prácticas, se le pedirá que configure su servidor de aplicaciones: usuarios del mundo real, grupos. Defina el módulo de inicio de sesión, defina algunos roles de seguridad de la aplicación, asígnelos a los grupos y usuarios del mundo real. Y luego también agregue algunos bits de seguridad programáticos dentro de la parte de su aplicación web, así que restrinja el acceso a ciertos bits dentro de las páginas también. Ah, y use una extracción del nombre de usuario actualmente conectado. En realidad, podría ser bastante divertido si consideras la aplicación de chat. Ahora verá quiénes son los usuarios que ingresan al chat. Bien, ahí tienes.