TEMA 126. LA SEGURIDAD EN EL NIVEL DE APLICACIÓN. TIPOS DE ATAQUES Y PROTECCIÓN DE SERVICIOS WEB, BASES DE DATOS E INTERFACES DE USUARIO.

Actualizado en 2022
11/01/2022

CONTENIDO

1. La seguridad a nivel de aplicación

La implementación de aplicaciones, cada vez más orientadas a sistemas abiertos, hace más necesario que, además de tener en cuenta cuestiones de seguridad a nivel de infraestructura o comunicaciones, sea necesario tener la seguridad en mente en el desarrollo de aplicaciones desde el comienzo de los proyectos.

Esta gestión debe ser multidisciplinar en cada una de las capas implicadas. En este tema resumimos las implicaciones en cuanto a la interfaz de usuario, la protección de los servicios web o la base de datos.

2. seguridad en interfaces de usuario

La interfaz de usuario es el elemento de interacción entre el usuario y el sistema, y es preciso contemplar cuestiones de seguridad en su definición.

2.1 Autenticación en aplicaciones o servicios

Autenticación es el proceso por el que el usuario acredita su identidad a través de unas credenciales (algo que se tiene, como una tarjeta, algo que se sabe, como una contraseña, o algo que se es, como un elemento biométrico). Por lo general, se considera segura una combinación de dos de estos elementos, por ejemplo, una tarjeta criptográfica combinado con un elemento biométrico como la huella dactilar.

Las posibles amenazas en este elemento tienen que ver con el robo de identidad de una persona, es decir, identificarte ante un sistema como quien no eres, o en la suplantación de una sesión activa de otro usuario.

2.2 Autorización en el sistema

La autorización es el proceso por el que el sistema comprueba que un usuario tiene permiso para realizar la operación que ha solicitado. Las dos amenazas a evitar son la concesión de permisos superiores a los que debe tener un usuario, así como la denegación de permisos a un usuario que deba tenerlos. La gestión de autorizaciones se suele implementar mediante distintos mecanismos como las listas de control(ACL), o los modelos de control basados en roles (RBAC), o atributos (ABAC).

2.3 Gestión de excepciones

Las excepciones son comportamientos no esperados de los sistemas. En el caso de la interfaz de usuario, ante una excepción en la operativa, es conveniente que este hecho no proporcione al usuario información interna del sistema, que un usuario malintencionado pueda aprovechar para atacar al mismo.

2.4 Validación de datos de entrada

Un elemento crítico es la validación de datos proporcionados por el usuario a la interfaz de usuario. Es un elemento que suele obviarse en una implementación deficiente de un sistema y que puede provocar que, a través de la introducción de datos inválidos, un atacante interactúe con un sistema de forma no

deseada, con objeto de obtener información a la que no está autorizado, suplantar a otro usuario, uso de inyección SQL, ataques de denegación de servicio, etc.

2.5 Trazabilidad y auditorías

Para la identificación de operativas incorrectas es conveniente que todas las capas de un sistema guarden información de excepciones o errores que permitan trazar su origen y, en su caso, corregir posibles defectos del software.

3. Seguridad en aplicaciones web

3.1. OWASP (Open Web Application Security Project)

La Fundación OWASP es un organismo sin ánimo de lucro formado por empresas, organizaciones educativas y particulares de todo el mundo. Es una comunidad de seguridad informática que trabaja para crear artículos, metodologías, documentación y herramientas que se liberan y pueden ser usadas gratuitamente. Publica y actualiza cada 3 años el proyecto **OWASP Top 10** que es un documento donde se recogen los diez riesgos de seguridad más importantes en aplicaciones Web.

3.2. Amenazas y ataques en aplicaciones web

Los diez riesgos más críticos en Aplicaciones Web según OWASP Top 10 - 2021 son:

INYECCIÓN

Se produce al enviar a un intérprete datos no confiables a través de consultas SQL, NoSQL, OS o XPath. Los datos dañinos del atacante pueden engañar al intérprete para que acceda a información no autorizada o para que se ejecuten comandos involuntarios. Una aplicación es vulnerable a este tipo de ataques cuando los datos suministrados por el usuario no son validados y filtrados correctamente por la aplicación.
Las herramientas de análisis estático (SAST) para el análisis del código fuente y las pruebas dinámicas (DAST) se usan antes del despliegue de la aplicación en producción para identificar errores de inyección.

Tipos de inyección:

- **Inyección SQL:** Insertar caracteres especiales SQL en los campos de entrada de una aplicación Web con el objetivo de acceder a una cuenta o modificar los datos.
- **Inyección SQL ciega (*Blind SQL*):** Obtener información sensible realizando consultas en las que las únicas opciones de respuesta posibles son verdadero o falso.
- **Inyección en base de datos NoSQL:** Debido a que estos ataques de inyección NoSQL pueden ejecutarse dentro de un lenguaje de procedimiento, en lugar de en el lenguaje declarativo de SQL, los impactos potenciales son mayores que la inyección de SQL tradicional.
- **Inyección LDAP (& XPath):** La inyección de LDAP se produce cuando la entrada del usuario no está correctamente validada y se utiliza como parte de un filtro LDAP generado dinámicamente. Esto resulta en una posible manipulación de las declaraciones LDAP realizadas en el servidor LDAP permitiendo ver, modificar u omitir las credenciales de autenticación. Los ataques XPath

Injection ocurren cuando un sitio web usa la información que introduce el usuario para construir una consulta (query) xpath para los datos XML, para desconcertar al servidor cuando analiza el sentido de la consulta XPath, provocando la revelación de contenido XML al cual el cliente no debería tener acceso.

Para prevenir inyecciones se requiere:

- Separar los datos de los comandos y las consultas.
- Minimizar los permisos de la aplicación Web en la Base de Datos.
- Filtrar palabras SQL reservadas (y meta caracteres).
- Utilizar consultas parametrizadas y procedimientos almacenados.
- Realizar validaciones de entradas de datos en el servidor utilizando "listas blancas".
- Limitar mensajes de error.
- Utilizar firewalls de aplicación (WAF).
- Utilizar LIMIT y otros controles SQL.

PÉRDIDA DE AUTENTICACIÓN

La fijación de sesión es un ataque conocido que **permite secuestrar una sesión válida de un usuario**. El ataque explora una limitación en la forma en que la aplicación web suministra los ID de sesión. Si la aplicación Web al autenticar a un usuario no asigna un nuevo ID de sesión válido, se puede inducir a un usuario a autenticarse con ese ID de sesión y luego secuestrar la sesión validada por el usuario.

Posibles defensas ante estos ataques:

- Implementar autenticación multifactor.
- No utilizar credenciales por defecto.
- Implementar controles y políticas de longitud, complejidad y rotación para evitar contraseñas débiles.
- Utilizar mensajes genéricos en todas las salidas de información.
- Asegurar que, no se permiten ataques de enumeración de usuarios.
- Limitar o incrementar el tiempo de respuesta de cada intento fallido de inicio de sesión.
- Registrar todos los fallos de inicio de sesión.
- Utilizar un gestor de sesiones que genere un nuevo ID de sesión aleatorio.
- El Session-ID no debe incluirse en la URL.

EXPOSICIÓN DE DATOS SENSIBLES

Defensas ante los ataques de exposición de datos sensibles:

- Clasificar los datos procesados, identificando qué información es sensible y aplicando los controles adecuados.
- No almacenar datos sensibles innecesariamente.
- Cifrar todos los datos sensibles
- Cifrar todos los datos en tránsito
- Aplicar el cifrado utilizando directivas como HSTS.
- Utilizar únicamente algoritmos y protocolos estándares y fuertes.
- Deshabilitar el almacenamiento en caché de datos sensibles.
- Almacenar contraseñas utilizando funciones de hashing.
- Verificar la efectividad de las configuraciones y parámetros de forma independiente.

ENTIDADES EXTERNAS XML (XXE)

Muchos procesadores XML antiguos o mal configurados evalúan referencias a entidades externas en documentos XML. Las entidades externas pueden utilizarse para revelar archivos internos mediante la

URI o archivos internos en servidores no actualizados, escanear puertos de la red LAN, ejecutar código de forma remota y realizar ataques de denegación de servicio (DoS).
Defensas ante los ataques XXE:

- Utilizar formatos de datos menos complejos como JSON y evitar la serialización de datos confidenciales.
- Actualizar los procesadores y bibliotecas XML que utiliza la aplicación.
- Utilizar validadores de dependencias.
- Actualizar SOAP a la versión 1.2 o superior.
- Deshabilitar las entidades externas de XML y procesamiento DTD en todos los analizadores sintácticos XML.
- Implementar la validación de entrada positiva en el servidor ("listas blancas"), filtrado y sanitización para prevenir el ingreso de datos dañinos dentro de documentos, cabeceras y nodos XML.
- Verificar que la funcionalidad de carga de archivos XML o XSL valida el XML entrante, usando validación XSD o similar.
- Realizar revisiones manuales de código fuente y utilizar herramientas para el análisis estático de código (SAST) que ayuden a detectar XXE.
- Utilizar Firewalls de Aplicaciones Web (WAFs) o gateways de seguridad de API para detectar, monitorizar y bloquear ataques XXE.

PÉRDIDA DE CONTROL DE ACCESO

Si no se aplican correctamente las restricciones sobre lo que los usuarios pueden hacer, un atacante puede acceder de forma no autorizada a funcionalidades y datos, cuentas de otros usuarios, ver archivos sensibles, modificar datos, cambiar permisos de acceso, etc.

Prevención para la pérdida de control de acceso:

- La política debe ser denegar todo de forma predeterminada, a excepción de los recursos públicos.
- Se debe minimizar el uso de intercambio de recursos HTTP de origen cruzado (CORS).
- Se debe deshabilitar el listado de directorios del servidor web.
- Registrar errores de control de acceso que alerten a los administradores.
- Limitar la tasa de acceso a las APIs
- Los tokens json (JWT) deben ser invalidados tras la finalización de la sesión.
- Incluir pruebas de control de acceso en las pruebas unitarias y de integración.

CONFIGURACIÓN DE SEGURIDAD INCORRECTA

Establecer una configuración por defecto o directamente una falta de configuración.
Prevención para evitar configuraciones de seguridad incorrectas:

- Implementar procesos seguros de instalación sin funcionalidades innecesarias.
- Eliminar o no instalar frameworks y funcionalidades no utilizadas.
- Entornos de desarrollo, preproducción y producción configurados de manera idéntica y con diferentes credenciales para cada entorno.
- Las aplicaciones deben tener arquitecturas segmentadas que proporcionen una separación efectiva y segura entre componentes.
- La aplicación debe enviar directivas de seguridad a los equipos cliente.
- Utilizar procesos automatizados para verificar la efectividad de los ajustes y configuraciones en todos los entornos.

CROSS SITE SCRIPTING (XSS)

Una aplicación envía datos no confiables al navegador web sin una validación y codificación apropiada. Los ataques XSS permiten ejecutar comandos en el navegador de la víctima mediante *javascript* y el atacante puede secuestrar una sesión, modificar (*defacement*) los sitios web, o redireccionar al usuario hacia un sitio malicioso.

Tipos de *Cross Site Scripting*:

- XSS persistente o directo: consiste en embeber código HTML peligroso en sitios que lo permitan por medio de etiquetas <script> o <iframe>. Es la más grave de todas ya que el código se queda implantado en la web de manera interna y es ejecutado al abrir la aplicación Web.
- No persistente o reflejado: no almacenan el código malicioso en el servidor, sino que lo pasan y presentan directamente a la víctima.
- XSS basados en DOM (type-0 XSS): manipula el código y variables de JavaScript con los que está construida una página Web provocando que el código del lado del cliente se ejecute de manera "inesperada".

Defensas ante los ataques XSS:

- Utilizar frameworks seguros.
- Realizar validaciones de los datos de entrada.
- Codificar los datos de peticiones HTTP no confiables en los campos de salida HTML.
- Proteger las cookies.
- Aplicar codificación sensible al contexto.
- Habilitar una Política de Seguridad de Contenido (CSP).
- Implementar políticas del mismo origen (SOP).

DESERIALIZACIÓN INSEGURA

La serialización (*marshalling*) es un concepto que implica convertir datos de un formato a otro que permita su transmisión o guardado. La deserialización insegura ocurre cuando una aplicación recibe objetos serializados dañinos y estos objetos pueden ser manipulados o borrados por el atacante para realizar ataques de repetición, inyecciones o elevar sus privilegios de ejecución. La deserialización insegura puede conducir a la ejecución remota de código en el servidor.

El único patrón de arquitectura seguro es no aceptar objetos serializados de fuentes no confiables o utilizar medios de serialización que sólo permitan tipos de datos primitivos. Si esto no es posible:

- Implementar verificaciones de integridad como firmas digitales.
- Exigir el cumplimiento estricto de verificaciones de tipo de dato.
- Aislar el código que realiza la deserialización.
- Registrar las excepciones y fallas en la deserialización.
- Monitorizar los procesos de deserialización.
- Restringir y monitorizar la conexión de red desde contenedores o servidores que utilizan funcionalidades de deserialización.

USO DE COMPONENTES CON VULNERABILIDADES CONOCIDAS

Si se explota un componente vulnerable, el ataque puede provocar una pérdida de datos o tomar el control del servidor. Defensas ante el uso de componentes con vulnerabilidades conocidas:

- Se deben eliminar dependencias, funcionalidades, componentes, archivos y documentación innecesaria y no utilizada.
- Utilizar herramientas para mantener un inventario de versiones de componentes.
- Monitorizar continuamente fuentes como CVE (https://cve.mitre.org/) y NVD (https://nvd.nist.gov/) en búsqueda de vulnerabilidades en los componentes utilizados.
- Utilizar herramientas de análisis automatizados.
- Suscribirse a alertas de seguridad.
- Obtener componentes únicamente de orígenes oficiales.
- Supervisar bibliotecas y componentes que no poseen mantenimiento.
- Monitorizar, evaluar y aplicar actualizaciones.

REGISTRO Y MONITORIZACIÓN INSUFICIENTES

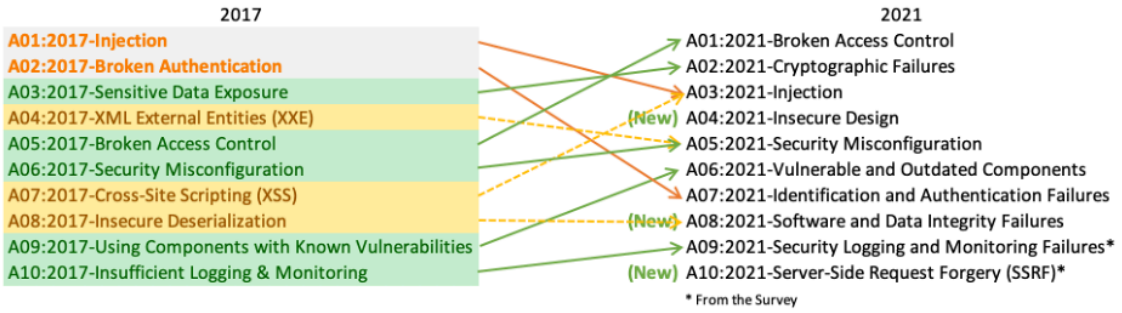
Es importante llevar a cabo un registro, monitorización y respuesta ante incidentes adecuada para evitar que un atacante pueda manipular, extraer o destruir datos.

Defensas ante el registro y monitorización insuficientes:

- Asegurar de que todos los errores de inicio de sesión, de control de acceso y de validación de entradas de datos del lado del servidor se pueden registrar
- Asegurar pistas de auditoría para todas las transacciones de alto impacto
- Establecer alertas
- Establecer o adoptar un plan de respuesta o recuperación de incidentes
- Utilizar frameworks de protección de aplicaciones comerciales y de código abierto

OWASP CUADRO RESUMEN

Los diez riesgos más críticos en Aplicaciones Web según OWASP Top 10 - 2021 son:



- **A01:2021-Broken Access Control**: El control de acceso hace cumplir la política de manera que los usuarios no puedan actuar fuera de los permisos previstos. Los fallos suelen provocar la divulgación de información no autorizada, la modificación o destrucción de todos los datos o la realización de una función empresarial fuera de los límites del usuario.
- **A02:2021-Cryptographic Failures** Nombrado anteriormente como Exposición de datos sensibles, el punto clave es la criptografía utilizada o la falta de la misma, tanto en datos almacenados como en tránsito.
- **A03:2021-Injection** Algunas de las inyecciones más comunes son SQL, NoSQL, comando OS, Object Relational Mapping (ORM), LDAP y Expression Language (EL) o Object Graph Navigation Library (OGNL). El concepto es idéntico entre todos los intérpretes. La revisión del código fuente es el mejor método para detectar si las aplicaciones son vulnerables a las inyecciones. Se recomienda encarecidamente realizar pruebas automatizadas de todos los parámetros, cabeceras, URL, cookies, JSON, SOAP y entradas de datos XML. Las organizaciones pueden incluir herramientas de pruebas de seguridad de aplicaciones estáticas (SAST), dinámicas (DAST) e interactivas (IAST) en la cadena de producción de CI/CD para identificar los fallos de inyección introducidos antes del despliegue en producción.
- **A04:2021-Insecure Design** El diseño inseguro es una categoría amplia, creada en 2021, que representa diferentes debilidades, expresadas como "diseño de control ausente o ineficaz". El diseño inseguro no es el origen de todas las demás categorías de riesgo del Top 10. Hay una diferencia entre diseño inseguro e implementación insegura. Diferenciamos entre defectos de diseño y defectos de implementación por una razón, tienen diferentes causas de origen y remediación. Un diseño seguro puede seguir teniendo defectos de implementación que den lugar a vulnerabilidades que puedan ser explotadas. Un diseño inseguro no puede arreglarse con una implementación perfecta, ya que, por definición, los controles de seguridad necesarios nunca se crearon para defenderse de ataques específicos. Uno de los factores que contribuyen a un diseño inseguro es la falta de un perfil de riesgo empresarial inherente al software o al sistema que se está desarrollando y, por tanto, la incapacidad de determinar qué nivel de diseño de seguridad se requiere.
- **A05:2021-Security Misconfiguration** Sin un proceso concertado y repetible de configuración de la seguridad de las aplicaciones, los sistemas corren un mayor riesgo. La antigua categoría de A4:2017-Entidades externas XML (XXE) forma parte ahora de esta categoría de riesgo
- **A06:2021-Vulnerable and Outdated Components** Los componentes vulnerables son un problema conocido que nos cuesta probar y evaluar el riesgo, y es la única categoría que no tiene ninguna Enumeración de Debilidades Comunes (CWE) asignada a los CWE incluidos, por lo que se utiliza una ponderación de exploits/impacto por defecto de 5,0. Los CWE más destacados que se incluyen son el CWE-1104: Uso de componentes de terceros sin mantenimiento y los dos CWE de los Top 10 de 2013 y 2017.

- **A07:2021-Identification and Authentication Failures** Conocida anteriormente como Broken Authentication, la confirmación de la identidad del usuario, la autenticación y la gestión de la sesión son fundamentales para protegerse de los ataques relacionados con la autenticación.
- **A08:2021-Software and Data Integrity Failures** Una nueva categoría para 2021 se centra en hacer suposiciones relacionadas con las actualizaciones de software, los datos críticos y los conductos de CI/CD sin verificar la integridad. Los fallos en la integridad del software y los datos están relacionados con el código y la infraestructura que no protegen contra las violaciones de la integridad. Un ejemplo de esto es cuando una aplicación depende de plugins, bibliotecas o módulos de fuentes, repositorios y redes de distribución de contenidos (CDN) que no son de confianza. Un canal de CI/CD inseguro puede introducir el potencial de acceso no autorizado, código malicioso o compromiso del sistema. A8:2017-Insecure Deserialization es ahora parte de esta categoría.
- **A09:2021-Security Logging and Monitoring Failures** Anteriormente conocida como A10:2017-Insufficient Logging & Monitoring, el registro, la detección, la supervisión y la respuesta activa pueden tener un gran impacto en la rendición de cuentas, la visibilidad, la alerta de incidentes y el análisis forense. El registro y la supervisión pueden ser difíciles de probar, a menudo implicando entrevistas o preguntando si se detectaron ataques durante una prueba de penetración, sin embargo son críticos.
- **A10:2021-Server-Side Request Forgery** Los fallos SSRF se producen cuando una aplicación web obtiene un recurso remoto sin validar la URL proporcionada por el usuario. Permite a un atacante coaccionar a la aplicación para que envíe una solicitud manipulada a un destino inesperado, incluso cuando está protegida por un cortafuegos, una VPN u otro tipo de lista de control de acceso a la red (ACL). A medida que las aplicaciones web modernas ofrecen a los usuarios finales características convenientes, la búsqueda de una URL se convierte en un escenario común. Como resultado, la incidencia de SSRF está aumentando. Además, la gravedad de la SSRF es cada vez mayor debido a los servicios en la nube y a la complejidad de las arquitecturas.

OTROS ATAQUES

- **Desplazamientos por directorios (*path traversal*):** permiten a un usuario acceder sin control a ficheros y directorios que se almacenan fuera de la carpeta raíz de la aplicación Web.
- ***Cross-Site Tracing* (XST):** Es un tipo *de Cross Site Scripting* (XSS) que utiliza el método HTTP TRACE que se utiliza principalmente para fines de depuración repitiendo toda la información enviada por el cliente al servidor (petición "echo"). Este método viene activado por defecto en muchos servidores.
- **Cross-Site Request Forgery (CSRF):** fuerza al navegador web de una víctima que está logado en algún servicio legítimo a enviar una petición a una aplicación web vulnerable con el fin de que la petición se procese en el nombre de la víctima.
- **Remote file inclusion (RFI) y Local File Inclusion (LFI): RFI** permite a un atacante ejecutar archivos remotos alojados en otros servidores. Esta vulnerabilidad existe solamente en páginas dinámicas en PHP. **LFI)** es similar a RFI excepto en que en lugar de incluir ficheros remotos sólo se pueden incluir para ejecución ficheros alojados en el mismo servidor víctima.
- **Ataques contra SSL/TLS:** Ver temas de seguridad en redes.

3.3. METODOLOGIAS DE SEGURIDAD EN APLICACIONES WEB

Una metodología de seguridad en aplicaciones web debe tener en cuenta lo siguiente:

- La seguridad en el ciclo de vida de desarrollo.
- Realizar pruebas de caja negra.

- Realizar pruebas de caja blanca.
- Implementar firewalls de nivel de aplicación (Web Application Firewalls, WAF).

OWASP ha desarrollado un marco de trabajo abierto denominado **SAMM** (*Software Assurance Maturity Model*) que pretende ayudar a las organizaciones en el proceso de formulación y aplicación de estrategias de seguridad en el software.
El CCN-CERT ha elaborado una guía de seguridad para los entornos y las aplicaciones Web: **CCN-STIC-812**.

4. SEGURIDAD EN SERVICIOS WEB

4.1 WS-SECURITY

WS-Security (WSS) es una extensión al estándar SOAP cuyo objetivo es proporcionar mecanismos de seguridad a los mensajes SOAP. En concreto, proporcionar autenticación, integridad y confidencialidad en el intercambio de estos. Al contrario que otras normas, WSS no desarrolla estándares de nuevas tecnologías para cifrado, firma o autenticación, sino que se usa para que las partes indiquen qué mecanismos concretos de seguridad están empleando mediante extensiones XML en la cabecera de los mensajes SOAP.

Es decir, WS-Security no desarrolla una nueva tecnología de seguridad sino que especifica cómo usar una serie de tecnologías y estándares de seguridad comúnmente aceptados para autenticación (por ejemplo, aserciones SAML o un certificado X.509 con una prueba de identidad), cifrado del mensaje (XML Encryption) y firma (XML DSignature).

De esta forma un proveedor de servicios web puede recibir un mensaje SOAP y mediante la lectura de las extensiones WSS de cabecera del mensaje SOAP identificar qué mecanismos ha escogido el emisor del mismo para autenticar al usuario final, cifrar el mensaje y firmarlo y de esta forma proceder a su validación. El objetivo final es ofrecer mecanismos de interoperabilidad que permitan la seguridad en los servicios web.

La especificación WS-Security fue originalmente propuesta por Microsoft, IBM y VeriSign en abril de 2002, y aprobada y estandarizada por OASIS en abril de 2004. Hoy en día WS-Security se está convirtiendo en el método principal para asegurar Servicios Web.

SEGURIDAD A NIVEL DE TRANSPORTE (TLS) Y A NIVEL DE MENSAJE (WSS)

Es importante considerar la diferencia entre WSS y TLS (Transport Layer Security). TLS ofrece un canal seguro entre dos extremos de nivel de transporte; normalmente el navegador del usuario y el servidor web si se emplea con HTTP (HTTPS). Por su parte, WSS está enfocado a garantizar la seguridad del mensaje SOAP de nivel de aplicación por sí mismo, independiente del canal de transporte que podría ser TCP o no. Además, el mensaje SOAP podría también mediante extensiones WSS incluir varias credenciales de autenticación que la aplicación proveedora identificaría e interpretaría por separado de acuerdo con su propia lógica. Las componentes de seguridad del mensaje SOAP permanecen independientemente del destino final del mensaje y del número de intermediarios. TLS, en cambio

ofrece mecanismos seguros para el nivel de transporte, pero su alcance finaliza en el punto de terminación de la conexión a nivel de transporte TCP. Por ello, ambos protocolos son compatibles. Así, si se utiliza SOAP sobre HTTP se podrían asegurar la integridad y confidencialidad de los mensajes SOAP mediante TLS, pero sólo durante una conexión TCP; en los casos de web services multi-hops que impliquen la transmisión de la identidad de usuario final o múltiples identidades, o que requieran asegurar la integridad del mensaje en toda la cadena, no bastaría con TLS.

A continuación, se resumen las diferencias entre los dos mecanismos:

TLS	*WSS*
Extremo a extremo de nivel de transporte (IP+puerto)	Extremo a extremo de nivel de aplicación.
Credenciales de autenticación de navegador y servidor únicos sin intermediarios.	Credenciales de autenticación de usuario final. Permite múltiples credenciales de autenticación.
No es flexible: se aplica a todo el payload de igual manera durante toda la sesión	Muy flexible: puede aplicarse sólo a una parte del mensaje y discriminar entre Requests y Responses
Se sitúa entre el nivel TCP y el de aplicación	La misma seguridad se puede aplicar en diferentes tecnologías de transporte.

Tecnologías de autenticación, confidencialidad e integridad soportadas por WSS

Como se ha comentado, la norma WSS define extensiones en la cabecera SOAP para la autenticación, integridad y confidencialidad del mensaje. Estas extensiones, al final, están ligadas a las tecnologías de seguridad que se emplean. A continuación, se definen las tecnologías soportadas:

- **Integridad**: WSS emplea **XML Signature** (W3C) para la firma de los mensajes SOAP. Permite firmar cualquier documento XML incluido, naturalmente, mensajes SOAP en su totalidad o sólo partes de él. Para más información consúltese el tema sobre tecnologías de firma electrónica. Un mensaje SOAP puede llevar ninguna, una o múltiples firmas XML Signature.
- **Confidencialidad**: se basa en la especificación **XML-Encryption** (W3C) que al igual que ocurre con XML Signature permite encriptar parte o la totalidad del mensaje.
- **Autenticación**: WS-Security utiliza extensiones XML en la cabecera denominadas *tokens de seguridad.* El estándar WS-Security es muy flexible ya que define extensiones para muchos tipos de tokens posibles y también permite extender el modelo. Algunos ejemplos de tokens disponibles son **username/password**, certificados **X.509**, tickets **Kerberos**, aserciones **SAML**, tokens XrML, tokens XCBF o referencias a URIs. El uso de SAML permite también intercambiar aserciones de autorización para usuarios del servicio.

Ejemplo de extensiones WSS

Para su comprensión gráfica se muestra a continuación un esquema de mensaje SOAP con extensión WS-Security. En este caso particular sólo habría una credencial de autenticación que sería el nombre/clave del usuario final, pero, como se ha comentado, podrían incluirse varios tokens de distintas tecnologías (aserciones SAML, Kerberos, certificados X.509 o incluso definidos por el propio usuario) o ninguno, de la misma forma que en el ejemplo sólo se muestra una firma, pero se podrían incluir varias o ninguna. Asimismo, se ha incluido un sólo elemento *ReferenceList* con referencias a segmentos del mensaje que estén cifrados pero se podrían incluir varios o ninguno.

```
<S:Envelope>
    <S:Header>
        <wsse:Security>
            <!– Tokens de seguridad (autenticación)-->
            <wsse:UsernameToken>
                <wsse:Username>Juan</wsse:Username>
                <wsse:Password>SoyEspartaco07</wsse:Password>
            </wsse:UsernameToken>

            <!– Firma XML Signature -->
            <ds:Signature>
                ...
                    <ds:Reference URI="#body">
                ...
            </ds:Signature>

            <!-- Referencias XML Encryption -->
                <xenc:ReferenceList>
                    <xenc:DataReference URI="#body"/>
                </xenc:ReferenceList>

        </wsse:Security>
    </S:Header>

    <S:Body>
        <!– Cuerpo del mensaje SOAP (Body) XML encriptado -->
        <xenc:EncryptedData Id="body" Type="content">
            ...
        </xenc:EncryptedData>
    </S:Body>

</S:Envelope>
```

Tokens de seguridad (Autenticación)

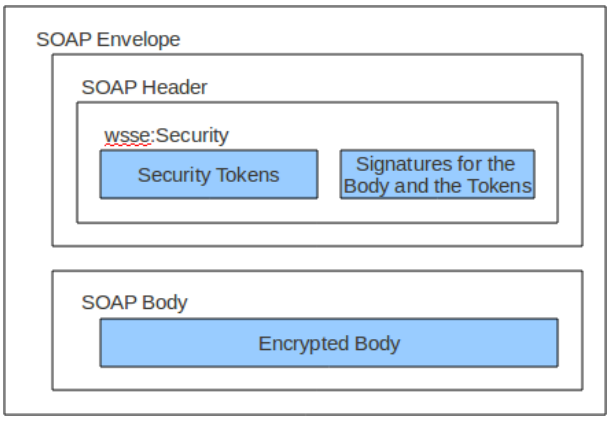
Extensiones WS-Security

Extensiones XML DSig (firma)

Extensiones XML Encryption (cifrado)

EJEMPLO DE MENSAJE SOAP CON EXTENSIONES WSS

En cuanto a la implementación, la mayoría de los frameworks de desarrollo incluyen APIs para el desarrollo de servicios web con soporte a WSS. En concreto Java ofrece el API Java API for XML Web Services (JAX-WS) y en el entorno .Net el API para construir y consumir Web Services está integrado en Windows Communication Foundation.



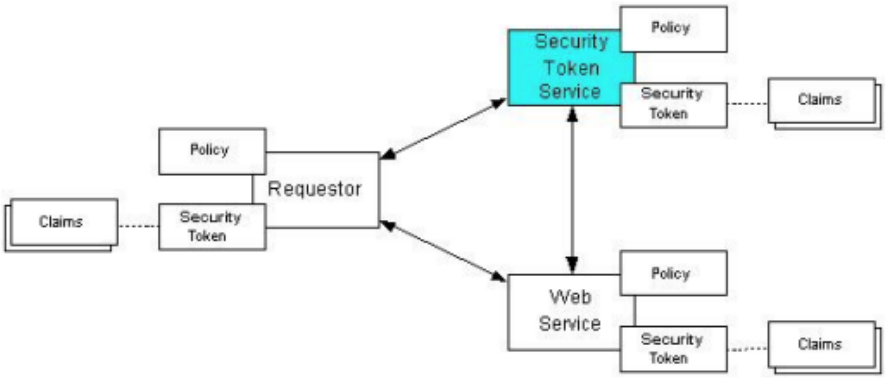
COMPONENTES DE WS-SECURITY Y EL MENSAJE SOAP

OTRAS EXTENSIONES PARA WEB SERVICES

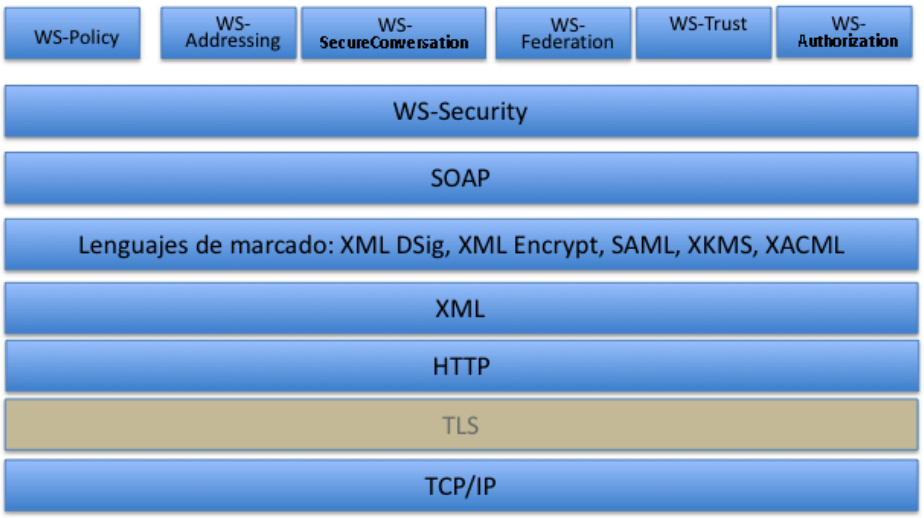
WS-Security se usa únicamente para autenticación, confidencialidad (cifrado) e integridad (firma). Existen otras extensiones SOAP **para el ámbito de la seguridad**. Su alcance es distinto del que ofrece WS-Security y por lo tanto todas son compatibles con este estándar. En concreto:

- **WS-Policy** es una serie de extensiones XML que permiten a un servicio web especificar sus requisitos de calidad de servicio y de seguridad para que puedan ser consultadas antes de invocar la interfaz WS. Las extensiones para especificar la tecnología aceptada de seguridad se denominan **WS-SecurityPolicy** y permiten definir qué tokens de autenticación se admiten, qué partes deben estar encriptadas, qué partes firmadas etc.
- **WS-Addressing**. Extensiones XML para la cabecera de los mensajes SOAP que proporciona mecanismos neutrales, independientes del protocolo de transporte, para direccionar mensajes y servicios web. WS-Addressing convierte a los mensajes SOAP en unidades autónomas de comunicación mediante la inclusión en la cabecera de los mensajes información sobre:
  - De dónde viene
  - La dirección de destino (Web Service destino)
  - Destinatario final del servicio: sistema o usuario específico dentro del servicio web de esa dirección de destino a quien va dirigido.
  - Dónde debería ir si no puede ser remitido como estaba previsto
- **WS-Trust**. La especificación WS-Trust define cómo intercambiar tokens de seguridad con credenciales de acceso a servicios web de distintos dominios de confianza. Define un modelo de servicio denominado Security Token Service (STS) y un protocolo para solicitar y entregar tokens de seguridad soportados por WS-Security y que los proveedores de servicios web especifican mediante WS-SecurityPolicy.

  El modelo considera 3 actores: el consumidor del servicio web (Requestor), el proveedor del servicio web (Web Service) y un tercero de confianza denominado Servicio de Tokens de Seguridad (Security Tokens Service). De acuerdo con el modelo, si el Requestor no posee credenciales válidas para acceder a un servicio web concreto (no cumple con la política de seguridad) puede solicitar al STS que le haga entrega de las credenciales necesarias (tokens de seguridad). Si el Requestor cumple la propia política de seguridad del STS, éste le entregará los tokens de seguridad necesarios para el acceso al web service demandado. La siguiente figura muestra un esquema del modelo.



- **WS-Federation**. Una Federación es un conjunto de dominios de seguridad que han acordado compartir recursos de forma segura. Está soportado por las extensiones ofrecidas por WS-Security (seguridad en el mensaje SOAP), WS-Trust (intercambio de tokens de seguridad entre distintos dominios de seguridad), and WS-SecurityPolicy (publicidad de la política de seguridad). Las federaciones están compuestas por Proveedores de recursos (Resource Providers), cada uno con su dominio de seguridad. En una Federación, un proveedor de recursos autoriza el acceso a un recurso a un usuario que ha sido avalado (autenticado y autorizado) por otro dominio. Como se ha comentado, se basa en WS-Trust pero su alcance es mayor pues está enfocado a **compartir recursos y a gestionar autorizaciones**. En concreto gestiona la identificación, el registro, la autenticación y la autorización de los usuarios; categoriza los niveles de confianza de las identidades, permite autenticarlas y asignarles atributos para su uso en los servicios web de todos los colaboradores federados.



Pila de protocolos sobre la que se sustentan las extensiones Web Service

- **WS-SecureConversation**: Extensiones a WS-Security para el establecimiento de contextos de seguridad para el intercambio de una serie de mensajes SOAP durante toda la sesión que se establezca entre partes de un Web-Service (lo que se denomina una "conversación"). Se diferencia de WS-Security en que ésta norma protege cada uno de los mensajes SOAP de forma independiente, lo que introduce un elevado overhead, mientras que WS-SecureConversation cubre toda la conversación. Con ésto se pretende reducir el overhead de las comunicaciones al establecer un contexto de seguridad acordado por las partes y que aplica a todos los mensajes intercambiados en la conversación de forma análoga a como se establecen las sesiones en TLS, sólo que en esta ocasión en nivel de los mensajes SOAP.

Crítica a WSS y alternativas

Tanto la arquitectura SOA basada en Web Services SOAP como la propia serie de normas WSS se encuentran muy extendidas. En la Administración General del Estado encontramos ambos estándares en sistemas como la Plataforma de Intermediación de Datos. Se trata de tecnologías maduras que cubren una gran cantidad de escenarios de uso, independientes del contexto introducido por las redes que la sustentan y que cuentan con numerosos APIs y Frameworks de desarrollo ofrecidos por la industria. Son especialmente adecuadas para garantizar la trazabilidad y la interoperabilidad.

La principal crítica realizada a WS sobre SOAP y sus extensiones WSS es el overhead que introducen y su lentitud. Por ello en los últimos años se han popularizado las denominadas arquitecturas RESTFul. RESTFul es en realidad un concepto de arquitectura de servicios, pero comúnmente -aunque no sea muy acertado- se entiende este término como una alternativa a los servicios web basados en SOAP cuya principal característica es que ofrecen interfaces directas HTTP. Los mensajes van directamente encapsulados en el payload de este protocolo frente a los Web Services SOAP que, aunque en última instancia también se transmiten normalmente sobre HTTP, introducen la capa SOAP y además emplean extensiones XML para seguridad y señalización que los hacen aún más pesados.

Los servicios RESTFul son más ligeros y su despliegue suele ser más rápido. No ofrecen soluciones globales interoperables, como pretenden WSS y sus extensiones, pero comúnmente, para los escenarios de servicios concretos que se presentan más a menudo, ofrecen una solución más ligera y rápida en su desarrollo y en sus prestaciones. Su seguridad suele estar garantizada mediante TLS.

Para más información sobre las distintas arquitecturas orientadas a servicios se recomienda estudiar el tema correspondiente.

Tanto si se implementa la API con servicios SOAP o servicios REST, existen medidas que se pueden tomar para fortalecer la seguridad:

- Utilice tokens. Configure identidades confiables y controle el acceso a los servicios y a los recursos utilizando los tokens asignados a dichas identidades.
- Utilice métodos de cifrado y firmas. Cifre sus datos mediante un método como TLS. Solicite el uso de firmas para asegurar que solamente los usuarios adecuados descifren y modifiquen sus datos.
- Identifique las vulnerabilidades. Mantenga actualizados sus elementos de API, sus controladores, redes y su sistema operativo. Manténgase al tanto de cómo funciona todo en conjunto, e identifique las debilidades que se podrían utilizar para entrar en sus API. Utilice analizadores de protocolos para detectar los problemas de seguridad y rastrear las pérdidas de datos.
- Utilice cupos y límites. Establezca un cupo en la frecuencia con la que se puede recurrir a su API, y dé seguimiento a su historial de uso. Encontrar más solicitudes a una API puede indicar un abuso. También podría ser un error de programación, como una solicitud a la API en un bucle sin fin. Establezca reglas de limitación para proteger sus API de ataques de denegación de servicio y picos de uso.
- Utilice una puerta de enlace de API. Las puertas de enlace de API funcionan como el principal punto de control para el tráfico de las API. Una buena puerta de enlace le permitirá autenticar el tráfico, así como controlar y analizar cómo se utilizan sus API

4.2 SAML

SAML (*Security Assertion Markup Language*) es un lenguaje de marcado desarrollado por el consorcio OASIS para el intercambio de información de autenticación y autorización entre proveedores de servicios de identidad, proveedores de servicios y usuarios de los servicios. Permite el establecimiento de identidades portables, entendidas éstas como la extensión de las operaciones de autenticación y autorización de un dominio de seguridad a otro distinto con el que el primero mantiene una relación de confianza. La última versión es la 2.0 y data de 2005.

El uso más extendido de SAML es como mecanismo de Single Sign On y de federación de identidades [1] pero su alcance va más allá de la mera autenticación permitiendo intercambiar atributos de los usuarios de los servicios con el fin de establecer perfiles para procesos de autorización.

SAML se usa intensiva y extensivamente en la Administración General del Estado en servicios como Cl@ve y en su extensión europea STORK.

Características generales

- **Múltiples tecnologías de transporte**

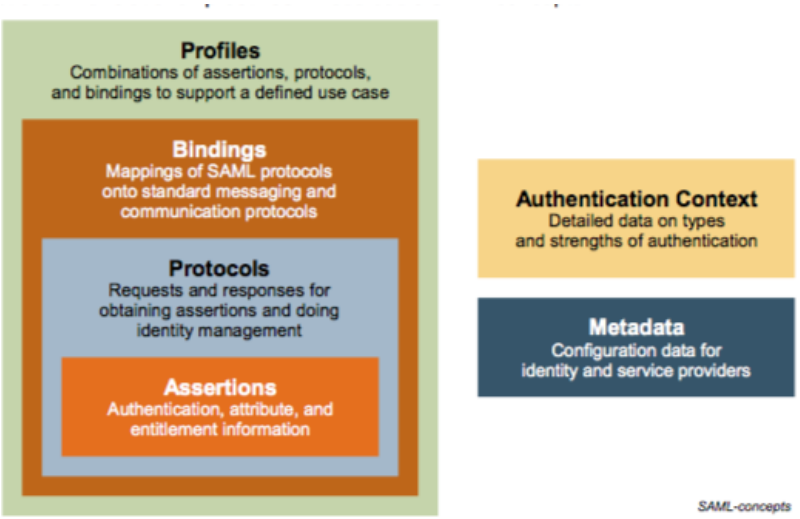
  SAML es un lenguaje de marcado basado en XML por lo que en sí es independiente del mecanismo de transporte. Los mecanismos más comunes son HTTP (es el caso de Cl@ve), y SOAP. Se puede usar sólo uno de los medios de transporte o ambos, según el escenario.

- **Distribuido**

  SAML no requiere una autoridad central. Basta con que el proveedor de servicio conozca alguna otra entidad de confianza que cuente con capacidad de autenticar al usuario para que solicite sus servicios o valide sus aserciones. No obstante, se puede usar para sistemas que cuentan con un único proveedor de servicios de identidad centralizado para un conjunto de dominios.

Mecanismos de funcionamiento

[1] Un sistema SSO es aquel en el que un usuario posee una identidad para distintos dominios de seguridad y no precisa volver a autenticarse para acceder a cualquiera de dichos dominios. Los sistemas SSO pueden disponer de un proveedor de identidad para todos los dominios o pueden ser federados. La Federación de identidades es un caso particular en el que cualquiera de los proveedores de servicio de la federación podría autenticar al usuario y éste no precisaría una nueva autenticación en el círculo de confianza. Dicho de otra forma: en una federación de identidades se establece un círculo de confianza que permite que un usuario autenticado en un organismo de la federación acceda a recursos de otro organismo de la misma federación sin autenticarse nuevamente.



CONCEPTOS BÁSICOS DE SAML: PERFILES, BINDINGS, PROTOCOLO Y ASERCIONES (OASIS)

- **Aserciones SAML**

  SAML define esquemas XML para expresar lo que denomina "aserciones" (assertions), que son verificaciones constatadas sobre un usuario. Existen 3 tipos:

  - **Autenticaciones**: verificaciones de identidad. Cuando un proveedor de servicios de identidad recibe una solicitud de autenticación de un usuario lo primero que hace es pedir a dicho usuario sus credenciales de autenticación, que procesa de acuerdo a su política. Si el proceso de autenticación es satisfactorio, el proveedor declarará mediante una aserción de autenticación que el usuario X fue identificado a través del método M en el instante T y enviará un mensaje al proveedor de servicio que le solicitó la autenticación.

    En concreto, creará un elemento *AuthenticationStatement,* que contendrá el nombre del usuario autenticado (*Subject*), el método (*AuthenticationMethod*) utilizado para la autenticación, y el instante (*AuthenticationInstant*). El proveedor de identidad puede usar múltiples mecanismos de autenticación: desde passwords a certificados X.509 pasando por tickets Kerberos.

    La siguiente figura muestra una aserción de autenticación en la que un proveedor de identidad confirma que el usuario *joeuser* se ha autenticado mediante su *password* en un instante determinado.

```
<saml:Assertion …>
  <saml:AuthenticationStatement
    AuthenticationMethod="password"
    AuthenticationInstant="2001-12-03T10:02:00Z">
    <saml:Subject>
      <saml:NameIdentifier
        SecurityDomain="smithco.com"
        Name="joeuser" />
      <saml:ConfirmationMethod>
        http://…core-25/sender-vouches
      </saml:ConfirmationMethod>
    </saml:Subject>
  </saml:AuthenticationStatement>
</saml:Assertion>
```

- **Autorizaciones**: mediante estas aserciones, las autoridades de autorización (normalmente las mismas que las de identificación) confirman o deniegan permisos de usuarios concretos ya identificados para realizar determinadas acciones. Las entidades autorizadas pueden ser usuarios humanos o sistemas.

- **Atributos**. Expresan datos sobre una entidad (normalmente personas) que permiten a los proveedores de servicio aplicar su propia política de control de acceso o de servicio en función de la respuesta. Por ejemplo, un comercio *online* podría solicitar a una entidad financiera datos sobre el límite de crédito de un usuario. La siguiente figura muestra una aserción de atributo en la que una entidad confirma que un usuario ha saldado su pago e informa de su límite de crédito.

```
<saml:Assertion …>
  <saml:AttributeStatement>
    <saml:Subject>…</saml:Subject>
    <saml:Attribute
      AttributeName="PaidStatus"
      AttributeNamespace="http://smithco.com">
      <saml:AttributeValue>
        PaidUp
      </saml:AttributeValue>
    </saml:Attribute>
  <saml:Attribute
      AttributeName="CreditLimit"
      AttributeNamespace="http://smithco.com">
      <saml:AttributeValue>
        <my:amount currency="USD">500.00
        </my:amount>
      </saml:AttributeValue>
    </saml:Attribute>
  </saml:AttributeStatement>
</saml:Assertion>
```

- **Roles**. SAML considera 3 roles:

  - **Principal**: se denomina así a la entidad que requiere ser autenticada; puede ser un usuario final u otro sistema.
  - **Proveedor de servicio (SP)**: desea autenticar, autorizar o conocer algún dato de algún usuario o sistema (principal)
  - **Proveedor de identidad (IDP)**: autentica, autoriza o envía datos de usuarios (principales).

- **Protocolos**.

  SAML define esquemas XML para la implementación de protocolos sencillos Request/Response mediante los que se pueda solicitar aserciones y responderlas. Se trata de un esquema completamente separado del de aserciones.

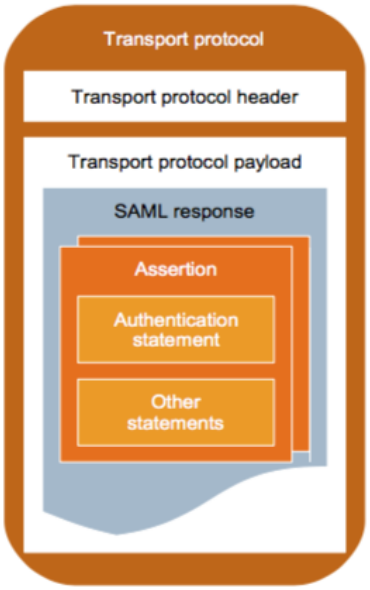
  Un requerimiento (request) contiene un claim mediante el que se solicita una aserción de autenticación, autorización o de atributos. La respuesta de la autoridad correspondiente contendrá la aserción resultante. Todo requerimiento debe contener el ID del sujeto en cuestión, que también se incluirá en la respuesta. Los requerimientos (Request) pueden ser, según se solicite, de autenticación, autorización o solicitud de atributos (*claim*).

  La siguiente tabla muestra los protocolos y su función:

Protocolo	Función
Authentication Request Protocol	Solicitud de autenticación de un usuario
Single Logout Protocol	Define la forma de realizar un logout simultáneo de las sesiones asociadas a un usuario. El logout puede ser iniciado directamente por el usuario, o iniciado por un IDP o SP debido a la expiración por tiempo o también por decisión de un administrador
Assertion Query and Request Protocol	Solicitud de aserciones. La forma de Request de este protocolo permite solicitar a un IDP un Assertion haciendo referencia a su ID. La forma de Query define como solicitar Assertions en base al usuario y el tipo de Assertion
Name Identifier Management Protocol	Provee mecanismos para cambiar el valor o formato del identificador usado para referirse al usuario. El protocolo también define la forma de terminar una asociación de la identidad del usuario entre el IDP y el SP
Name Identifier Mapping Protocol	Provee un mecanismo para poder mapear un identificador de usuario usado entre un IDP y un SP de forma de obtener el utilizado entre el IDP y otro SP.
Artifact Resolution Protocol	Permite transportar los Assertions por referencia (a esta referencia se le denomina Artifact en el estándar) en vez de valor. Esto facilita el uso de SAML en un entorno web.

- **Bindings**

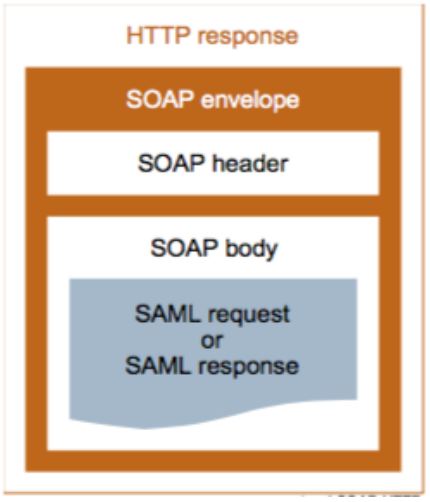
Un *binding* es especificación que indica cómo encapsular los mensajes Request/Response del protocolo SAML en el payload del mensaje de un mecanismo de transporte concreto.



ENCAPSULADO DE UN MENSAJE SAML EN EL PAYLOAD DE UN PROTOCOLO DE TRANSPORTE GENÉRICO.

SAML define numerosos bindings. Algunos de los más importantes son:

- Binding de SAML sobre SOAP (Binding SOAP): indica cómo insertar un mensaje SAML en el cuerpo (*body*) de un mensaje SOAP con indicaciones de cómo transportar a su vez el mensaje SOAP sobre HTTP. No se debe confundir con el envío de aserciones en la cabecera (*header*) de los mensajes SOAP definido por WS-Security que tienen como función proporcionar seguridad al mensaje SOAP en sí y no ofrecer un servicio de autenticación o autorización.



SAML SOBRE SOAP

- Binding de SAML sobre el método POST de HTTP. (Binding POST)
- Binding de SAML sobre mensajes HTTP de redirección (Binding Redirección).

- **Perfiles SAML**

  Los perfiles SAML definen cómo combinar aserciones SAML, protocolos y bindings para poder cubrir determinados escenarios de uso y garantizar la interoperabilidad.

  Cabe destacar:

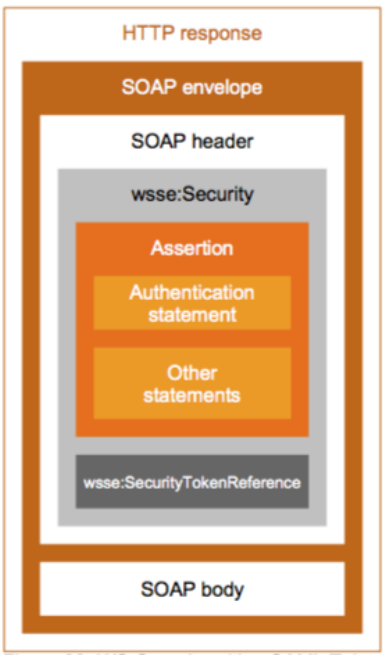
  - Perfil SSO para browsers (Web Browser SSO Profile). Pensado para ofrecer servicios de SSO a clientes con navegadores estándar.
  - Perfil mejorado de cliente y proxy (Enhanced Client and Poxy - ECP). Caso especial en que se emplea un mediador de proveedores de servicios de identidad (caso de Cl@ve).

SAML y Web Service-Security

Como se ha comentado, WSS consiste en un conjunto de especificaciones para proporcionar confidencialidad, integridad y autenticación a mensajes SOAP. Para ello se especifica un tag (*wsse*) para la cabecera de los mensajes SOAP dentro del cual se coloca la información que proporciona la seguridad al mensaje.

Al contrario que la especificación de SAML sobre SOAP, en que los mensajes del protocolo SAML se incluyen en el cuerpo del mensaje SOAP y son parte de un protocolo de autenticación y autorización, en la norma WSS sólo se especifica como insertar aserciones SAML encaminadas a proteger el mensaje SOAP en que se encuentran. Típicamente contienen claves empleadas para firmar los mensajes que una vez verificadas permiten confirmar la identidad del emisor del mensaje SOAP.

Es decir, mientras que los mensajes de SAML sobre SOAP van en el cuerpo (*body*) de SOAP y forman parte de un protocolo Request/Response de autenticación, autorización o consulta de atributos, los mensajes SAML de la norma WSS son sólo aserciones que van en la cabecera (*header*) SOAP y tienen como objetivo participar de la seguridad del mensaje SOAP en sí. Normalmente son aserciones de autenticación que van firmadas mediante XML DSignature y que permiten establecer la identidad del emisor del mensaje SOAP.



MENSAJE SOAP CON EXTENSIONES WSS QUE INCLUYE ASERCIONES SAML COMO TOKENS DE SEGURIDAD

5. SEGURIDAD DE LAS BASES DE DATOS

Otro de los elementos críticos de un sistema, son sus repositorios de datos, que generalmente se implementan a través de bases de datos relacionales, documentales o de otro tipo. Las vulnerabilidades más habituales que nos encontramos en las bases de datos son las siguientes:

- **Debilidad de credenciales**: Un usuario o una contraseña que puedan ser capturados por un atacante para acceder directamente al repositorio de datos impersonando un usuario válido.
- **Gestión de usuarios mediante grupos**: Una buena práctica es que los usuarios de una base de datos reciban sus privilegios a través de grupos, de forma que la seguridad se pueda manejar de forma colectiva.
- **Funciones no utilizadas**: Es conveniente, tanto en la base de datos, como en otro tipo de software asegurarnos que únicamente instalamos el software mínimo necesario para la operativa que tenemos. Una buena forma de evitar incorporar bugs a un sistema es no desplegar funcionalidades que no vamos a utilizar.
- **Mínimos privilegios**: Es otra buena práctica el definir para un grupo de usuarios los mínimos privilegios necesarios que precisa para operar en el sistema. No deben utilizarse más que cuando no exista otra alternativa para los usuarios administradores de la base de datos.
- **Elección y mantenimiento adecuados del SGBD**: Una elección del sistema de base de datos adecuada y un correcto y puntual mantenimiento del mismo pueden evitar problemas en la operación del mismo y riesgos de seguridad.
- **Inyección SQL**: Uno de los principales riesgos de uso de un gestor de base de datos, en particular cuando utilizamos bases de datos relacionales, es la inyección SQL. Una buena opción para protegerse del mismo es validar todas las entradas que se remiten al gestor de base de datos.