Single sign-on y federación de identidades

José María Palazón Romero
Antoni Felguera
Jordi Castellà-Roca

PID_00183948

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Desde el inicio de la informática, los diferentes sistemas han necesitado la capacidad de limitar el acceso a los recursos computacionales disponibles. Según el caso, las motivaciones han oscilado desde la vigilancia de los enormes costes de dichos recursos en los años sesenta hasta la actualidad, donde por motivos de seguridad el control de acceso a programas o datos almacenados se adivina como un requisito imprescindible.

De forma genérica, la seguridad de la información tiene tres dimensiones esenciales: la disponibilidad, la integridad y la confidencialidad. A través de la disponibilidad queremos expresar la necesidad de que los sistemas estén disponibles cuando se les requiera, garantizando así la prestación de las funciones para las cuales han sido diseñados.

Por otro lado, la integridad se refiere a la propiedad de que nadie, excepto los usuarios permitidos, puede alterar la información de otro modo que no sea el permitido por las políticas de seguridad. Por último, la confidencialidad enuncia que nadie, excepto los usuarios permitidos, puede conocer el contenido de la información protegida.

Esta limitación y control de acceso de un sujeto a los recursos ofrece concretamente características de integridad y confidencialidad a la protección de la información. A través de ella se pretende evitar que los sujetos no permitidos puedan alterar la información protegida, así como que puedan acceder a su contenido y recuperarla. Con dicho objetivo, se han definido diferentes operaciones, entre las cuales podemos encontrar la identificación del sujeto, su autenticación o el control de acceso.

Antaño, cuando los sistemas eran monolíticos y no disponían de conexión entre sí, las operaciones mencionadas eran gestionadas íntegramente por el sistema al que se quería acceder. Al conjunto abstracto formado por los diferentes usuarios, recursos, permisos y registros de acceso del sistema vinculados a la identificación y control de acceso, se le ha llamado dominio de identidad o seguridad.

A medida que los sistemas de información se conectaban entre sí, surgió la necesidad de que diferentes sujetos de diferentes dominios de identidad pudiesen acceder a recursos de otros dominios en los que se confiaba. Es en este momento cuando surge la idea de la federación de identidades, basada en el concepto de integrar y coordinar diferentes dominios de identidad entre ellos con el objetivo de formar un dominio más grande para la gestión y control del acceso, fruto de la suma de sus características individuales. La Federación de Identidades permite dar servicios complejos, pero no es menos cierto que

su evolución, como casi todo lo relacionado con la identidad digital, nació y creció de forma fragmentada, con múltiples iniciativas atomizadas que se han ido desarrollando de forma más o menos paralela, a la luz de los esfuerzos de las grandes multinacionales y agrupaciones creadas.

Adicionalmente, también cabe destacar que la Federación de Identidades no se usa regularmente a la hora de construir sistemas de información, lo cual dificulta más si cabe su adopción por parte del gran público. Únicamente acrónimos como el de *Single-Sign-On (SSO)* son más o menos conocidos como una *commodity* de mercado, pero lejos de constituir un verdadero requerimiento a la hora de planificar la construcción de un sistema de información.

En una sociedad enormemente conectada, con servicios dinámicamente compuestos, una alta fluctuación de las relaciones y una estrecha relación entre el mundo físico y virtual, la federación de identidades está llamada no a ser una opción, sino una necesidad acuciante de desarrollo de la denominada Internet del Futuro.

Objetivos

En este módulo el estudiante hallará los contenidos y herramientas que le permitirán alcanzar los objetivos siguientes:

1. Comprender el concepto de Federación de Identidades, los términos asociados y su relación con los diferentes elementos de la gestión de la identidad y acceso.

2. Adquirir el conocimiento de los diferentes patrones de Federación de Identidad que existen, basados en el establecimiento de la confianza inicial entre los pares.

3. Conocer los diferentes estándares de Federación de Identidad, además de las tecnologías asociadas y sus principales componentes y acciones.

4. Entender el funcionamiento de los métodos y tecnologías que permiten la Federación de Identidades aunque no hayan sido diseñados estrictamente para ello.

Todos estos objetivos se realizarán a través de una exposición sencilla y clara de la materia, conducida a través de unos ejemplos prácticos introducidos al inicio.

Se mostrarán, además, porciones de código de programación que ilustrarán más concretamente los hechos específicos de los protocolos más importantes. Es relevante destacar que a través de dichos ejemplos no se pretende convertir el presente módulo en un manual de programación de dichos protocolos, sino simplemente introducir al estudiante en las operaciones más importantes y su morfología concreta a bajo nivel.

1. La federación de identidades

En este apartado, y a partir de sencillos ejemplos prácticos, se introducen los conceptos previos necesarios para definir la federación de identidades, el *single sign-on*, así como los diferentes patrones de federación existentes.

En los dos siguientes ejemplos se presenta la federación de identidades como una solución a problemas reales existentes, y que servirán para explicar de forma didáctica los conceptos del módulo.

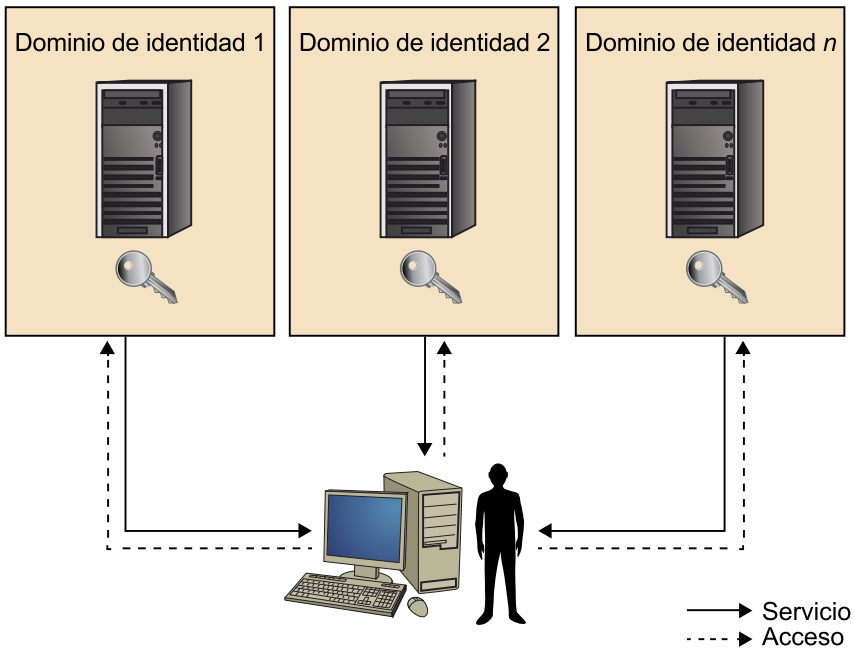
Unas cómodas vacaciones

Cuando estamos planeando unas vacaciones, no es extraño que lo primero que queramos asegurar sean los vuelos disponibles al destino que hemos elegido. Fácilmente accederíamos al sitio web de la compañía aérea seleccionada, reservaríamos nuestro viaje y nos preguntarían diligentemente sobre la existencia de un registro previo. Si lo hubiésemos completado en una ocasión anterior, habitualmente introduciríamos nuestro usuario y contraseña para que el proveedor recuperase nuestros datos. Si no, introduciríamos en aquel momento todos nuestros datos tanto personales como económicos para que la compañía aérea pudiese completar la transacción. Finalmente, confirmaríamos todos los datos de la operación y pagaríamos.

A continuación nos daríamos cuenta de que necesitamos un coche con el que, por ejemplo, desplazarnos hasta nuestro hotel. Para ello elegiríamos nuestra compañía de alquiler de coches favorita, seleccionaríamos el producto deseado y volveríamos a realizar otra vez todo el proceso de registro o recuperación de nuestros datos, confirmación y ejecución de la operación.

Si este proceso se produce varias veces, por ejemplo porque tuviésemos que contratar también un hotel en la zona, reservar entradas para unas atracciones, transportes especiales, etc., volveríamos a realizar la operación otras tantas veces, mostrando un esquema como el que se muestra en la figura siguiente.

Acceso a múltiples proveedores de servicio aislados con múltiples credenciales



Como se muestra en el esquema, el usuario tiene tantas identidades como proveedores de servicio existen cuando en realidad es la misma persona. Los proveedores de servicio

se mantienen aislados entre sí, no integrando sus propuestas como tampoco su gestión de identidades.

El usuario, por su parte, tiene que recordar tantos usuarios y contraseñas como proveedores de servicio existen y proporcionar la misma información cada vez a todos y cada uno de los proveedores de servicio con los que quiere mantener relación. Finalmente, cuando el usuario modifique sus datos personales o quiera borrarlos tendrá a su vez que cambiarlos o borrarlos en todos y cada uno de los proveedores de servicio existentes.

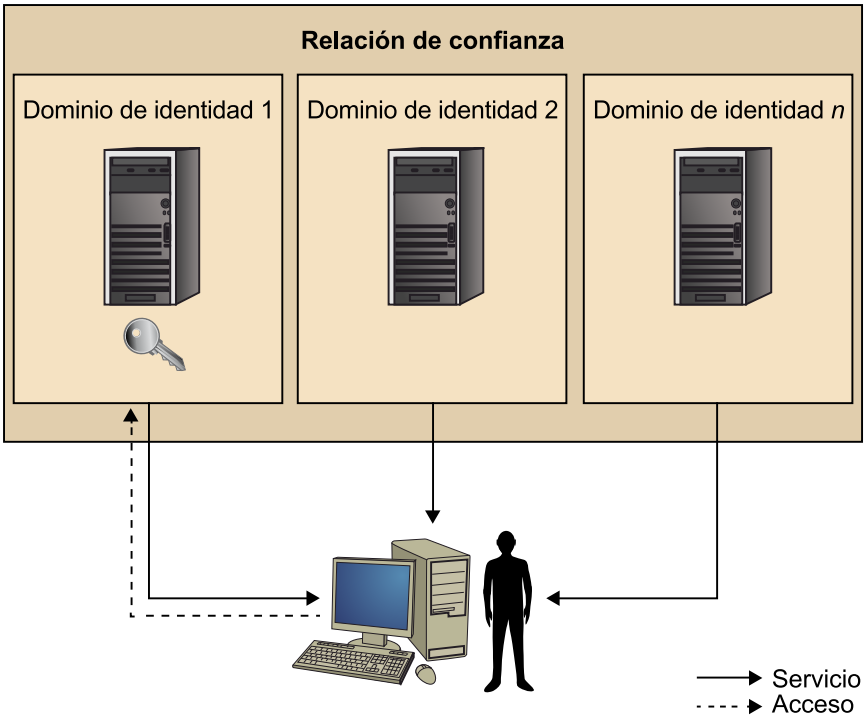
El servicio mundial de movilidad académica

La comunidad académica se ha caracterizado siempre por una gran movilidad física. Muchos profesores viajan continuamente de una universidad a otra para realizar sus estancias o impartir clases. Los estudiantes también se benefician de facilidades para la movilidad, como las becas Erasmus, que permiten el intercambio de estudiantes europeos entre Universidades del continente.

Así, fruto de esta necesidad se creó EduRoam, un servicio de movilidad segura para la comunidad académica mundial. A través de él cualquier miembro de la comunidad académica que pertenezca a una institución vinculada a dicho servicio puede viajar a cualquier otra institución e, inmediatamente, tener acceso a recursos, como por ejemplo la conexión Internet.

Como se puede ver en la figura siguiente, los usuarios (profesores, administrativos, estudiantes, etc.) no necesitan pedir a la institución de destino que les facilite credenciales, ya que disponen de unas en su organización de origen, con la cual se tiene una relación de confianza que les permite autenticar las credenciales mostradas y dar acceso inmediato a una serie de recursos, como la conexión a Internet.

Acceso a múltiples proveedores de servicio con un único juego de credenciales



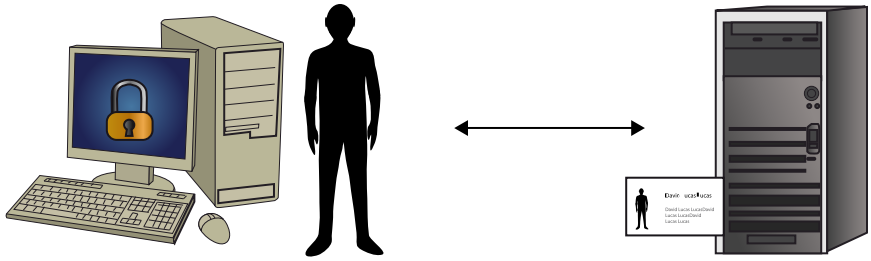
1.1. Conceptos previos para la definición de la federación de identidades

En este subapartado presentaremos los conceptos previos esenciales de la federación de identidades antes de pasar a su definición. Estos conceptos previos nos permiten afianzar y clarificar ideas que utilizaremos posteriormente de forma repetida.

1.1.1. Iniciando el camino

Aunque hablemos de federación de identidades, el concepto básico que se sigue manteniendo, el elemento esencial, es el de un sujeto que quiere acceder a un objeto de forma segura, tal y como se muestra en la figura siguiente.

Esquema conceptual base para servicios de identidad



Esta primera aproximación realizada es necesariamente muy genérica. Aunque pudiera parecer que el **sujeto** es simplemente una persona y que el **objeto** pudieran ser los datos, la verdad es que la evolución de la gestión de identidades y su aplicación en los diferentes paradigmas actuales y futuros de la computación ha generalizado estos conceptos. De forma no exclusiva, podríamos decir que:

- Un **sujeto** podría ser una persona pero también un proceso o *thread*, o un dispositivo, por ejemplo.

- Un **objeto** serían datos, pero también servicios, recursos computacionales o cualquier bien que custodie un sistema informático (físico o lógico).

- Y la expresión *de forma segura* pretende asegurar que se proveen las dos características básicas para las que está configurado: integridad y confidencialidad.

Para este requisito, la federación de identidades se nutre de las operaciones básicas de la gestión de identidades:

- **Identificación**. Pocos sistemas de identidad ofrecen la posibilidad de identificar al sujeto de forma unívoca. Entre ellos, algunos sistemas biométricos avanzados, siempre que el universo de individuos no sea demasiado grande.

- **Autenticación**. Proceso por el cual el sistema constata, a través de algún mecanismo, que el usuario es quien dice ser. Existen muchos métodos diferentes de autenticación esencialmente basados en tres aspectos: lo que se conoce, lo que se tiene o lo que se es.

Sistemas biométricos de identificación

Los sistemas biométricos de identificación se basan en las características físicas o de comportamiento de las personas. Estas características pueden ser el reconocimiento de huella dactilar, de voz, de retina, la forma de andar, etc.

- **Autorización o control de acceso**. Mecanismo a través del cual se permiten las operaciones de acceso o alteración de los bienes que se están protegiendo.

- **Registro**. Cuando accedemos a un determinado bien una vez identificados, autenticados y superado el control de acceso, existe otra operación básica llamada registro. A través de esta acción se guardan todas las operaciones que se han ejecutado sobre un determinado recurso, con el objeto de tener un cuaderno de bitácora de las acciones realizadas.

Ejemplo de autorización

Un ejemplo claro de autorización es el control que el sistema operativo ejerce al intentar acceder a nuestros ficheros, basándose en la configuración concreta de los permisos del usuario.

A los protocolos que tienen estas tres últimas operaciones se les llama protocolos AAA del inglés *authentication*, *authorization and accountability*. Los sistemas de identidad federados, al ser sistemas de identidad en sí mismos, han de disponer de las mismas operaciones que un sistema de identidad al uso. El usuario dispondrá de ellas bien sea porque las implementa el sistema específicamente, o bien porque las hereda de uno de los sistema federados.

1.1.2. La dificultad de federar identidades

Como hemos visto en los ejemplos al inicio del apartado, cuando dos entidades deciden federar sus sistemas de identidad y control de acceso, han de garantizar las operaciones básicas así como las características de seguridad que deben proveer. Es decir, cuando la compañía aérea decide federar sus sistemas con la compañía de alquiler de coches, han de proveer conjuntamente las capacidades de autenticación, registro o la de control de acceso a los recursos, que dotarán de las características de integridad y confidencialidad que hemos mencionado anteriormente. Pensemos, además, que cada uno de los sistemas puede haber sido ideado de forma diferente, con diferentes estructuras, sistemas de privilegios o conceptos base, con lo que ponerlos a funcionar juntos no es, en principio, algo obvio.

Como muestra de esta complejidad, podemos observar que únicamente para el control de acceso, por ejemplo, el mundo de la gestión de la identidad oscila entre los sistemas DAC[1] basados en una lista de acceso por recurso, a los sistemas RBAC[2] basados en roles, pasando por los sistemas MAC[3] para sistemas militares, entre otros. La fragmentación en otras operaciones, como la de autenticación, es aún mucho mayor, con multitud de métodos y técnicas con diferentes niveles de seguridad.

(1)DAC son las siglas de *discretionary access control*.

(2)RBAC son las siglas de *role-based access control*.

(3)MAC son las siglas de *mandatory access control*.

Esta continua proliferación de aproximaciones a la hora de acometer la implementación de los sistemas de gestión de acceso planteó dificultades a la hora de aproximarse a las posibles soluciones respecto a la federación de identidades. El reto estaba sobre la mesa: si todos los sistemas son tan diferentes entre ellos, ¿cómo vamos a conseguir comunicarlos entre sí?

1.1.3. Propuestas iniciales

Una vez planteado el reto, se propusieron diferentes soluciones antes de llegar hasta lo que hoy conocemos como los **sistemas de identidad federados**. Estas soluciones estaban basadas en una centralización de alguno de sus elementos clave, que permitía una coordinación posterior entre los diferentes sistemas que lo componían. No podemos llamar a estos sistemas como federados, pues realmente no constituían una federación debido a la relación jerárquica que mantenían. Concretamente, podemos clasificar las propuestas del modo siguiente:

- **Metadirectorios**. Es un repositorio de información con la copia de la información de todos los repositorios de identidad. Se sincroniza cada cierto tiempo para mantener el estado de los diferentes repositorios.

- **Directorios virtuales**. De la misma forma que los metadirectorios, son repositorios de información, aunque en lugar de tener una copia de la información de los diferentes repositorios tiene apuntadores donde realmente se encuentra la información.

Se ha escrito mucho sobre la adecuación de los sistemas centralizados a la hora de resolver los problemas expuestos. Las principales críticas estriban en la escalabilidad de la solución, dado que mantener identificadores únicos (ID canónicos únicos) puede ser una tarea altamente compleja debido al gran número de actores que, potencialmente, pueden verse involucrados.

Otra de las críticas es su adaptación a la realidad específica de organizaciones y aplicaciones, es decir, que el mantenimiento de ID canónicos únicos puede afectar no únicamente a su mantenimiento (adaptación frente a cambios) y escalabilidad, como ya hemos visto en el párrafo anterior, sino también a la capacidad de adaptación a diferentes realidades. Un entorno centralizado implica una serie de características a la hora de autenticar, de mantener un conjunto de privilegios con una determinada estructura, etc., que puede no ser adecuada a diferentes realidades. Por ejemplo, un sistema RBAC, basado en roles y muy extendido en ambientes empresariales, puede no ser adecuado en un entorno militar, donde las prioridades son otras y están muy extendidos los sistemas MAC. Es por ello por lo que se hace difícil pensar que un entorno centralizado pueda satisfacer todos los requerimientos de las diferentes realidades organizacionales que pueden existir.

Aunque un entorno centralizado constituye un único punto de control, lo que permite una focalización de esfuerzos a la hora de asegurar y administrar un sistema de información, también constituye un único punto de fallo, lo que aumenta el riesgo de no disponibilidad al conjunto de los sistemas a los que da servicio.

Todas estas razones: la escalabilidad, la adaptación a las diferentes realidades y el único punto de fallo han constituido en la práctica barreras insalvables para este tipo de sistemas, aunque en determinados entornos (por ejemplo, dentro de una única empresa con políticas homogéneas) todavía sigan vigentes. La complejidad de la federación de identidades ha sobrepasado las capacidades de este tipo de entornos y ha dado paso a diferentes aproximaciones fruto de diferentes génesis, que poco a poco van convergiendo y que mantienen como denominador común el respeto por la gestión distribuida de los recursos y, por ende, de los sistemas de identidad y acceso.

1.2. Definición de federación de identidades

Es importante, antes de seguir adelante, que definamos los diferentes conceptos que vamos a ir utilizando repetidamente a lo largo del módulo con el objeto de clarificarlos y puntualizarlos antes de que puedan conducir a cualquier tipo de equivocación.

Según hemos visto anteriormente, la federación de identidades mantiene los mismos objetivos que un sistema de identificación y gestión de acceso al uso, es decir, el de un sujeto que quiere acceder a un objeto de forma segura. Para ello ya definimos en el subapartado 1.1 los conceptos de *sujeto*, *objeto* y *forma segura*.

Estamos en disposición ahora de formalizar las definiciones que intervendrán en un sistema de identidad federado, tal y como se muestra en la figura siguiente:

- Un **sujeto** es cualquier elemento capaz de presentarse ante un sistema de identificación para reclamar una identidad y autenticarse ante él.

- Un **objeto** es un bien accesible por un sujeto a través de medios electrónicos.

- Una **tabla de capacidades** son las capacidades de acceso que tiene un determinado sujeto a los diferentes objetos protegidos por un sistema.

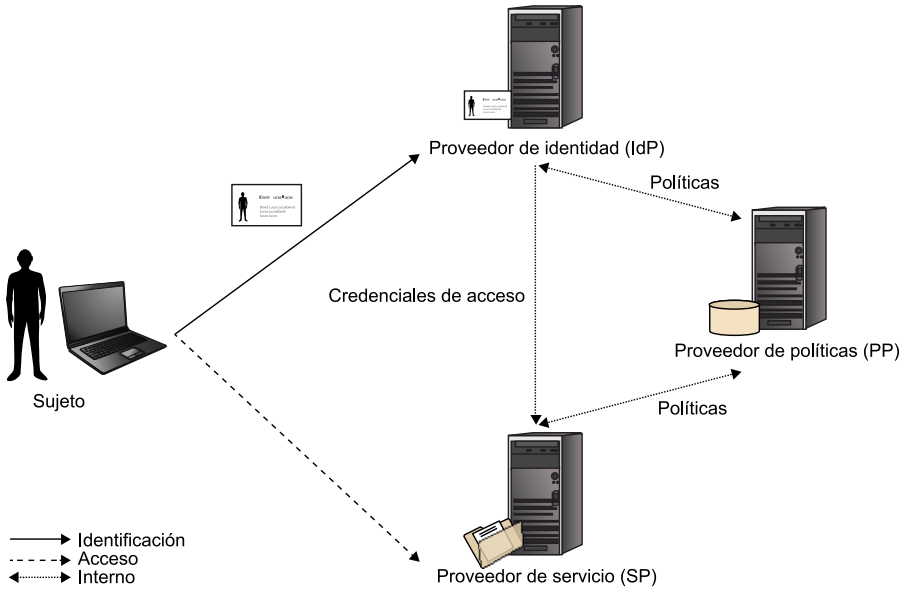
- Un **dominio de identidad (o dominio de seguridad)** es un conjunto de sujetos, objetos y mecanismos para la identificación y la gestión de acceso.

- Un **proveedor de servicio (SP)** es un sistema que provee de los objetos del sistema y los pone a disposición de los sujetos pertinentes.

- Un **proveedor de identidad (IdP)** es un sistema que provee de los mecanismos propios de un sistema de identificación y control de acceso.

- Un **proveedor de políticas (PP)** es un sistema que configura, almacena y distribuye las políticas de seguridad referidas a la identidad y control de acceso.

Diagrama de relación entre los diferentes actores de un sistema de identidad



Tal y como vemos en la figura anterior, entre el sujeto y el proveedor de identidad se realiza el proceso de identificación y autenticación a través de una aportación de credenciales por parte del sujeto. En este proceso el proveedor de políticas también interviene, ofreciendo las políticas de identidad necesarias, como por ejemplo, las restricciones horarias para la autenticación. En este momento, el proveedor de identidad certifica la identidad de un sujeto al proveedor de servicio que, de acuerdo a la tabla de capacidades de ese sujeto, permitirá o no la operación requerida.

La **federación de identidad** se puede definir como los mecanismos necesarios para permitir a una identidad perteneciente a un dominio de identidad concreto operar con objetos de un proveedor de servicio asociado a un dominio de identidad federado, de acuerdo a las políticas de seguridad establecidas para cada uno de ellos y para la federación en su conjunto.

Nota

Se están investigando en este momento sistemas de reputación y confianza que ayuden a establecer relaciones de federación de identidades sin conocimiento previo, pero todavía están en estadios muy iniciales.

Los sistemas de identidad federados pretenden, por decirlo así, reunir las diferentes identidades existentes de un determinado sujeto en múltiples dominios de identidad. Por ello, es necesario asumir que, para que haya una federación

de identidades, necesitaremos que haya más de un dominio de identidad. Si sólo hubiera un dominio de identidad y los diferentes recursos confiasen en la gestión que este dominio hace sobre la identificación y control de acceso, no estaríamos hablando de federación de identidades sino de control de acceso delegado, que también trataremos más adelante a modo de clarificación.

Es importante destacar que los sistemas de identidad federados constituyen, en sí mismos, una relación de confianza entre dos partes. Esta confianza tiene que estar previamente definida para que la tecnología pueda comunicarla después, por ejemplo, a través de diferentes tecnologías como SAML. En todo caso, no es posible tecnológicamente poder establecer *a priori* la confianza inicial, sin esa relación previa mencionada fuera del ambiente tecnológico.

Ved también

Las tecnologías SAML se describen en el subapartado 3.1 de este módulo.

1.3. Patrones de federación

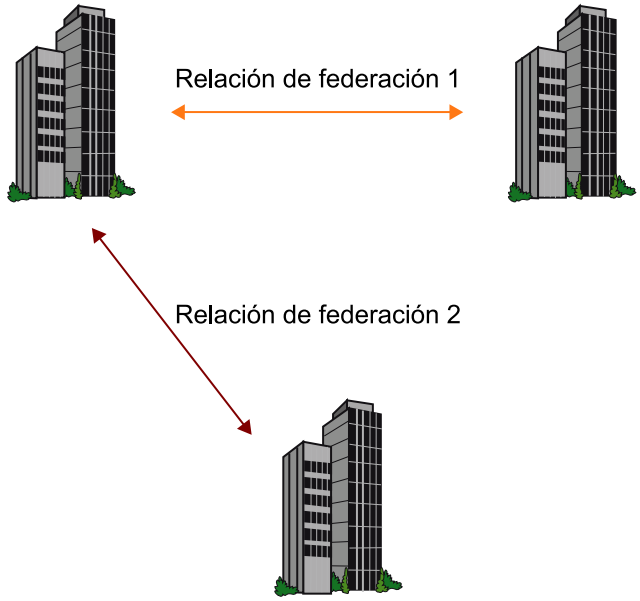
Un sistema de identidad y gestión de acceso implica diferentes aspectos aparte del tecnológico. Cuando pensamos en ello no únicamente hemos de tener en cuenta la forma de autenticarse como elemento más evidente y expuesto, sino también, por ejemplo, la forma de distribución de las credenciales, la política de uso, el registro que se va a llevar a cabo, las evidencias electrónicas, la privacidad, etc.

Como ejemplo de esto último, en la federación de identidades es muy importante cómo se establece el nivel de confianza inicial entre dos dominios de identidad como parte integral del proceso de despliegue del mismo. Dependiendo de ello se establecen diferentes patrones de federación, que impactan decididamente en los resultados del proceso de federación.

1.3.1. Federación *ad-hoc*

La federación *ad-hoc* es un tipo de patrón de federación en el que la relación se hace 1 a 1 entre pares, tal como se ve en la figura siguiente. Todos los acuerdos relativos se realizarán entre dos dominios de identidades diferentes, y potencialmente pertenecientes a dos organizaciones también diferentes. Debido a esto, si existiese otra relación *ad-hoc* dentro de una misma empresa, tendrían que volverse a negociar exactamente igual todos los términos del acuerdo de federación de identidades con la destinataria de esa otra relación de federación.

Diagrama conceptual de patrones de federación *ad-hoc*



Generalmente no se dispone de estándares y, si se tienen, han de ser adoptados de la misma forma por las dos partes de la relación. En este sentido, las decisiones de la federación tienen un alto impacto en cómo las identidades se gestionan en cada una de las partes, lo cual implica que realmente se está haciendo poco por solventar los problemas de las aproximaciones centralizadas expuestas anteriormente.

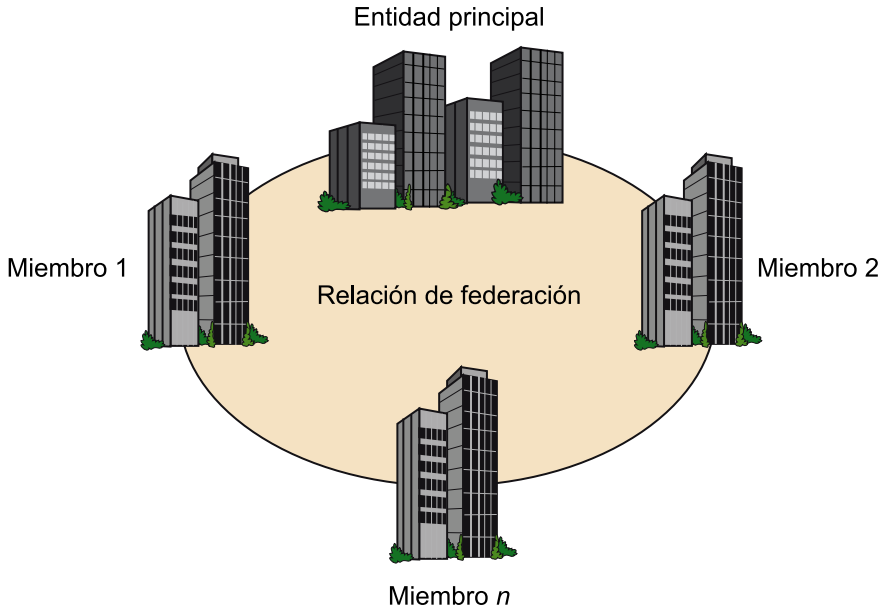
Este tipo de relaciones existen desde hace tiempo, generalmente como respuesta a las oportunidades de negocio temporales o bien a las relaciones de dependencia de algunas empresas respecto de otras mayores, como es el caso de las empresas suministradoras de grandes conglomerados empresariales.

En este caso, realizar una relación de federación no tiene que ser muy traumático pero la complejidad varía dependiendo del número de relaciones que haya que adoptar, pues la cantidad de trabajo que comporta es lineal. Así, la capacidad de una empresa de mantener diferentes relaciones de federación *ad-hoc* dependerá de su capacidad para gestionar múltiples relaciones independientes y sus costes técnicos y económicos asociados.

1.3.2. Federación *hub-and-spoke*

La federación *hub-and-spoke* es un tipo de federación que se basa en una organización dominante que dicta el conjunto de estándares técnicos, las políticas de confiabilidad, la gestión del riesgo, etc., y un conjunto de organizaciones, generalmente de menor tamaño, que siguen estos dictámenes, creando un archipiélago en cuanto a identidad se refiere.

Diagrama conceptual de federación *hub-and-spoke*



Los acuerdos son 1 a 1 y siempre una de ellas es la entidad principal del *hub*, lo que no facilita la escalabilidad del patrón.

Ejemplo

Retomando el primer ejemplo descrito al inicio del apartado, podríamos pensar que la compañía aérea en sí misma podría ser la entidad principal por la fuerza económica que tiene, y que diferentes empresas más pequeñas, como las de alquiler de coches o los servicios de hotel, pudiesen ser miembros de la federación. En la realidad, grandes empresas como Boeing, Amazon o General Motors tienen este tipo de patrón de federación.

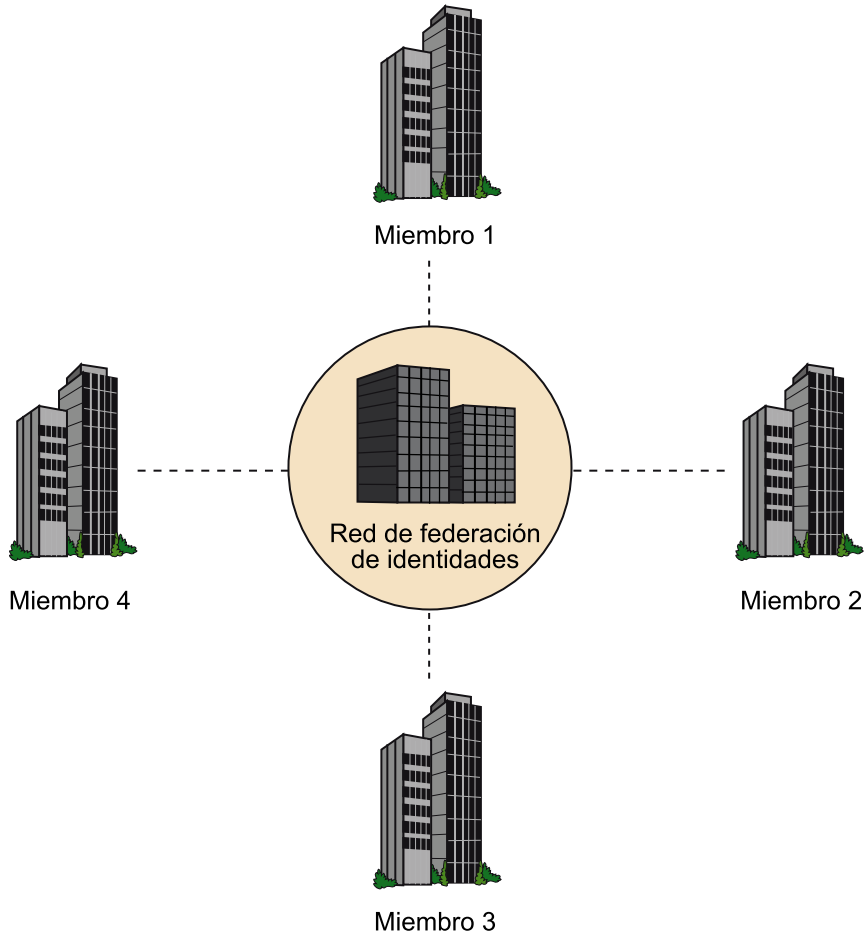
Si alguna de las organizaciones miembro de la federación tuviese que federarse con otra organización que también tuviese un patrón del mismo estilo, posiblemente tendría que volver a realizar los mismos pasos que siguió para unirse a la primera federación, pues la inversión realizada en la primera no le serviría de mucho, dado que habría de adoptar seguramente diferentes estándares técnicos, legales o de calidad de servicio.

Finalmente, recalcar que la confianza de la federación la aporta casi exclusivamente la empresa de mayor tamaño, esta es quien dicta las normas y políticas que regulan la federación. En un patrón de este tipo, es frecuente observar que si hay empresas de similar tamaño que están federando identidades, cada una de ellas tenderá a proteger sus intereses, dando lugar muchas veces a conflictos internos.

1.3.3. Red de federación de identidades

Una red de identidades es una organización independiente completa y únicamente dedicada a los aspectos técnicos y administrativos de la federación de identidades. Está exclusivamente para que la red funcione de forma adecuada y sirva a sus miembros, que son realmente los propietarios de la red en sí misma. Los miembros grandes y pequeños tienen todos los mismos deberes y obligaciones incluso si compiten entre ellos.

Diagrama conceptual de federación a través de una red de federaciones



Con esta organización, la red se puede centrar en dotar de valor a la propia red gestionando, por ejemplo, los aspectos relativos a la privacidad, la relación con los gobiernos y entes legales, o bien luchar contra el fraude y el robo de identidad. La red estipula también una serie de normas comunes y tiene articulados mecanismos de resolución de conflictos para la no obediencia de las normas establecidas, que puede llegar hasta la expulsión de la red.

VISA

Una de las redes de identificación más famosas es la red VISA, que se encarga de federar la identidad de tarjetas de crédito entre entidades bancarias. De hecho, VISA es la mayor red de federación de pagos, con decenas de miles de afiliados a lo largo de todo el mundo.

Esta red surgió a partir de las primeras emisiones de tarjetas de crédito por parte del Bank of America, llamada Bank Americard. Después de pasar por diferentes etapas, entre las que se encuentra la federación *ad-hoc* y la *hub-and-spoke*, se creó la red de identidades que hoy conocemos como VISA y que es la mayor red de federación de tarjetas de crédito del mundo, y ha dado respuesta a los diferentes problemas existentes en los dos patrones de federación antes mencionados.

1.4. *Single sign-on*

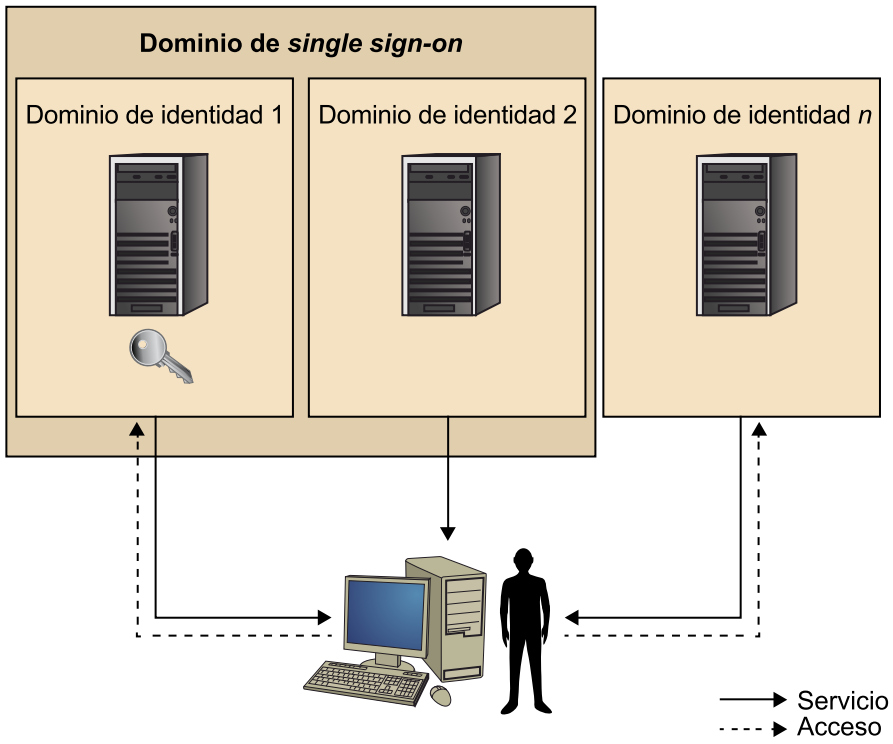
Dentro de todos los servicios para la identificación y la gestión de acceso que hemos comentado anteriormente, la autenticación ha sido el que más ha disfrutado de los beneficios de la federación llegando, incluso, a asociar federación con *single sign-on*.

El *single sign-on*, o autenticación única, se basa en que el usuario se autentica una única vez en uno de los sistemas federados y, a partir de ahí, puede acceder al resto de sistemas sin tener que volver a autenticarse de nuevo.

Operaciones de *single sign-of*

De la misma forma que hay operaciones de *single sign-on*, también hay operaciones de *single sign-of*, imprescindible de igual modo para evitar, por ejemplo, secuestros de sesión.

Diagrama que muestra el proceso de autenticación único para diferentes dominios de identidad



A favor de este mecanismo está la reducción del número de credenciales a recordar (si estas son del tipo usuario/contraseña), disminuyendo así lo que se ha llamado fatiga de autenticación. También permite acelerar los procesos de acceso, ya que no requiere intervención humana. Permite gestionar los accesos centralizadamente y reducir los costes de mantenimiento global.

En su contra tenemos que, una vez se han robado las credenciales de un usuario, inmediatamente se tiene acceso a multitud de sistemas sin ningún mecanismo de seguridad que lo impida. Para mitigar este hecho, y dado que únicamente se ha de recordar una única contraseña, en este tipo de sistemas las políticas de generación de contraseñas suelen ser más estrictas.

Los sistemas de identidad, necesarios para cualquier proceso de federación, suelen ser sistemas de una misma organización que incluso muchas veces delegan los procesos de identificación a terceros, con lo que se desvanecería entonces el concepto inicial de federación hacia un concepto de autenticación delegada.

Ved también

El concepto de autenticación delegada se tratará en el subapartado 3.7 de este módulo.

Finalmente, cabe comentar que se han generado múltiples aproximaciones para el servicio de *single sign-on*, aunque son los estándares más desarrollados los que se acaban imponiendo a la hora de realizar despliegues de una cierta entidad. Incluso en este grupo de estándares, cada uno de ellos ha nacido y crecido con un objetivo en mente, por lo que se adecuará mejor a aquellas misiones que coincidan con el enfoque para el cual se concibieron.

2. Estándares

Como hemos visto anteriormente, la federación de identidades se basa en el intercambio de información entre diferentes dominios de identidad. Para ello, es imprescindible disponer de estándares adecuados que permitan un lenguaje común entre sistemas de diferente índole.

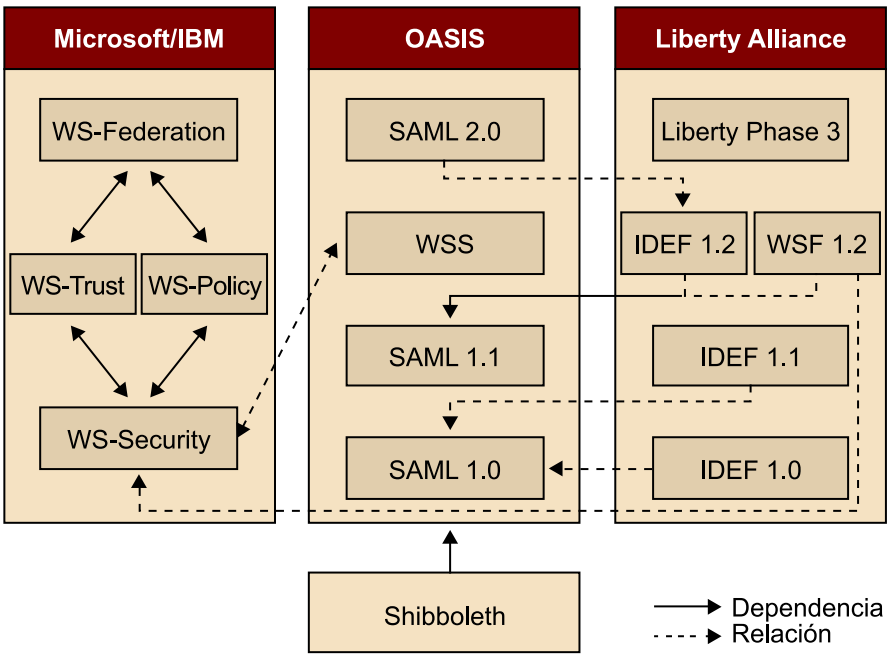
No es de extrañar que hayan surgido diferentes iniciativas para dar respuesta a dichas necesidades, generalmente a partir de asociaciones o uniones empresariales de gran impacto en el mundo de las tecnologías de la información y la comunicación.

Dentro de estos estándares nos centraremos básicamente en cuatro:

- Microsoft/IBM
- OASIS
- Liberty Alliance
- Shibboleth (Internet 2)

La figura siguiente muestra la relación existente entre los diferentes estándares y los protocolos asociados.

Diagrama de relación entre estándares



Parece difícil saber cuál va a convertirse en el estándar *de facto* en el mercado. La tendencia actual apunta a que SAML se ha adoptado, de una u otra forma, por Microsoft/IBM y por Liberty Alliance, pero no se vislumbra cuál va a ser la relación entre Liberty Alliance y Microsoft/IBM.

2.1. Microsoft/IBM

A principios de la década del 2000, Microsoft e IBM crearon un papel[4] conjunto para la creación de web services seguros. Esta especificación se llamó WS-* y está basada en la creación de una serie de *tokens* que se adjuntan a los mensajes, alguno de los cuales tiene que ver con la identidad.

(4)Del inglés, *paper*

Se estructura de forma modular, aislando componentes de la seguridad diferenciados en distintas especificaciones. A continuación describimos los más importantes, aunque existen diferentes especificaciones WS-* para resolver diferentes aspectos:

- **WS-Policy**. Es un lenguaje que describe la política de seguridad de un determinado *web service*. Por ejemplo, dado que WS-Security puede utilizar diferentes sistemas de *tokens* (como SAML o *Kerberos*), la política de seguridad podría enunciar el uso de SAML 1.1 y no *Kerberos*, por ejemplo.

- **WS-Trust**. Es un lenguaje que permite a los servicios de una autoridad en la que se confía, poder intercambiar un *token* con otra. Este aspecto es la base de la federación en sí misma.

- **WS-Federation**. Es un lenguaje que orquesta las interacciones de los diferentes objetos WS-Trust (identidades, atributos, autenticaciones, etc.) participantes en un servicio web.

2.2. OASIS

SAML fue creado por la Organization for the Advancement of Structured Information Standandards (OASIS), que es un consorcio internacional sin ánimo de lucro que produce estándares tecnológicos de negocio. Además de este lenguaje, el más ampliamente utilizado y conocido, ha producido otros también muy populares, como el Service Provisioning Markup Language (SPML) y el eXtensible Access Control Markup Language (XACML).

En general, los sistemas de automatización de la identidad necesitan de una manera de distribuir las aserciones de autenticación y autorización, como se hace por ejemplo a través de Kerberos. SAML en este aspecto ha ido ganando adeptos en los últimos años y se adivina como el estándar *de facto* en un futuro.

Ved también

SAML y Kerberos se estudian respectivamente en los subapartados 3.1 y 3.7 de este módulo.

SAML se basa esencialmente en un cliente que pide algo sobre una identidad a una autoridad SAML[5], que a su vez responde con lo que se ha llamado aserciones. Se pueden tener tres tipos de aserciones, y por tanto tres tipos de autoridades SAML:

[5]Veremos con más detalle los mecanismos involucrados en el apartado 3.

1) Aserciones de autenticación. En este caso el cliente pide sobre la autenticación del sujeto **S** de la forma **F** en el tiempo **T**.

Por ejemplo, una aserción de este tipo podría ser: "Alice en micompania.com se autenticó a través de contraseña el 20011-07-20T17:00:00-05:00".

2) Aserciones de atributo. Una vez el sujeto **S** se ha autenticado, el cliente puede preguntar una serie de atributos asociados al mismo, con sus correspondientes valores.

Por ejemplo, "Alice está asociada con el atributo Departamento con valor Ingeniería y con el atributo correo electrónico y valor alice@micompania.com".

3) Aserciones de autorización. Cuando el sujeto **S** quiere acceder a un determinado recurso, el cliente pregunta a un *policy decission point* (servidor SAML de aserciones de autorización) por la posibilidad de acceder realizando la acción **A**.

Por ejemplo, "El sujeto http://boo.com/foo tiene permiso para leer http://b.com/bar evidenciado por las aserciones A1, A2 y A8".

No necesariamente los tres tipos de aserciones tienen que ser emitidas por tres servicios diferentes. Un servicio puede emitir los tres tipos de aserciones, a la vez que un servicio de aserciones también puede ser cliente y preguntar a otro servicio SAML.

Generalmente, una aserción contiene:

- ID[6] del emisor y fechado de tiempo de expedición;
- ID único de la aserción.
- Sujeto, incluyendo el nombre del dominio de seguridad, y opcionalmente los datos de autenticación.
- Informacional adicional de la entidad emisora llamada advine.
- Condiciones bajo las cuales la aserción es válida (por ejemplo, fecha de caducidad).
- Restricciones de audiencia.
- Restricciones de destino, como el recurso específico (por ejemplo, URL).
- Condiciones específicas de la aplicación.

[6]La siglas ID corresponden al término identificador.

2.3. Liberty Alliance

Liberty Alliance es un consorcio de más de 170 empresas para la creación de estándares y especificaciones para la federación de identidades. En un inicio el consorcio quería crear una única especificación para la federación de identidades, no obstante, finalmente la transformaron en tres especificaciones diferentes:

1) Identity Federation Framework (ID-FF). Permite *single sign-on* y el vínculo de cuentas entre participantes que tienen una relación de confianza.

2) Identity Web Services Framework (ID-WSF). Permite que grupos de participantes confiables entre sí puedan unirse a otros grupos, permitiendo a los usuarios el control sobre cómo se comparte esa información.

3) Identity Services Interface Specifications (OD-SIS). Se utiliza para la construcción de un conjunto de servicios interoperables sobre ID-WSF.

Ved también

En el subapartado 3.1 se muestra con más detalle las particularidades de SAML, sin duda el lenguaje de federación más adoptado en estos momentos por la industria, aunque la mayoría de productos ofrecen una gran compatibilidad con muchos otros estándares de federación.

Enlace de interés

Para saber más sobre el proyecto Liberty Alliance, podéis consultar la web http://www.projectliberty.org.

2.4. Internet 2 y Shibboleth

Internet 2 es una coalición de universidades americanas que trabajan para crear la próxima generación de Internet, y Shibboleth es su estructura arquitectónica para la gestión de identidades. Shibboleth está por completo basado en SAML, proveyendo básicamente servicios de *single sign-on* y administración federada del acceso a recursos restringidos.

Al estar basada su negociación en atributos, es un protocolo que tiene por diseño mecanismos para garantizar la privacidad. Esto último contribuye a una tendencia reciente de tener en cuenta el diseño de los sistemas a la seguridad (*security by design*) y a la privacidad (*privacy by design*).

3. Tecnología para la gestión de identidades federadas

El objetivo de este apartado es el de repasar aquellos estándares que más se están adoptando en la industria para cubrir las diferentes necesidades de la gestión de identidades federadas; desde el control de acceso, a la autenticación, pasando por la distribución de políticas o la provisión de identidades.

3.1. SAML

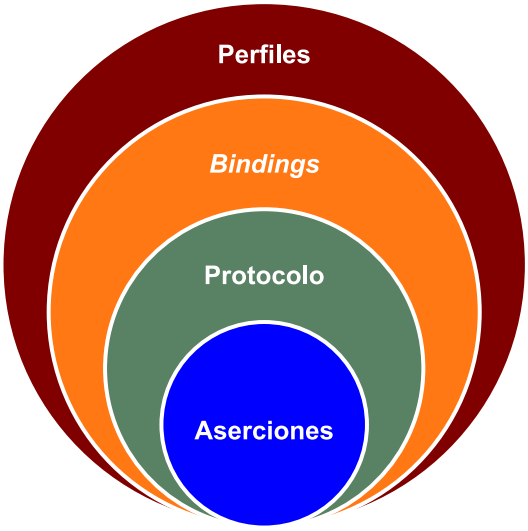
Security assertion markup language (SAML) es un estándar de OASIS basado en XML para el intercambio de información de autenticación y autorización entre dominios de seguridad.

La especificación SAML 2 está divida en los componentes que se muestran en la siguiente figura.

Versiones de SAML

Actualmente, existen dos versiones diferentes activas, SAML 1.1 y SAML 2, aunque nos centraremos aquí en su última versión, la 2.0.

Diagrama de componentes de SAML 2.0



3.1.1. Aserciones SAML

Una aserción SAML es una unidad de información emitida en un momento determinado por una autoridad SAML o emisor, sobre un sujeto acogido a determinadas condiciones, y queda definido por la siguiente estructura:

```
<saml:Assertion ...>
...
</saml:Assertion>
```

Dentro de una aserción, podemos añadir tres tipos de sentencias:

1) Autenticación. Usamos esta sentencia cuando queremos afirmar que un usuario se autenticó por medio de un mecanismo y en un momento determinado. Sólo se obtiene información sobre la acción de autenticación, no de la realización de dicha acción.

2) Atributo. Usamos esta sentencia cuando queremos establecer una relación entre un determinado usuario y una serie de atributos.

3) Autorización. Usamos esta sentencia para informar si aceptamos o denegamos el acceso de un usuario a un recurso protegido. Además, podemos recibir información extra en una petición de decisión de autorización que el emisor haya considerado determinante a la hora de evaluar nuestra respuesta.

Asimismo, una aserción puede contener más de un tipo de sentencia, por lo que es común encontrarse aserciones con sentencias de autenticación y de atributos, como podemos ver en el siguiente ejemplo.

Ejemplo

```
<saml2:AssertionID="hgd1jcigrccv9osuk564do272i" IssueInstant=
 "2009-10- 08T16:44:50Z" Version="2.0">
<saml2:Issuer>http://localhost/ExampleNifLocator</saml2:Issuer>
<saml2:Subject>
<saml2:NameID
   Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">
   andres
</saml2:NameID>
<saml2:SubjectConfirmation
   Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"/>
</saml2:Subject>
<saml2:Conditions NotBefore="2009-10-08T16:39:50Z"
   NotOnOrAfter="2009-10-08T16:54:50Z"/>
<saml2:AuthnStatement AuthnInstant="2009-10-08T16:44:50Z">
<saml2:AuthnContext>
<saml2:AuthnContextClassRef>
   urn:oasis:names:tc:SAML:2.0:ac:classes:Password
</saml2:AuthnContextClassRef>
</saml2:AuthnContext>
</saml2:AuthnStatement>
<saml2:AttributeStatement>
<saml2:Attribute Name="NifNie">
<saml2:AttributeValue>28806345S</saml2:AttributeValue>
</saml2:Attribute>
</saml2:AttributeStatement>
</saml2:Assertion>
```

La estructura de la aserción SAML mostrada es la siguiente:

- Emisor de la aserción, mediante el elemento `<saml2:Issuer>`, indicando quién la ha generado a través de una cadena de caracteres. Normalmente es una URI[7].
- Sujeto, a través del elemento `<saml2:Subject>`, donde se indica sobre qué principal o usuario se ha generado la aserción y el formato en el que se está especificando dicha información.
- Condiciones, con el elemento `<saml2:Conditions>`, indicando desde qué momento es válida la aserción y hasta cuándo.
- Sentencias de atributos y autenticación.

[7]Las siglas URI corresponden a las siglas en ingles de *uniform resource identifier*.

Ejemplo

En la siguiente clase Java, veremos cómo crear una aserción SAML con una estructura similar a la vista en el ejemplo anterior.

```
import com.sun.xml.wss.saml.*;
import java.io.StringWriter;
import java.util.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import org.w3c.dom.Document;

public class Example2 {

   public static void main(String[] args) throws Exception {
      SAMLAssertionFactory af = SAMLAssertionFactory.newInstance(
      SAMLAssertionFactory.SAML2_0);

      String assertionId = UUID.randomUUID().toString();

      NameID nameId = af.createNameID("cr01", null, null);

      GregorianCalendar issuerInst = new GregorianCalendar();

      Subject subject = af.createSubject(nameId,
      af.createSubjectConfirmation(null,
      "urn:oasis:names:tc:SAML:2.0:cm:bearer"));
```

```
        List<Object> statements = new LinkedList<Object>();
        List attributes = new LinkedList();
        String nameAttr = "sn1";
        List<String> valuesAttr = new LinkedList<String>();
        valuesAttr.add("Rodriguez");
        attributes.add( af.createAttribute(nameAttr, valuesAttr) );
        AttributeStatement attrSt = af.createAttributeStatement(attributes);
        statements.add(attrSt);
        AuthnContext ac = af.createAuthnContext(
        "urn:oasis:names:tc:SAML:2.0:ac:classes:Password", null);
        AuthnStatement as = af.createAuthnStatement(null, null, ac, null,
        null);
        statements.add(as);
        Assertion assertion = af.createAssertion
        (assertionId, nameId, issuerInst, null, null, subject, statements);
        showAssertion(assertion);
    }
}
```

En este ejemplo, estamos usando las clases que nos ofrece la librería Metrov1.4 para trabajar con aserciones SAML. Como vemos, tenemos prácticamente una clase por elemento que aparece dentro de una aserción.

La clase `SAMLAssertionFactory` nos permite crear, de manera segura, cada uno de los elementos que necesitamos. El principal aspecto a tener en cuenta usando esta factoría es que permite la generación de elementos para SAML 1 y SAML 2; como las aserciones y los elementos que hay dentro de ellas son diferentes entre dichas versiones, hay que usar sólo los métodos adecuados.

Para crear una aserción SAML 2, necesitaremos definir los siguientes elementos:

- Identificador para la aserción; en este caso, lo generamos dinámicamente:

```
String assertionId = UUID.randomUUID().toString();
```

- Identificador de nombre, que identifica al sujeto sobre el que se emite la aserción:

```
NameID nameId = af.createNameID("cr01", null, null);
```

- Sujeto, que contiene tanto el identificador de nombre como información de cómo probar la identidad del usuario por parte del receptor de la aserción:

```
Subject subject = af.createSubject(nameId,
af.createSubjectConfirmation(null,
      "urn:oasis:names:tc:SAML:2.0:cm:bearer"));
```

- Instante en el que se emite la aserción:

```
GregorianCalendar issuerInst = new GregorianCalendar();
```

- Sentencia de autenticación, que indica cómo se ha comprobado la identidad del usuario:

```
AuthnContext ac = af.createAuthnContext(
      "urn:oasis:names:tc:SAML:2.0:ac:classes:Password", null);
      AuthnStatement as = af.createAuthnStatement(null, null, ac,
null, null);
```

- Sentencia de atributos, con la lista de atributos que están asignados para dicho sujeto:

```
List attributes = new LinkedList();
String nameAttr = "sn1";
List<String> valuesAttr = new LinkedList<String>();
valuesAttr.add("Rodriguez");
attributes.add( af.createAttribute(nameAttr, valuesAttr) );
AttributeStatement attrSt = af.createAttributeStatement(attributes);
```

3.1.2. Protocolo SAML

SAML define un protocolo de intercambio de mensajes tipo petición/respuesta (*request*/*response*) con el objetivo de obtener aserciones de un usuario.

De esta forma, una petición SAML puede solicitar información sobre los siguientes elementos:

- Autenticación
- Decisión de autorización
- Atributos

Cada tipo de petición está orientado a obtener sentencias que corresponden a cada uno estos tipos. Estas sentencias son usadas para que, en la especificación SAML 2, se definan los siguientes protocolos:

- **Petición y consulta de atributos**. Con este protocolo se puede solicitar una aserción utilizando su referencia y también consultar los atributos de un sujeto.

- **Petición de autenticación**. Gracias a este protocolo se puede iniciar el proceso de autenticación del sujeto con el objetivo de obtener aserciones que lo identifiquen.

- **Resolución de artefacto**. La especificación SAML permite enviar una referencia, llamada artefacto, en lugar de un mensaje. De este modo, cuando un receptor obtiene dicho artefacto, podrá utilizar el servicio de resolución de artefactos de la entidad que lo haya emitido para obtener el mensaje al que representaba.

- **Gestión de identificadores de nombre** (o *name identifier*), que nos permite gestionar, en determinados aspectos, el identificador de nombre de un sujeto en un proveedor de identidad.

- ***Logout* simple**, que permite finalizar la sesión de un sujeto en un dominio de identidad de un proveedor de servicio.

- **Asignaciones de identificadores de nombre**, que permiten solicitar la conversión del formato que se utiliza a la hora de expresar un identificador de nombre.

De todos ellos, el más usado es el protocolo de petición y consulta de atributos.

Ejemplo

El siguiente ejemplo muestra la estructura típica de una petición de atributo.

```
<samlp:AttributeQueryxmlns:samlp='urn:oasis:names:tc:SAML:2.0:
   protocol'
   ID='6985f6b9-8946-4c9f-9dac-4180ea3f6c88'
   IssueInstant='2009-11-22T11:20:35Z' Version='2.0'>
<saml:Issuer xmlns:saml='urn:oasis:names:tc:SAML:2.0:assertion'>
   http://www.prise.es/uoc-book/example2
</saml:Issuer>
<saml:Subject xmlns:saml='urn:oasis:names:tc:SAML:2.0:assertion'>
<saml:NameID
   Format='urn:oasis:names:tc:SAML:2.0:nameid-format:unspecified'
```

```
    xmlns:saml='urn:oasis:names:tc:SAML:2.0:assertion'>
    cr01
</saml:NameID>
</saml:Subject>
</samlp:AttributeQuery>
```

Para generar una petición de atributo como la mostrada en este ejemplo, utilizaremos la siguiente clase Java.

```
import java.io.StringReader;
import java.text.SimpleDateFormat;
import java.util.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.InputSource;
public class Example2 {
   private static final String reqMsg =
      "<samlp:AttributeQuery " +
      " xmlns:samlp='urn:oasis:names:tc:SAML:2.0:protocol' " +
      " ID='%%ID%%' IssueInstant='%%ISSUEINST%%' Version='2.0'>" +
      " <saml:Issuer xmlns:saml='urn:oasis:names:tc:SAML:2.0:assertion'>"
      +
      "%%ISSUER%%" +
      " </saml:Issuer>" +
      " <saml:Subject xmlns:saml='urn:oasis:names:tc:SAML:2.0:assertion'>"
       +
      " <saml:NameID Format='urn:oasis:names:tc:SAML:2.0:nameid-format:
       unspecified'" +
      " xmlns:saml='urn:oasis:names:tc:SAML:2.0:assertion'> " +
      "%%SUBJECT%%" +
      " </saml:NameID>" +
      " </saml:Subject>" +
      "</samlp:AttributeQuery>";
   public static String getAttributeRequest(String id, String issuer,
   String subject) {
      Calendar cal = Calendar.getInstance(TimeZone.getTimeZone("US/
      Arizona"));
      SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
      SimpleDateFormat sdfTime = new SimpleDateFormat("HH:mm:ss");
   String res = Example2.reqMsg
   res = res.replaceAll("%%ID%%", id);
   res = res.replaceAll("%%ISSUEINST%%", sdf.format(cal.getTime()) +
   "T" +
   sdfTime.format(cal.getTime()) + "Z");
   res = res.replaceAll("%%ISSUER%%", issuer);
   res = res.replaceAll("%%SUBJECT%%", subject);
   return res;
   }

   public static Element getElementFromString(String s) throws
      Exception {
      DocumentBuilderFactory factory = DocumentBuilderFactory.
      newInstance();
      factory.setNamespaceAware(true);
      DocumentBuilder builder = factory.newDocumentBuilder();
      InputSource is = new InputSource(new StringReader(s));
      Document d = builder.parse(is);
      return d.getDocumentElement();
   }
   public static void main(String[] args) throws Exception {
      String id = UUID.randomUUID().toString();
      String issuer = "http://www.prise.es/uoc-book/example2";
      String subject = "cr01";
      String reqMsg = getAttributeRequest(id, issuer, subject);
      System.out.println(reqMsg);
      Element req = getElementFromString(reqMsg);
   }
}
```

Como vemos en el método `getAttributeRequest`, una de las formas más cómodas de trabajar con SAML es representar, directamente en un objeto tipo `String`, el mensaje, puesto que reduce complejidades a la hora de importar librerías a nuestro proyecto. A pesar de ello, obviamente habrá que tener cuidado con la estructura del mensaje, ya que podemos confundirnos y no utilizar correctamente los elementos permitidos y sus valores en el formato adecuado en el mensaje que queremos transmitir.

Una vez que tenemos el mensaje completo, utilizaremos la función `getElementFromString` para obtener un objeto tipo `org.w3c.dom.Element` que nos resultará más fácil de integrar en peticiones SOAP[8] en un futuro.

[8]SOAP es la sigla de *simple object access protocol.*

En las peticiones de atributos podemos consultar todos los atributos del usuario, para lo cual no indicamos ninguna lista de atributos en la petición, o bien podemos solicitar unos atributos determinados, indicando en la petición cuáles queremos consultar.

Una petición de atributos será respondida por una **respuesta SAML** `<samlp:Response>`, que tiene la siguiente estructura:

- Emisor de la respuesta SAML, por medio del elemento `<saml:Issuer>`.
- Tipo de respuesta, informando si ha ocurrido algún tipo de error a la hora de procesar la petición o generar la respuesta, reflejado en el elemento `<samlp:Status>`.
- Una aserción SAML o más de una, siendo su aparición opcional.

Para el caso de la petición de atributos, se incluirá una aserción como la mostrada en el primer ejemplo, donde, dentro de una sentencia de atributos, serán incluidos aquellos atributos que sí tiene el usuario y que la entidad SAML, generalmente el servicio de autoridad de atributos (AA) del proveedor de identidad, ha autorizado a emitir a dicho emisor de la petición, en función de su política de emisión de atributos.

Autoridad y aserciones de atributos

Una autoridad de atributos es una entidad del sistema que produce las aserciones de atributos. Una aserción de atributos es una aserción que transmite información sobre los atributos de un sujeto. Un atributo es una nota característica de un objeto.

Ejemplo

```
<samlp:Response xmlns:samlp="..." xmlns:saml="..." xmlns:ds="..."
ID="_6c3a4f8b9c2d" Version="2.0" IssueInstant="2004-03-27T08:42:
00Z">
<saml:Issuer>https://www.example.com/SAML</saml:Issuer>
<ds:Signature> ... </ds:Signature>
<Status>
<StatusCode Value="..."/>
</Status>
<saml:Assertion>
...
<saml:Assertion>
</samlp:Response>
```

El atributo `Value` del elemento `<StatusCode>` informa del tipo de respuesta que se está devolviendo y, aunque la especificación de SAML 2 define una numerosa lista de valores con su semántica, podemos destacar los siguientes:

- `urn:oasis:names:tc:SAML:2.0:status:Success`, en el caso de que la petición haya sido procesada correctamente.
- `urn:oasis:names:tc:SAML:2.0:status:AuthnFaile`, si no se ha podido verificar la identidad del sujeto.
- `urn:oasis:names:tc:SAML:2.0:status:InvalidAttrNameOrValue`, si ha ocurrido un error al procesar el nombre o el valor de alguno de los atributos.

3.1.3. SAML bindings

En SAML, se le llama *binding* al procedimiento de asignar un protocolo SAML, como el de petición/respuesta que hemos visto más arriba, a una capa de transporte que permita su transmisión.

Los *bindings* disponibles en SAML 2 son:

- **SOAP 1.1**. Permite enviar y recibir mensajes SAML utilizando SOAP 1.1.
- **SOAP invertido o PAOS**. Este *binding* permite responder a un cliente HTTP que intenta acceder a un recurso con un mensaje SOAP que, a su vez, incluye una petición SAML. De esta forma, dicho cliente HTTP procesa la petición y responde con una respuesta SAML a través de SOAP. Si esta es correcta, el receptor permite el acceso a dicho cliente HTTP al recurso protegido.
- **Redirecciones HTTP**. Este *binding* permite enviar mensajes del protocolo SAML mediante los parámetros de una URL.
- **Mensajes HTTP con método POST**. A través de este se pueden transmitir mensajes SAML codificados en base 64 a través de un formulario HTML.
- **Artefactos en mensajes HTTP**. En este *binding* se transmite tanto la petición SAML como la respuesta a través de un artefacto. Así, cuando recibi-

mos un artefacto, tendremos que utilizar otro de los *bindings* para resolver dicho artefacto.

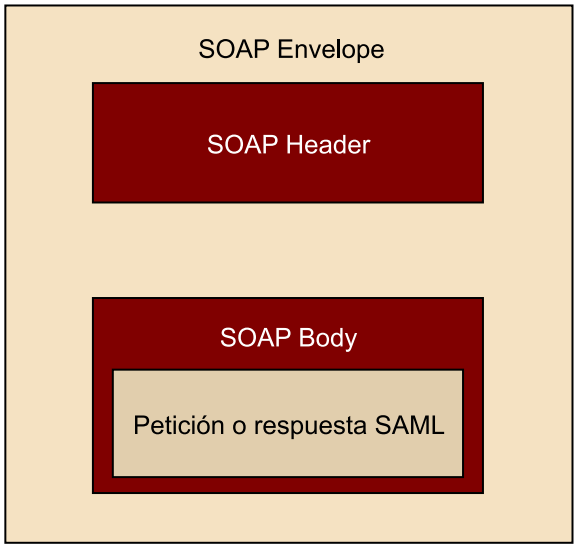
- **URI**. Referencia una aserción SAML con un identificador independiente del medio de transporte, que puede ser incluido en elementos como `<saml:AssertionIDRef>`.

Nosotros nos centraremos en los *bindings* SOAP 1.1 y mensajes HTTP con métodos POST, ya que son los más comunes dentro de las infraestructuras de autenticación y autorización.

3.1.4. *Binding* SOAP 1.1

Como hemos dicho anteriormente, el *binding* SOAP 1.1 permite enviar y recibir peticiones/respuestas SAML por medio del protocolo SOAP 1.1.

Diagrama de componentes de un mensaje SAML



Ejemplo

En el siguiente ejemplo podemos ver una petición HTTP de una solicitud de atributo a través de SOAP 1.1.

```
POST /SamlService HTTP/1.1
Host: www.prise.es
Content-Type: text/xml
Content-Length: nnn
SOAPAction: http://www.oasis-open.org/committees/security
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
<samlp:AttributeQuery
   xmlns:samlp='urn:oasis:names:tc:SAML:2.0:protocol'
   ID='6985f6b9-8946-4c9f-9dac-4180ea3f6c88'
   IssueInstant='2009-11-22T11:20:35Z' Version='2.0'>
   ...
   </samlp:AttributeQuery>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Como vemos, tan sólo tenemos que encapsular nuestra anterior petición de atributos dentro del cuerpo del mensaje SOAP. En el siguiente ejemplo se puede observar la creación de la petición de atributos bajo SOAP 1.1 en Java.

```
import java.io.StringReader;
import java.text.SimpleDateFormat;
import java.util.*;
import javax.xml.parsers.*;
import javax.xml.soap.*;
import org.w3c.dom.*;
import org.xml.sax.InputSource;

public class Example3 {
private static final String reqMsg =
      "<samlp:AttributeQuery " +
      " xmlns:samlp='urn:oasis:names:tc:SAML:2.0:protocol' " +
      " ID='%%ID%%' IssueInstant='%%ISSUEINST%%' Version='2.0'>" +
      " <saml:Issuer xmlns:saml='urn:oasis:names:tc:SAML:2.0:assertion'>"
       +
      "%%ISSUER%%" +
      " </saml:Issuer>" +
      " <saml:Subject xmlns:saml='urn:oasis:names:tc:SAML:2.0:assertion'>"
       +
      " <saml:NameID Format='urn:oasis:names:tc:SAML:2.0:nameid-format:
       unspecified'" +
      " xmlns:saml='urn:oasis:names:tc:SAML:2.0:assertion'> " +
      "%%SUBJECT%%" +
      " </saml:NameID>" +
      " </saml:Subject>" +
      "</samlp:AttributeQuery>";

   public static String getAttributeRequest(String id, String issuer,
      String subject) {
      Calendar cal =
      Calendar.getInstance(TimeZone.getTimeZone("US/Arizona"));
      SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
      SimpleDateFormat sdfTime = new SimpleDateFormat("HH:mm:ss");
      String res = Example3.reqMsg;

      res = res.replaceAll("%%ID%%", id);
      res = res.replaceAll("%%ISSUEINST%%", sdf.format(cal.getTime()) +
      "T" +
      sdfTime.format(cal.getTime()) + "Z");
      res = res.replaceAll("%%ISSUER%%", issuer);
      res = res.replaceAll("%%SUBJECT%%", subject);
      return res;
   }

   public static Element getElementFromString(String s) throws
      Exception {
      DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
      factory.setNamespaceAware(true);
      DocumentBuilder builder = factory.newDocumentBuilder();

      InputSource is = new InputSource(new StringReader(s));
      Document d = builder.parse(is);

      return d.getDocumentElement();
   }

   public static void main(String[] args) throws Exception {
      String id = UUID.randomUUID().toString();
      String issuer = "http://www.prise.es/uoc-book/example2";
      String subject = "cr01";
      String reqMsg = getAttributeRequest(id, issuer, subject);
      Element req = getElementFromString(reqMsg);

      MessageFactory messageFactory = MessageFactory.newInstance();
      SOAPMessage message = messageFactory.createMessage();
      SOAPPart soapPart = message.getSOAPPart();
      SOAPEnvelope envelope = soapPart.getEnvelope();
      SOAPBody body = envelope.getBody();
```

```
        body.addDocument(req.getOwnerDocument());
        message.saveChanges();

        SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
        SOAPConnection sc = scf.createConnection();
        SOAPMessage reply = sc.call(message,
        "http://www.ejemplo.com/SamlService");
    }
}
```

Como podemos ver, dentro del método estático `main`, una vez que tenemos la petición de atributos en un objeto `org.w3c.dom.Element`, generamos el mensaje SOAP y añadimos dentro de su cuerpo dicha petición.

```
MessageFactory messageFactory = MessageFactory.newInstance();
   SOAPMessage message = messageFactory.createMessage();
   SOAPPart soapPart = message.getSOAPPart();
   SOAPEnvelope envelope = soapPart.getEnvelope();
   SOAPBody body = envelope.getBody();
   body.addDocument(req.getOwnerDocument());
message.saveChanges();
```

De esta forma, ya tenemos una petición SOAP con una cabecera vacía –ya que no es necesario añadir ningún elemento para este ejemplo– y un cuerpo con la petición de atributos. Tan sólo nos queda realizar el envío de esta petición.

```
SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection sc = scf.createConnection();
SOAPMessage reply = sc.call(message,
   "http://www.ejemplo.com/SamlService");
```

El mensaje queda enviado al servicio desplegado en http://www.ejemplo.com/SamlService y obtenemos la respuesta en la variable `reply`.

En el caso de que queramos mostrar, por consola, cuál ha sido la respuesta, tendríamos que añadir las siguientes líneas.

```
TransformerFactory transformerFactory = TransformerFactory.
 newInstance();
Transformer transformer =
   transformerFactory.newTransformer();
Source sourceContent = reply.getSOAPPart().getContent();
StreamResult result = new StreamResult(System.out);
transformer.transform(sourceContent, result);
```

Una vez enviada la petición a dicho servicio, éste procesará la información y devolverá una respuesta SAML dentro de un mensaje SOAP.

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<samlp:Response ...>
   ...
</samlp:Response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Como observamos, el elemento `<samlp:Response>`, que representa la respuesta de la petición SAML recibida, se incluirá dentro del cuerpo de un mensaje SOAP.

3.1.5. *Binding* mensaje HTTP con método POST

El *binding* mensaje HTTP con método POST se utiliza en aquellos casos en los que el emisor y el receptor del mensaje SAML necesitan utilizar el navegador del usuario como intermediario.

Los mensajes SAML se codifican dentro de un formulario HTML y se envían a través del método POST.

En el caso de que el mensaje SAML sea una petición, el elemento que contiene el mensaje SAML dentro del formulario se llamará `SAMLRequest`, mientras que si es una respuesta se llamará `SAMLResponse`.

También suele incluirse en el formulario HTML, como un campo oculto, el parámetro opcional `RelayState`. Este se utiliza para facilitar la gestión del estado de la información y en este *binding* no se deberá superar los 80 caracteres. En el caso de que se envíe un `RelayState`, el elemento que genere el mensaje SAML de respuesta deberá incluir el mismo valor para la respuesta, mientras que si no se envía ninguno en la petición, al generar la respuesta se puede establecer un valor al `RelayState` siempre y cuando haya un acuerdo previo en el uso del *binding*.

Ejemplo

A continuación, presentamos un ejemplo de envío de petición SAML bajo *binding* HTTP Post.

```
<form method='post' action='SamlService'>
<input type='hidden' name='RelayState'
   value='http://www.prise.es/uoc-book/SamlService' />
<input type='hidden' name='SAMLRequest' value='PEVudGVyIHRleH...' />
</form>
```

Así, en dicho ejemplo, incluimos una petición SAML, que queda enviada mediante este *binding*, ya que tanto dicha petición como el `RelayState` quedan asociados al formulario que se ha generado y que el navegador ha enviado. Además, en la mayoría de los casos, el formulario se envía automáticamente gracias a que, con JavaScript, se indica que se envíe al cargar dicha página.

Para procesar la petición, podemos desarrollar un sencillo *servlet*, que compruebe si han llegado los dos parámetros y envíe la aserción como respuesta a la URL enviada en el campo `RelayState`.

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Map;
import javax.servlet.ServletException;
import javax.servlet.http.*;
import sun.misc.BASE64Decoder;

public class Example4 extends HttpServlet {

   private String processRequest(String samlRequest) { ... }

   protected void doPost(HttpServletRequest request,
   HttpServletResponse response)
   throws ServletException, IOException {
      PrintWriter out = response.getWriter();

      Map params = request.getParameterMap();
      if (params.containsKey("SAMLRequest") &&
      params.containsKey("RelayState")) {
         String relayState = request.getParameter("RelayState");
         String samlReq = request.getParameter("SAMLRequest");

         BASE64Decoder decoder = new BASE64Decoder();
         byte[] decodedBytes = decoder.decodeBuffer(samlReq);

         String samlRequest = new String(decodedBytes);
         String samlResponse = processRequest(samlRequest);
         out.println("<html>");
         out.println("<head><title>Servlet</title></head>");
         out.println("<body>");
         out.println("<form method='post' action='"+relayState+"'>");
```

```
            out.println("<input type='hidden' name='SAMLResponse' value='"+
             samlResponse+"' />");
            out.println("<input type='hidden' name='RelayState' value='"+
             relayState+"' />");
            out.println("<input type='submit' name='submit' value='Aceptar'/>");
            out.println("</form>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}
```

Obviamos el código del método `processRequest`, ya que su lógica se ha explicado en los ejemplos anteriores, y nos centraremos en cómo debemos comprobar los parámetros y generar la respuesta al navegador o agente del usuario.

```
Map params = request.getParameterMap();
if (params.containsKey("SAMLRequest") &&
params.containsKey("RelayState")) {
   String relayState = request.getParameter("RelayState");
   String samlReq = request.getParameter("SAMLRequest");
   BASE64Decoder decoder = new BASE64Decoder();
   byte[] decodedBytes = decoder.decodeBuffer(samlReq);
```

El primer paso es comprobar que están todos los parámetros que se esperaban en el mensaje recibido por método POST, y decodificar la petición SAML en base 64.

```
String samlRequest = new String(decodedBytes);
String samlResponse = processRequest(samlRequest);
```

La lógica principal de este *servlet* es procesar dicha petición SAML y generar una respuesta.

```
out.println("<html>");
out.println("<head><title>Servlet</title></head>");
out.println("<body>");
out.println("<form method='post' action='"+relayState+"'>");
out.println("<input type='hidden' name='SAMLResponse' value='"+
 samlResponse+"' />");
out.println("<input type='hidden' name='RelayState' value='"+
 relayState+"' />");
out.println("<input type='submit' name='submit' value='Aceptar'/>");
out.println("</form>");
out.println("</body>");
out.println("</html>");
```

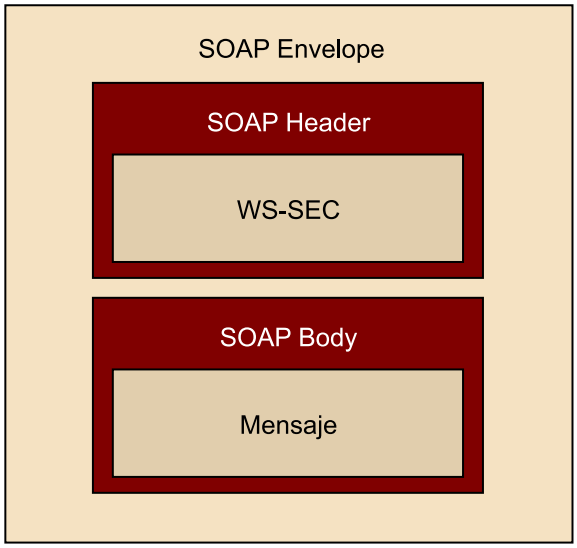
Por último, generamos otro formulario que se encarga de enviar la respuesta SAML a la URL indicada por el parámetro `RelayState` mediante POST.

3.2. Seguridad servicios web

El estándar Web Services Security (WS-SEC o WSS), publicado por OASIS, permite dotar a los mensajes SOAP transmitidos hacia servicios web una capa de autenticación y autorización.

Por medio del elemento `<wsse:Security>`, que queda asociado a la cabecera SOAP del mensaje, incluiremos los *tokens* de seguridad que contienen información que el receptor final, o uno intermediario, del mensaje utiliza para autenticar o verificar el mensaje recibido.

Diagrama de un mensaje WS-SEC



En una cabecera SOAP, puede aparecer el elemento `<wsse:Security>` más de una sola vez, aunque deberá especificar a quién va dirigido cada uno de ellos a través de su atributo actor. Otro atributo común en dicho elemento es `mustUnderstand`, que permite indicar si el receptor del mensaje debe o no procesar este elemento de seguridad, mediante los valores `true` y `false` respectivamente.

```
<S11:Envelope>
<S11:Header>
   ...
<wsse:Security S11:actor="..." S11:mustUnderstand="...">
   ...
</wsse:Security>
   ...
</S11:Header>
   ...
</S11:Envelope>
```

3.2.1. *Tokens* de seguridad

La especificación WS-SEC define una amplia lista de tipos de *tokens* de seguridad, pudiendo clasificarlos en tres grandes grupos:

1) *Token* de nombre de usuario o *username token*, que nos permite indicar el nombre de usuario del solicitante y, opcionalmente, una contraseña o una clave compartida para identificarlo.

2) *Token* de seguridad en binario o *binary security token*, que facilita la transmisión de *tokens* basados en tickets Kerberos o certificados digitales X.509.

3) *Token* XML, para aquellos que estén basados en XML.

Nos centraremos tanto en el *token* de nombre de usuario como en el de XML.

3.2.2. *Token* de nombre de usuario

El *token* de nombre de usuario permite identificar a un usuario o emisor del mensaje SOAP por medio de un nombre de usuario y una clave compartida o contraseña. Aunque este último sea opcional, se recomienda que se incluya en el caso de que se quiera dar fiabilidad a la autenticación del usuario o emisor.

La estructura general del *token* de nombre de usuario es la siguiente:

```
<wsse:UsernameToken wsu:Id="token-user">
<wsse:Username> ... </wsse:Username>
<wsse:Password Type="..."> ... </wsse:Password>
<wsse:Nonce EncodingType="..."> ... </wsse:Nonce>
<wsu:Created> ... </wsu:Created>
</wsse:UsernameToken>
```

Como podemos ver, el elemento `<wsse:UsernameToken>` representa al *token* que contendrá la siguiente información:

- Nombre de usuario, mediante el elemento `<wsse:Username>`.
- Contraseña del usuario, a través del elemento `<wsse:Password>`. Este, además, especificará en su atributo `Type` qué tipo de contraseña se está incluyendo:
  - En el caso de que sea una contraseña sin ningún tipo de cifrado, se utiliza el valor `#PasswordText`.
  - Por otro lado, si se está enviando la huella o *digest* de la contraseña, se indica por medio del valor `#PasswordDigest`.
- Valor criptográfico aleatorio, en el elemento `<wsse:Nonce>`, que nos permite detectar ataques de repetición. Se suele utilizar la codificación base 64 para incluir este valor. En caso de que queramos utilizar otra codificación, indicaremos cuál se ha utilizado en su atributo `EncodingType`.
- Cuándo fue creado el *token* a través del elemento `<wsu:Created>`.

De todos estos elementos, sólo el nombre de usuario es obligatorio a la horade enviar un *token* de nombre de usuario.

Ejemplo

En el siguiente ejemplo desarrollado en Java, vemos cómo incluir este tipo de *token* de seguridad en una cabecera SOAP.

```
import java.io.StringReader;
import java.math.BigInteger;
import java.security.*;
import java.text.SimpleDateFormat;
import java.util.*;
import javax.xml.parsers.*;
import javax.xml.soap.*;
import org.w3c.dom.*;
import org.xml.sax.InputSource;
import sun.misc.BASE64Encoder;

public class Example5 {

public static final String USERNAME_TOKEN = "<wsse:Security xmlns:
 wsse=
'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd' xmlns:wsu=
'http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd'>" +
" <wsse:UsernameToken wsu:Id='%%USERTOKENID%%'>" +
" <wsse:Username>%%USERNAME%%</wsse:Username>" +
" <wsse:Password>%%PASSWORD%%</wsse:Password>" +
" <wsse:Nonce>%%NONCE%%</wsse:Nonce>" +
" <wsu:Created>%%CREATED%%</wsu:Created>" +
" </wsse:UsernameToken>" +
"</wsse:Security>";

public static String getNonce() {
SecureRandom rand = null;
try {
rand = SecureRandom.getInstance("SHA1PRNG");
} catch (NoSuchAlgorithmException e) {
throw new IllegalArgumentException(e.toString());
}
rand.setSeed(System.currentTimeMillis());
byte[] nonce = new byte[16];
rand.nextBytes(nonce);
BASE64Encoder b64 = new BASE64Encoder();
return b64.encode(nonce);
}
public static String getUsernameToken(String username, String
password) {
Calendar cal = Calendar.getInstance(TimeZone.getDefault());
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
SimpleDateFormat sdfTime = new SimpleDateFormat("HH:mm:ss");

String res = USERNAME_TOKEN.replace("%%USERNAME%%", username);
res = res.replaceAll("%%PASSWORD%%", password);
res = res.replaceAll("%%NONCE%%", getNonce());
res = res.replaceAll("%%USERTOKENID%%",
new BigInteger(130, new SecureRandom()).toString(32));
res = res.replaceAll("%%CREATED%%", sdf.format(cal.getTime()) + "T" +
sdfTime.format(cal.getTime()) + "Z");

return res;
}

public static Element getElementFromString(String s) throws
Exception {
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
DocumentBuilder builder = factory.newDocumentBuilder();
InputSource is = new InputSource(new StringReader(s));
Document d = builder.parse(is);
return d.getDocumentElement();
}
public static void main(String[] args) throws Exception {
```

```
String reqMsg = "<requestMsg/>";
Element req = getElementFromString(reqMsg);
Element headerElem = getElementFromString(
getUsernameToken("alicia", "m1p4ssw0rd.!"));
MessageFactory messageFactory = MessageFactory.newInstance();
SOAPMessage message = messageFactory.createMessage();
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPHeader header = envelope.getHeader();
org.w3c.dom.Node headerNode = header.getOwnerDocument().importNode
(headerElem, true);
header.appendChild(headerNode);
SOAPBody body = envelope.getBody();
body.addDocument(req.getOwnerDocument());
message.saveChanges();
SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection sc = scf.createConnection();© FUOC · PID_00158713 50
Infraestructuras de autenticación y autorización delegada
SOAPMessage reply = sc.call(message, "http://www.prise.es/Service");
}
}
```

Este ejemplo trata de enviar el mensaje `<requestMsg/>` con un *token* de usuario incluido en la cabecera SOAP.

El primer paso es crear el *token* de usuario:

```
Element headerElem = getElementFromString(
   getUsernameToken("alicia", "m1p4ssw0rd.!"));
```

Como vemos en el código, realiza una llamada al método `getUsernameToken` indicando que el nombre de usuario es `alicia` y su contraseña es `m1p4ssw0rd.!`. Gracias a la definición de `Example5.USERNAME_TOKEN`, generamos el *token* de usuario reemplazando esos valores, pero también es necesario un valor para el *nonce*, otro para indicar cuándo se está creando el *token* y, finalmente, definir el identificador del elemento `<wsse:UsernameToken>`.

```
public static String getUsernameToken(String username,
String password) {
   ...
   Calendar cal = Calendar.getInstance(TimeZone.getDefault());
   SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
   SimpleDateFormat sdfTime = new SimpleDateFormat("HH:mm:ss");
   ...
   res = res.replaceAll("%%NONCE%%", getNonce());
   res = res.replaceAll("%%USERTOKENID%%",
   new BigInteger(130, new SecureRandom()).toString(32));
   res = res.replaceAll("%%CREATED%%", sdf.format(cal.getTime()) + "T" +
   sdfTime.format(cal.getTime()) + "Z");
   ...
}
```

Una vez que tenemos el *token* de usuario representado en un objeto `org.w3c.dom.Element`, vamos a incluirlo dentro de la cabecera SOAP.

```
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPHeader header = envelope.getHeader();
Node headerNode = header.getOwnerDocument().importNode(
   headerElem, true);
header.appendChild(headerNode);
```

De esta forma, ya tenemos un mensaje SOAP listo para ser transmitido a un servicio web.

```
SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection sc = scf.createConnection();
SOAPMessage reply = sc.call(message, "http://www.prise.es/Service");
```

3.2.3. *Token* XML

Uno de los *token* XML más popular es la aserción SAML, por lo que describiremos este tipo de *tokens* a través de los representados en dicha tecnología.

La aserción SAML contiene una serie de afirmaciones sobre un sujeto, y esto puede ser validado de tres formas diferentes:

1) ***Holder of key***. Confirmación de que el sujeto era el propietario de la clave utilizada en la generación de la firma digital de la aserción.

2) ***Sender vouchers***. El emisor de la aserción SAML responde por la verificación de la identidad del sujeto.

3) ***Bearer***. No se aporta ninguna información sobre la verificación de la identidad del usuario, por lo que depende de la confianza establecida en la entidad que ha generado la aserción.

Ejemplo

En este subapartado vamos a analizar un ejemplo de transmisión de mensaje SOAP incluyendo un *token* SAML con verificación de usuario *bearer*:

```
import com.sun.xml.wss.saml.*;
import java.io.StringReader;
import java.util.*;
import javax.xml.parsers.*;
import javax.xml.soap.*;
import org.w3c.dom.*;
import org.xml.sax.InputSource;

public class Example6 {

public static Assertion getSamlToken() throws Exception {
SAMLAssertionFactory af = SAMLAssertionFactory.newInstance(
SAMLAssertionFactory.SAML2_0);
String assertionId =
"a144e8f3&#8722;adad&#8722;594a&#8722;9649&#8722;924517abe933";
NameID nameId = af.createNameID("cr01", null, null);
GregorianCalendar issuerInst = new GregorianCalendar();
Subject subject = af.createSubject(nameId,
af.createSubjectConfirmation(null,
"urn:oasis:names:tc:SAML:2.0:cm:bearer"));
List<Object> statements = new LinkedList<Object>();
List attributes = new LinkedList();
String nameAttr = "sn1";
List<String> valuesAttr = new LinkedList<String>();

valuesAttr.add("Rodriguez");
attributes.add( af.createAttribute(nameAttr, valuesAttr) );
AttributeStatement attrSt = af.createAttributeStatement(attributes);
statements.add(attrSt);

AuthnContext ac = af.createAuthnContext(
"urn:oasis:names:tc:SAML:2.0:ac:classes:Password", null);
AuthnStatement as = af.createAuthnStatement(null, null, ac,
null, null);
statements.add(as);

Assertion assertion = af.createAssertion(assertionId, nameId,
issuerInst, null, null, subject, statements);
return assertion;
}

public static Element getElementFromString(String s) throws
Exception {
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
DocumentBuilder builder = factory.newDocumentBuilder();
```

```
InputSource is = new InputSource(new StringReader(s));
Document d = builder.parse(is);

return d.getDocumentElement();
}

public static void main(String[] args) throws Exception {
String reqMsg = "<requestMsg/>";
Element req = getElementFromString(reqMsg);
Assertion assertion = getSamlToken();
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();
Element assertionElem = assertion.toElement(doc);
Element headerElem =
doc.createElementNS("http://docs.oasis-open.org/wss/2004/01/oasis
-200401-wss-wssecurity-secext-1.0.xsd", "Security");
headerElem.setPrefix("wsse");
headerElem.appendChild(assertionElem);

MessageFactory messageFactory = MessageFactory.newInstance();
SOAPMessage message = messageFactory.createMessage();
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPHeader header = envelope.getHeader();
org.w3c.dom.Node headerNode = header.getOwnerDocument().importNode
(headerElem, true);
header.appendChild(headerNode);
SOAPBody body = envelope.getBody();
body.addDocument(req.getOwnerDocument());
message.saveChanges();

SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection sc = scf.createConnection();
SOAPMessage reply = sc.call(message, "http://www.prise.es/Service");
}
}
```

Como vemos en el ejemplo, el primer paso es crear la aserción SAML, que obtenemos al llamar en nuestro método principal a `getSamlToken()`, que devuelve un objeto `com.sun.xml.wss.saml.Assertion`.

Este es capaz de generar una representación de la aserción SAML en un objeto `org.w3c.dom.Element`, por lo que la forma en que generamos la cabecera SOAP es diferente en este ejemplo:

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();
Element assertionElem = assertion.toElement(doc);
Element headerElem =
doc.createElementNS("http://docs.oasis-open.org/wss/2004/01/oasis-
 200401-wss-wssecurity-secext-1.0.xsd", "Security");
headerElem.setPrefix("wsse");
headerElem.appendChild(assertionElem);
```

Pero esta aserción SAML está viajando sin ningún tipo de verificación criptográfica. Por ello, podemos firmar la aserción fácilmente incluyendo estas líneas:

```
PublicKey pubKey = getPublicKey();
PrivateKey privKey = getPrivateKey();
Element assertionElem = assertion.sign(pubKey, privKey);
Document doc = assertionElem.getOwnerDocument();
Element headerElem =
doc.createElementNS("http://docs.oasis-open.org/wss/2004/01/oasis
-200401-wss-wssecurity-secext-1.0.xsd", "Security");
headerElem.setPrefix("wsse");
```

```
headerElem.appendChild(assertionElem);
```

Los métodos `getPublicKey()` y `getPrivateKey()` obtendrán una clave pública y privada respectivamente.

3.3. OpenID

OpenID es un estándar de autenticación de usuarios que, desde un punto de vista descentralizado, permite acceder a recursos que tienen desplegado un control de acceso. Además, dichos usuarios podrán acceder a diferentes recursos con una misma identidad digital, proveyendo de esta forma un sistema de *single sign-on*. El servicio donde el usuario se autentica se llama proveedor de OpenID u OpenID provider, mientras que el recurso protegido es conocido como Relaying Party.

La principal diferencia con los sistemas más comunes de federación de identidad digital es que OpenID no proporciona un único punto central para autenticar al usuario, sino que existen numerosos proveedores de OpenID desplegados en Internet que pueden ser utilizados para acceder a cualquier recurso protegido por esta tecnología.

Afortunadamente, contamos con una amplia colección de librerías abiertas para trabajar con OpenID en casi todas las plataformas, tanto en PHP, Python, Javao.NET. Para los ejemplos de esta sección, vamos a utilizar la librería basada en PHP desarrollada por Janrain.

3.3.1. Protocolo de autenticación

El protocolo de autenticación de OpenID disponible en su versión 2, se basa en el envío de mensajes HTTP utilizando tanto los métodos GET como POST.

El flujo de mensajes que se producen a la hora de acceder un usuario a un Relaying Party es el siguiente:

1) El usuario indica al Relaying Party cuál es su identificador OpenID, también llamado *user-supplied identifier*, mediante su navegador o agente. Por regla general, se establecerá este valor mediante un formulario HTML, el cual debería tener como nombre `openid_identifier`. De esta forma, el navegador podrá identificar claramente si existe la posibilidad de autenticarse en el Relaying Party mediante OpenID.

2) Una vez que el Relaying Party normaliza el identificador OpenID, que puede ser un XRI o un URI, realiza el proceso de *discovery* o descubrimiento. De esta forma, obtiene información sobre la URL del proveedor de OpenID del usuario.

3) Opcionalmente, el proveedor de OpenID y el Relaying Party establecen una asociación que resulta en una clave secreta compartida por ambas partes. De esta forma, se elimina la necesidad de estar verificando las firmas digitales cada vez que se solicite la autenticación del usuario.

4) El Relaying Party redirige al usuario a través de su navegador o agente a su proveedor de OpenID por medio de un mensaje de petición de autenticación.

5) El usuario se autentica contra su proveedor de OpenID. Las cuestiones relativas al proceso de la autenticación están fuera del ámbito del protocolo, quedando delegado a las decisiones de los responsables de dicho proveedor.

6) El proveedor de OpenID redirige al usuario, por medio del navegador o agente, al Relaying Party que solicitó dicha petición de autenticación, indicando si la autenticación fue correcta o, en caso contrario, no fue posible.

7) El Relaying Party verifica la información que recibe del proveedor de OpenID a través del navegador o agente. Esta verificación implica comprobar los valores del mensaje que recibe, así como la firma digital de dicho mensaje, bien a través de la clave compartida obtenida en el paso 3 o bien enviando una petición directamente al proveedor de OpenID del usuario.

3.3.2. Ejemplo de Relaying Party

Complementar, o incluso reemplazar, la típica protección con un par usuario y contraseña con acceso mediante OpenID es realmente sencillo. En este ejemplo, aprenderemos a programar en PHP el sistema necesario para incluir autenticación mediante OpenID en nuestras aplicaciones.

Basándonos en el ejemplo de consumo de la librería PHP de JanRain, que se encuentra en su directorio `php-openid/examples/consumer/`, generamos la siguiente estructura de una aplicación web:

```
Auth/
common.php
index.php
try_auth.php
openid_response.php
```

Ese conjunto de ficheros y directorios lo podemos dividir en tres grupos:

1) Librería para trabajar con la tecnología de OpenID. En este grupo está el directorio `Auth` y el fichero `common.php`.

2) Inicio de la solicitud de autenticación OpenID, que corresponde a los ficheros `index.php` y `try_auth.php`.

3) Procesamiento de la respuesta de autenticación, que corresponde al fichero `openid_response.php`.

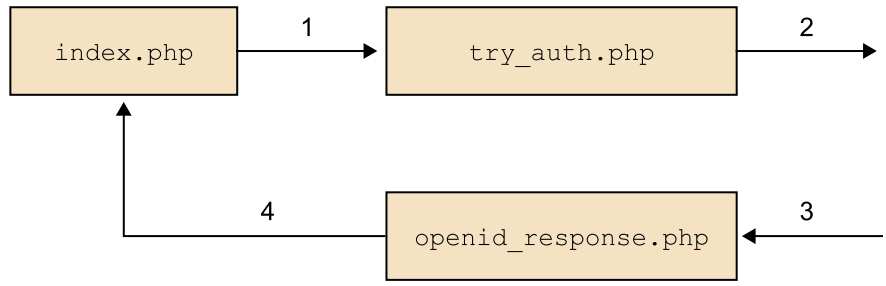
El directorio `Auth` es la librería para trabajar con peticiones hacia proveedores de OpenID y está incluido dentro de la distribución del software de JanRain, mientras que el fichero `common.php` es prácticamente igual al distribuido en dicha librería, salvo que vamos a modificar la función `getReturnTo`:

```
function getReturnTo($file) {
return sprintf("%s://%s:%s%s/".$file,
getScheme(), $_SERVER['SERVER_NAME'],
$_SERVER['SERVER_PORT'],
dirname($_SERVER['PHP_SELF']));
}
```

De esta forma, podemos indicar qué fichero queremos usar para la respuesta del proveedor de OpenID.

El resto de los ficheros del ejemplo definen el flujo de trabajo del nuestro, tal como muestra la figura siguiente.

Diagrama de relación entre partes



El primer paso a la hora de incluir autenticación OpenID en nuestro recurso es incluir un formulario para que el usuario indique cuál es su proveedor de OpenID, como podemos ver en el siguiente ejemplo.

```
<html>
      <head>
          <title>Relaying Party OpenID Example</title>
      </head>
      <body>
<?php
session_start():
if (array_key_exists("logged", $_SESSION)) {
?>
<p>Autenticación correcta. Tu identificador OpenID es:</p>
<p><?php echo $_SESSION["logged"]; ?></p>
<?php
)
```

```
else {
      if (array_key_exists("openid_error_msg", $_REQUEST)) {
?>
<p>Error en la autenticación OpenID:</p>
<p><?php echo $_REQUEST["openid_error_msg"]; ?></p>
<?php
                  }
?>
<p>Necesita autenticarse mediante OpenID.</p>
      <form method="get" action="try_auth.php">
            URL identidad OpenID:
            <input type="hidden" name="action" value="verify" />
            <input type="text" name="openid_identifier" value="" />
            Input type="submit" value="Aceptar" />
      </form>
<?php
}
?>
      </body>
</html>
```

Como vemos en el código, lo primero que se hace es comprobar si existe algún valor de sesión para la clave `logged`. En el procesamiento de la respuesta de autenticación de OpenID, guardaremos para dicha clave, en la sesión, el identificador digital del usuario que nos devuelve su proveedor de OpenID. Si fuera así el caso, muestra en el navegador dicho identificador.

En caso contrario, significa que no está autenticado en el sistema, por lo que se le muestra el formulario donde podrá indicar su proveedor de OpenID. Este hará una petición HTTP con método GET al fichero `try_auth.php`. El parámetro `openid_identifier` contendrá dicho valor de proveedor de OpenID, mientras que el campo `action` tendrá el valor `verify`, que indicará a la librería de OpenID que se quiere iniciar el proceso de autenticación con dicho proveedor.

Además, si al procesar la respuesta de autenticación por parte del fichero `finish_auth.php` hubiera ocurrido algún error, éste lo indicará con el parámetro `openid_error_msg`.

El fichero `try_auth.php` contendrá la lógica que procesará el valor introducido por el usuario para indicar su proveedor de OpenID y, si este tiene un formato correcto, iniciará el proceso de autenticación.

```
<?php
require_once "common.php";
session_start();
```

```
$returnFile = 'openid_response.php';
$openid = $_GET['openid_identifier'];

$consumer = getConsumer();
$auth_request = $consumer->begin($openid);

if (!$auth_request) {
$msg_error = "Authentication error; not a valid OpenID.";
header("Location:            ".$returnFile."?openid_msg_error="
. base64_encode($msg_error));
exit();
}

if ($auth_request->shouldSendRedirect()) {
$redirect_url =             $auth_request->redirectURL(getTrustRoot(),
getReturnTo($returnFile));
if (Auth_OpenID::isFailure($redirect_url)) {
$msg_error = "Could not redirect to server: ".
$redirect_url->message;
header("Location:             ".$returnFile."?openid_msg_error=".
base64_encode($msg_error));
exit();
} else {
// Send redirect.
header("Location: ".$redirect_url);
exit();
}
} else {
$form_id = 'openid_message';
$form_html = $auth_request->htmlMarkup(getTrustRoot(),
getReturnTo($returnFile),
false,
array('id' => $form_id));
if (Auth_OpenID::isFailure($form_html)) {
$msg_error = "Could not redirect to server: ".
$form_html->message;
header("Location:             ".$returnFile."?openid_msg_error=".
 base64_encode($msg_error));
exit();
} else {
echo $form_html;
}
}
?>
```

Como se observa en el código, este fichero procesa el valor del campo `openid_identifier` y lo normaliza, con el objetivo de tener una URI completa a la que realizar la petición. Para realizar dicha petición, primero comprueba si el proveedor de OpenID del usuario soporta OpenID v2 o no, para mostrar un formulario que se ejecuta automáticamente, o bien, para OpenID v1, realizará una petición HTTP con método GET. En esta petición se indica que la respuesta de la solicitud queremos que se procese en el fichero `openid_response.php` de nuestro proyecto web.

```
<?php
require_once "common.php";
session_start();
function escape($thing) {
return htmlentities($thing);
}

function OpenID_finish_auth_success($openid_identity) {
session_start();
$_SESSION["logged"] = $openid_identity;
header ("Location: index.php");
exit;
}

function OpenID_finish_auth_error($return_to, $msg) {
header ("Location: index.php?openid_error_msg=".$msg);
exit;
}
$responseFile = 'openid_response.php';

$consumer = getConsumer();

// Complete the authentication process using the server's response.
$return_to = getReturnTo($responseFile);
$response = $consumer->complete($return_to);

// Check the response status.
if ($response->status == Auth_OpenID_CANCEL) {
// This means the authentication was cancelled.
$msg = 'Verification cancelled.';
OpenID_finish_auth_error($return_to,$msg);
} else if ($response->status == Auth_OpenID_FAILURE) {
// Authentication failed; display the error message.
$msg = "OpenID authentication failed: " . $response->message;
OpenID_finish_auth_error($return_to,$msg);
} else if ($response->status == Auth_OpenID_SUCCESS) {
// This means the authentication succeeded.
$openid = $response->getDisplayIdentifier();
```

```
$esc_identity = escape($openid);
OpenID_finish_auth_success($esc_identity);
}
?>
```

El primer paso para procesar la respuesta de autenticación es especificar que se indicó en la petición que se quería la respuesta en el fichero `openid_response.php`, requerimiento de la librería de Janrain para comprobar que la respuesta es esperada por nuestra aplicación web.

Una vez procesada la respuesta, tenemos el resultado de la autenticación en el atributo `status` del objeto `$response`:

- `Auth_OpenID_CANCEL`: la autenticación fue cancelada por el usuario en el proveedor de OpenID.

- `Auth_OpenID_FAILURE`: la autenticación no fue correcta, por lo que se redirecciona al usuario a `index.php` indicando en el parámetro `openid_error_msg` cuál fue el motivo del fallo.

- `Auth_OpenID_SUCCESS`: la autenticación fue correcta, por lo que se genera un contexto de seguridad en la aplicación web, mediante la función `OpenID_finish_auth_success`, y redirige al usuario a `index.php`. Un contexto de seguridad es un contexto web común pero con un vínculo y variables asociadas al dominio de identidad donde se ha autenticado.

3.4. OAuth

Open Authorization (OAuth) es un estándar abierto que permite incluir seguridad en la autenticación para aplicaciones web y de escritorio.

OAuth implementa la forma de proteger no sólo un recurso web, sino también elementos de otras aplicaciones, como pueden ser las de escritorio o las móviles.

La diferencia entre OAuth y otros sistemas federados es que estos últimos ofrecen una identidad única para acceder a múltiples recursos, mientras que OAuth permite a otras aplicaciones el acceso a un recurso, pero sin dar ningún tipo de información de carácter personal. Esto favorece a las aplicaciones web ofrecer sus servicios sin forzar a los usuarios a exponer sus contraseñas u otras credenciales.

La estructura de autorización y autenticación mediante OAuth se basa en varios elementos básicos:

- **Proveedor de servicio (*service provider*)**. Es la aplicación web donde se encuentran los recursos que están protegidos mediante OAuth y se encarga de denegar o permitir el acceso a los mismos.

- **Usuario (*user*)**. Individuo que tiene una cuenta en la aplicación web proveedora del servicio.

- **Consumidor (*consumer*)**. Aplicación web que utiliza OAuth para acceder al proveedor de servicio en nombre del usuario.

- **Recurso protegido (*protected resource*)**. Datos controlados por el proveedor de servicio con OAuth a los que podrá acceder, mediante autenticación, el consumidor.

Para garantizar la seguridad al dar acceso a los recursos, OAuth define un método para validar la autenticidad de las peticiones HTTP. Este método se denomina *signing request* e intenta solventar los siguientes aspectos:

- Los datos enviados con las peticiones deben enviarse cifrados para evitar que una tercera persona pueda interceptarlos y hacer un uso ilícito de ellos.

- Debe existir un mecanismo que asocie una petición con unas credenciales concretas, ya que de no realizarse así, unas credenciales correctas podrían ser utilizadas en cualquier petición que se enviara.

- Debe permitirse interactuar con dos tipos de credenciales, las del consumidor, que garantizan que este es un consumidor autorizado previamente, y las del usuario, que lo certifican como usuario del proveedor del servicio.

Según el grado de seguridad del protocolo utilizado en la comunicación, el método varía. Si va sobre HTTPS, se sigue el método PLAINTEXT, que delega la mayor parte de la seguridad en la capa HTTPS. Cuando va sobre HTTP, el método de seguridad debe ser mucho más elaborado y es el que vamos a comentar en este subapartado.

Para acreditar al consumidor, se utilizarán la *consumer key* y el *consumer secret*, que identificarán al consumidor ante el proveedor de servicio.

Para identificar al usuario, dispondremos de un *token* y del *token secret*. Estos elementos representarán al usuario, pero diferirán del nombre de usuario y contraseña originales en el proveedor de servicio. Esto permite al proveedor

de servicio y al usuario más control y seguridad, dando acceso al consumidor, y además, permitiendo a un usuario revocar un *token* sin tener que cambiar contraseñas en otras aplicaciones.

Como método para garantizar la integridad, OAuth utiliza firmas digitales en vez de enviar las credenciales completas, verificándose así que el contenido de la petición no se ha modificado en el camino entre una aplicación y otra. La garantía de confianza dependerá del algoritmo de firma que se utilice y de la forma de aplicación del mismo.

Como dicha firma electrónica no verifica la identidad del que envía el mensaje, utilizamos la firma combinada con el secreto compartido. Para esto, será necesario que ambos elementos acepten un secreto que sólo conocerán ellos.

Para no permitir reenvíos por terceras personas, OAuth incorpora dos elementos: un número identificador para cada petición que el consumidor envíe al proveedor de servicio y una marca de tiempo que permitirá a los proveedores de servicio mantener los identificadores por un tiempo limitado.

Existen tres métodos de firma diferentes que engloban las características comentadas en este subapartado:

- **PLAINTEXT**. Es el más simple de los tres y sólo se recomienda utilizar sobre HTTPS.
- **HMAC-SHA1**. Combina HMAC, que ofrece secreto compartido simétrico y SHA1 como algoritmo *hash*.
- **RSA-SHA1**. Es el más complejo de los métodos y combina RSA como mecanismo de seguridad de las claves y SHA1 como algoritmo de *hash*.

3.4.1. Protocolo de autenticación

La autenticación mediante OAuth es el proceso mediante el cual los usuarios conceden acceso a sus recursos protegidos sin compartir sus credenciales con el consumidor.

En vez de utilizar las credenciales del usuario en las peticiones a recursos protegidos, OAuth utiliza los *tokens* generados por el proveedor de servicio que denominamos *request token* y *access token*:

- ***Request token***. Utilizado por el consumidor para pedir al usuario acceso a los recursos protegidos. Una vez el *request token* es autorizado por el usuario, se intercambia por un *access token*, que debe utilizarse sólo una vez y no podrá usarse para otro cometido.

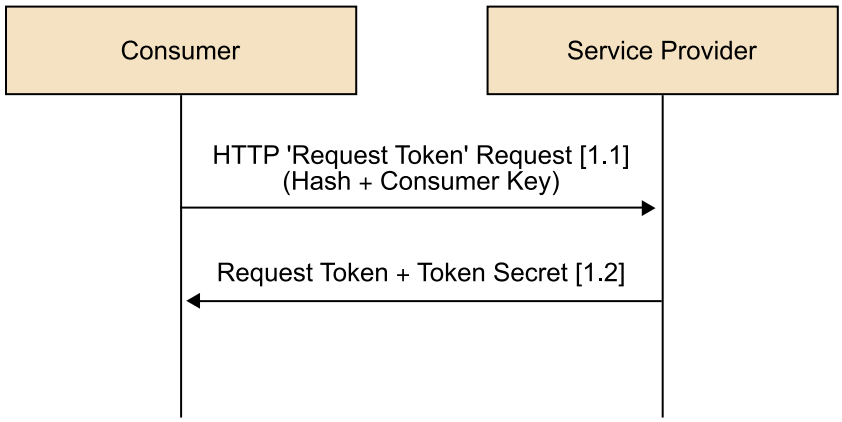
- ***Access token***. Utilizado por el consumidor para acceder a los recursos protegidos en nombre del usuario. Puede limitar a más de un recurso y tener un tiempo de vida limitado. Sólo el *access token* podrá usarse para acceder a los recursos protegidos, siendo éste susceptible de ser revocado.

La autenticación de OAuth se hace en tres pasos:

Paso 1. El consumidor obtiene un *request token* sin autorizar mediante el envío de una petición HTTP a la URL dispuesta para ello en el proveedor de servicio. La petición deberá estar firmada por el consumidor y contener la *consumer key*. Una vez recibida, el proveedor de servicio verifica la firma y la clave del consumidor y, en caso de ser correctos, generará un *request token* y un *token secret* y los devolverá al consumidor.

En el diagrama de flujo de la figura siguiente, vemos cómo se comunican el consumidor y el proveedor de servicio en esta primera fase.

Flujo de mensajes OAuth en una primera fase



La petición HTTP deberá ser POST y tendrá el siguiente formato.

```
GET /directorio/recursos?nombre=foto1.jpg
HTTP/1.1
Host: http://serviceprovider.com
Content-Type: application/atom+xml
Authorization:
OAuth oauth_token="1%2Fab3cd9j4ks73hf7g",
oauth_signature_method="RSA-SHA1",
oauth_signature="wOJIO9AvZbTSMK%2FPY%3D...",
oauth_consumer_key="ejemplo.com",
oauth_timestamp="137131200",
oauth_nonce="4572616e48616d6d",
oauth_version="1.0"
```

La respuesta contendrá al menos los parámetros siguientes.

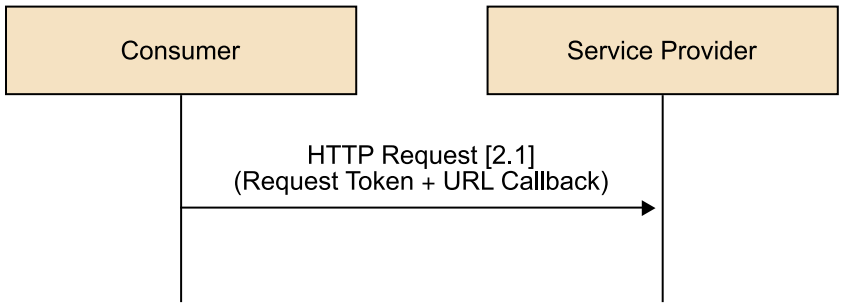
```
oauth_token: 1%2Fab3cd9j4ks73hf7g
oauth_token_secret: 47ba47e0048b7f2105db67df18ffd24bd072688a
```

Como vemos, el `oauth_token` contiene el mismo valor que el que tenía el parámetro `oauth_token` en la petición, ya que representa al *request token*.

Paso 2. El consumidor redirecciona al usuario a la URL que el proveedor de servicio ha dispuesto para la autenticación, enviando a su vez una petición HTTP con el *request token* y la URL a la que deberá devolverse al usuario una vez terminado el proceso. En este punto, el usuario se autenticará en el proveedor de servicio y establecerá la política de acceso que desee para los recursos protegidos. Una vez terminado, el proveedor de servicio devolverá el usuario al consumidor tal y como se estableció en la petición.

En el diagrama de la figura siguiente podemos observar el flujo de peticiones que se realizan en esta segunda etapa.

Flujo de mensajes OAuth en la segunda etapa



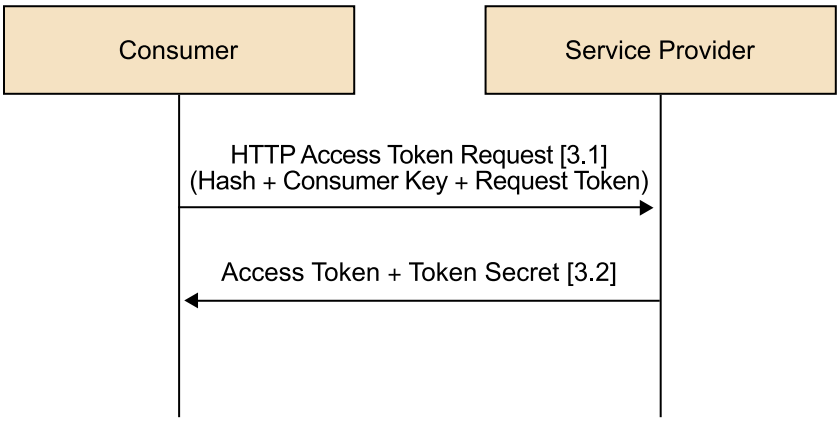
El *request token* será el generado anteriormente, y la URL *callback* será la URL a la que deberá redirigir el proveedor de servicio al usuario una vez tratados los permisos de acceso de los recursos protegidos.

Paso 3. Una vez autorizado, el consumidor intercambia el *request token* por un *access token* que sea capaz de acceder a los recursos protegidos. Para ello, el consumidor hace una petición HTTP de un *access token* a la URL especificada para ello por el proveedor de servicio. Esta petición deberá estar firmada mediante el método *signing request* especificado anteriormente y contendrá la clave del consumidor, el *request token* y la firma entre otros parámetros.

Para dar acceso a un recurso, el proveedor de servicio deberá asegurarse de que la firma de la petición se verifique, que ese *request token* no haya sido intercambiado ya y que la clave del consumidor coincida con la del *request token* especificado. En caso de que esto se cumpla, el proveedor de servicio generará un *access token* y un *token secret*, el cual es un secreto que utiliza el consumidor y que lo acredita como propietario de un *token* determinado, devolviéndolos al consumidor. Estos datos se almacenarán y serán utilizados para firmar las peticiones de los recursos protegidos.

En el diagrama de la figura siguiente podemos ver el flujo de mensajes que se realiza en esta tercera etapa de la comunicación entre el consumidor y el proveedor de servicio.

Flujo de mensajes OAuth en la tercera etapa



La petición HTTP contendrá la firma digital, la clave del consumidor y el *request token* que se desea intercambiar (que será similar al intercambiado en los pasos anteriores).

La respuesta del proveedor de servicio deberá incluir los siguientes parámetros.

```
oauth_token: 1%2Fab3cd9j4ks73hf7g
oauth_token_secret: 34ba34e0048bdse6505db67df165fd24b7672688a
```

El parámetro `oauth_token` contendrá el *access token* generado por el proveedor de servicio, mientras que el parámetro `oauth_token_secret` acreditará al consumidor identificándolo.

3.4.2. Ejemplo de un consumidor

A continuación, mostramos un ejemplo en el que explicamos cómo debe implementarse un consumidor para poder interactuar con un proveedor de servicio mediante OAuth.

Nota

La librería utilizada en este subapartado es la `oauth-php`.

En primer lugar, debe crearse una base de datos que le sirva al consumidor como medio de almacenamiento de la información obtenida con los *tokens* y las peticiones HTTP. Para esto, utilizamos el *script* PHP que encontraremos en `library/store/mysql/install.php` y que importa la estructura de la base de datos que nos da la librería y que se encuentra en `library/store/mysql/mysql.sql`.

Una vez realizado esto, hay que obtener una instancia que permita acceder a la base de datos, lo cual haremos de la siguiente forma.

```
$bd = OAuthStore::instance('tipoBD', $opciones);
```

Donde `tipoBD` será el tipo de base de datos (por ejemplo, MySQL) y `$options` un *array* con los atributos servidor, contraseña, nombre de usuario y base de datos.

Después, añadiremos el servidor a la instancia de la base de datos para así poder interactuar con él. Primero creamos un *array* con los datos del consumidor respecto a un proveedor de servicio.

```
$descserv = array( 'consumer_key' => 'Clave_Del_Consumidor_En_El_SP',
'consumer_secret' => 'Secreto_del_Consumidor',
'server_uri' => 'http://sp_ejemplo/',
'signature_methods' => array('HMAC-SHA1','PLAINTEXT'),
'request_token_uri' => 'http://sp_ejemplo/request_token',
'authorize_uri' => 'http://sp_ejemplo/authorize',
'access_token_uri' => 'http://sp_ejemplo/access_token' );
```

Luego declaramos e inicializamos un identificador del usuario cuyos recursos queremos utilizar, y almacenamos en la base de datos el servidor que tenemos en el *array* `$descserv`.

```
$ident_user = 1;
$consumer_key = $bd->updateServer($descserv, $ident_user);
```

Para tratar el proceso de autenticación, necesitamos declarar e inicializar la URL de regreso que se le tiene que enviar al proveedor de servicio, a la que le añadimos la clave del consumidor codificada según el RFC 1738 y el identificador del usuario correspondiente.

```
$callback_uri = 'http://consumidor.com/callback?
consumer_key='.rawurlencode($consumer_key).'
&usr_id='.intval($ident_user);
```

Paso 1. El consumidor obtiene un *request token* sin autorizar enviando una petición al proveedor de servicio. Esto se lleva a cabo mediante la siguiente directiva.

```
$token = OauthRequester::requestRequestToken($consumer_key,
 $ident_user);
```

Paso 2. El consumidor redirecciona al usuario a la URL que el proveedor de servicio ha dispuesto para la autenticación en caso de haber recibido un *token* autorizado en el paso 1, enviando al redireccionar el *request token* y la URL, que deberá devolverse al usuario una vez terminado el proceso.

```
if (!empty($token['authorize_uri'])){
```

```
if (strpos($token['authorize_uri'], '?')) {
$uri = $token['authorize_uri'] . '&';
}else {
$uri = $token['authorize_uri'] . '?';
}
$uri .= 'oauth_token='.rawurlencode($token['token']).'
&oauth_callback='.rawurlencode($callback_uri);
}else {
$uri = $callback_uri . '&oauth_token='.rawurlencode($token['token']);
}
header('Location: '.$uri);
exit();
```

Una vez que el usuario ha sido autorizado y el proveedor de servicio lo redirecciona a la dirección de *callback* proporcionada, se solicita el intercambio del *request token* por el *access token*.

```
$consumer_key = $_GET['consumer_key'];
$oauth_token = $_GET['oauth_token'];
$ident_user = $_GET['usr_id'];
OAuthRequester::requestAccessToken($consumer_key, $oauth_token,
$ident_user);
```

Tras realizar estos pasos, ya estamos listos para pedir el acceso a los recursos protegidos en cuestión mediante la directiva.

```
$request = new OAuthRequester($uri_request, 'GET', $parametros);
```

Donde `$uri_request` será la URI del proveedor de servicio a la que accederemos (que se nos facilitó al registrar el servidor); el segundo parámetro podrá ser tanto GET como POST, y `$parametros` será un *array* donde se añadirán los parámetros requeridos por el proveedor de servicio para funcionar.

Por último, sólo nos quedará hacer la petición mediante la siguiente línea de código.

```
$result = $req->doRequest($user_id);
```

Siendo el resultado un *array* con los parámetros `code` (un entero), `headers` (un *array*) y `body` (una cadena) que dependerán del proveedor de servicio al que se está accediendo.

3.5. XACML

Cuando estamos hablando de gestionar centenares o miles de sistemas de información, la creación y distribución de políticas de seguridad no es una tarea fácil. Muchos de ellos son diferentes entre sí y tienen una forma de imple-

mentar las políticas que distan mucho los unos de los otros. Cualquier cambio, aunque sea mínimo, implica que las decenas de administradores existentes entiendan la política correctamente y modifiquen todos y cada uno de los sistemas mencionados en la misma línea. En este contexto, existen muchas probabilidades de cometer un error.

Con este objeto, la organización OASIS creó XACML[9], un lenguaje basado en XML para el almacenamiento y distribución de políticas de control de acceso. Todavía no es un estándar ampliamente aceptado y tiene otros competidores como PERMIS, pero también un ecosistema de desarrollo y soporte interesantes, como puede ser el proyecto HERAS.

[9]XACML es la abreviatura de *extensible access control markup language*.

Enlace de interés

El proyecto PERMIS es un motor de decisión RBAC basado en políticas. Se puede consultar en la siguiente dirección web: http://www.openpermis.org.

HERAS

HERAS es un framework de soporte a XACML 2.0 y para la seguridad de aplicaciones web. Para saber más sobre HERAS podéis visitar la siguiente dirección web: http://www.herasaf.org.

Existen muchas complementariedades con SAML; los dos son lenguajes de pregunta/respuestas, basados en XML, con los mismos términos (sujeto, objeto, etc.), pero mientras que SAML ofrece un lenguaje para traspasar la identidad y el control de acceso, XACML dicta qué hacer con la información. Se puede apoyar, por ello, en múltiples atributos del usuario o del contexto, por lo que a priori puede parecer un lenguaje más complicado.

Políticas

Ejemplos de políticas podrían ser:

- Sólo se puede acceder a este conjunto de documentos desde las 9:00 h a las 17:00 h.
- Unicamente los miembros del departamento de recursos humanos tienen acceso de modificación, mientras que el resto pueden leerlo únicamente.
- Si accedes a través de HTTPS tienes acceso, de otra forma no.
- Si accedes a través de HTTPS y perteneces al departamento de recursos humanos tienes acceso de escritura, de otra forma, únicamente tienes acceso de sólo lectura.

Normalmente estas políticas se distribuyen a los *policy decision points* (PDP), puntos dentro de una arquitectura de identidad que son los encargados de tomar decisiones sobre la identidad y el control de acceso. Dentro de una estructura real de identidad puede haber múltiples puntos de este tipo.

Los diferentes componentes de XACML son:

- ***Policy***, representa una política única de control de acceso, expresada a través de una serie de reglas. Es la base para la autorización de la decisión.

- ***PolicySet***, aglutina una serie de políticas o conjunto de políticas y procedimientos de cómo combinar el resultado de sus evaluaciones. Sirve principalmente para combinar diferentes políticas en una única.

- ***Rule***, expresión booleana que puede evaluarse de forma aislada. Puede reutilizarse para varias políticas.

- ***Target***, define un conjunto de objetos, sujetos y acciones a los que aplica una determinada regla.

- ***Obligation***, operación definida en una política (o conjunto de políticas) que debe ser realizada junto a la evaluación de las decisiones de autorización.

- ***Condition***, expresión que evalúa a `True`, `False` o `Indeterminate`.

- ***Effect***, la consecuencia de una regla satisfecha, permitida o denegada.

Ejemplo

Imaginemos entonces que tenemos una política que dicta que:

"Para acceder a un servidor que se llama *SampleServer*, el usuario debe haberse identificado y autenticado (*login*) contra el sistema y debe hacerlo entre las 09:00 h y las 17:00 h".

La forma de la política sería como la que sigue:

```
<Policy PolicyId="SamplePolicy" RuleCombiningAlgId="urn:oasis:names:
  tc: xacml:1.0: rule-combining-algorithm: first-applicable">

(POLICY PARAMETERS)

(RULE1)

(RULE2)

...

<! -- A final, "fall-through" Rule that always Denies -->
<Rule RuleId="FinalRule" Effect="Deny"/>

</Policy>
```

El *target* se expresaría:

```
<! -- This Policy only applies to requests on the Sample Server -->
<Target>
<Subjects>
<AnySubject/>
</Subjects>
<Resources>
<ResourceMatch MatchId="urn:oasis:names: tc:xacml:1.0: function:
 string-equal">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
 SampleServer</AttributeValue>
<ResourceAttributeDesignator
 DataType="http: //www. w3.org/2001/XMLSchema#string"AttributeId=
 "urn: oasis: names: tc:
 xacml:1. 0: resource: resource-id"/>
</ResourceMatch>
</Resources>
<Actions>
<AnyAction/>
</Actions>
</Target>
```

La regla de que ha de estar identificado y autenticado contra el sistema seguiría el siguiente patrón.

```
<! -- Rule to see if we should allow the Subject to login -->

<Rule RuleId="LoginRule" Effect="Permit">
<! -- Only use this Rule if the action is login -->
<Target>
<Subjects>
<AnySubject/>
</Subjects>
<Resources>
```

```
<AnyResource/>
</Resources>
<Actions>
<ActionMatch MatchId="urn:oasis:names: tc:xacml:1.0: function:
 string-equal">
<AttributeValue  DataType="http: //www. w3. org/2001/
 XMLSchema#string">login</AttributeValue>
<ActionAttributeDesignator  DataType="http://www.w3.org/2001/
 XMLSchema#string"
 AttributeId="ServerAction"/>
</ActionMatch>
</Actions>
</Target>
```

Y la condición de que sea entre las 9:00 h y las 17:00 h:

```
<! -- Only allow logins from 9am to 5pm -->

<Condition FunctionId="urn: oasis: names: tc: xacml: 1.0: function:
 and">
<Apply FunctionId="urn: oasis: names: tc: xacml: 1.0: function:
 time-greater-than-or-equal">
<Apply FunctionId="urn: oasis: names: tc: xacml: 1.0: function:
time-one-and-only">
<EnvironmentAttributeSelector DataType="http://www.w3.org/2001/
 XMLSchema#time"
 AttributeId="urn:oasis: names: tc: xacml: 1. 0: environment:
 current-time"/>
</Apply>
<AttributeValue DataType="http: //www. w3. org/2001/XMLSchema#time">
 09:00:00</AttributeValue>
</Apply>
<Apply FunctionId="urn: oasis: names: tc: xacml: 1. 0: function:
 time-less-than-or-equal">
<Apply FunctionId="urn: oasis: names: tc: xacml: 1. 0: function:
 time-one-and-only">
<EnvironmentAttributeSelector DataType="http://www.w3. org/2001/
 XMLSchema#time"
 AttributeId="urn:oasis: names: tc: xacml: 1. 0: environment:
 current-time"/>
</Apply>
<AttributeValue DataType="http: //www.w3.org/2001/XMLSchema#time">
 17: 00:00</AttributeValue>
</Apply>
</Condition>
</Rule>
```

3.6. SPML

SPML son las siglas de *service provisioning markup language*, un lenguaje basado en XML para el aprovisionamiento automático de identidades en sistemas de información y la gestión de una identidad a lo largo de su ciclo de vida, desde crearla a revocarla, pasando por la modificación de la misma.

Se pueden identificar tres roles diferentes:

- ***Requesting authority*** **(RA)**: la entidad que realiza la petición de aprovisionamiento.

- ***Provisioning service provider*** **(PSP)**: software capaz de responder a peticiones SPML.

- ***Provisioning service target*** **(PST)**: proveedor de las identidades. Este rol puede estar unido a PSP, pero mientras que PSP necesita entender SPML, PST no.

SPML constituye un lenguaje de pregunta/respuesta, como SAML. Para ello necesitamos que RA y PSP establezcan una relación de confianza, que puede ser representada también a través de SAML, introduciendo las aserciones correspondientes en la cabecera y las instrucciones SPML en el *payload*.

Tiene básicamente cinco operaciones:

1) `<addRequest/>`, utilizado para crear una cuenta.

2) `<modifyRequest/>`, se emplea para actualizar una cuenta.

3) `<deleteRequest/>`, utilizado para pedir el borrado de una cuenta.

4) `<searchRequest/>`, se utiliza para interrogar al PSP acerca de cuentas y sus propiedades.

5) `<schemaRequest/>`, permite pedir a la RA el esquema de aprovisionamiento de un PSP.

Ejemplo

Como ejemplo, mostramos cómo sería una petición de creación de una cuenta:

```
<addRequest>
      <attributes>
            <attr name="objectclass">
                  <value>urn:...:SPML:interop:interopUser</value>
            </attr>
            <attr name="cn">
                  <value>Mario López</value>
            </attr>
            <attr name="mail">
                  <value>mlopez@micompania.com</value>
            </attr>
            <attr name="registrationTime">
                  <value>21-Jul-2011 12:00</value>
            </attr>
      </attributes>
</addRequest>
```

A lo que PSP podría responder con:

```
<addResponse result = "urn:...:SPML:1:0#success">
            <identifier type="urn:...:SPML:1:0#EMailAddress">
                  <id>mlopez@micompania.com</id>
            </identifier>
            <attributes>
                  <attr name="mailBoxLimit">
                        <value>50MB</value>
                  </attr>
            </attributes>
</addResponse>
```

3.7. Otras tecnologías delegadas de control de acceso

Durante la historia de los sistemas de información, la seguridad ha generado un gran interés dando lugar a diferentes técnicas, métodos, tecnologías y teorías. Dentro de la gestión de la identidad y el acceso, no ha sido diferente y se han propuesto distintos sistemas en este ámbito.

Sin embargo, hemos de diferenciar los sistemas existentes en materia de control de acceso, como pueden ser Kerberos o RADIUS, de los sistemas federados, que obedecen necesariamente a una realidad diferente, con múltiples dominios de identidad/seguridad que pueden pertenecer a diferentes organizaciones, con políticas y aproximaciones a la gestión de las identidades dispares.

Sin embargo, es interesante que repasemos alguna de estas tecnologías con el objetivo de hacer un ejercicio de diferenciación respecto a lo que quiere decir realmente la federación de identidades.

3.7.1. Kerberos

Kerberos posiblemente es el sistema de control de acceso más conocido y utilizado; por ejemplo, es empleado por los actuales sistemas Windows y durante años, por los sistemas Unix.

> Concretamente, Kerberos es un sistema de gestión de acceso para sistemas heterogéneos, además de un sistema de *single sign-on* y de distribución de claves. Provee de capacidades de autenticación y gestión de acceso a una serie de *principals*, es decir, a los componentes que hay que proteger dentro del sistema.

El componente más importante de Kerberos es el Key Distribution Center (KDC), en el que todos los elementos confían y que es el encargado de guardar los secretos que servirán para realizar los diferentes procesos de seguridad que gestiona Kerberos. En líneas generales, cuando alguien quiere acceder a un recurso, por ejemplo la impresora, le pide al KDC con sus credenciales el acceso. El KDC mira la tabla de capacidades del usuario y, si puede acceder, le da un ticket para que se lo presente a la impresora en el momento de solicitar la impresión.

Aunque Kerberos gestiona diferentes recursos, no maneja diferentes dominios de identidad, con diferentes ID, políticas, estructuras, etc., sino que los pone en común y se abstrae de cómo esté hecho cada repositorio de identidad. En este caso, los recursos que hablen del protocolo de Kerberos estarán supeditados a un único dominio de identidad.

Normalmente, se suele utilizar dentro de una misma empresa aunque algunos fabricantes han intentado extender el protocolo para darle más alcance.

SESAME

Secure european system for applications in a multi-vendor environment (SESAME), por ejemplo, es una nueva aproximación a Kerberos que quiere corregir algunos aspectos de este último, pero que todavía no dispone de los elementos que permitan constituir un sistema para la federación de identidades.

3.7.2. RADIUS

El protocolo RADIUS[10] (o su versión posterior Diameter), y después TACACS[11], permiten la autenticación remota de personas y sistemas dándoles después acceso a una serie de recursos, previa autenticación negociada con protocolos como *password authentication protocol* (PAP), *challenge-handshake authentication protocol* (CHAP) o *extensible authentication protocol* (EAP). Radius utilizaba conexiones PPP (*point-to-point protocol*) y más tarde TACACS permitió conexiones TCP.

(10)RADIUS es la sigla de *remote authentication dial in user service.*

(11)TACACS es la sigla de *terminal access controller access-control system.*

Aunque permite el acceso a sistemas remotos, no federa la identidad. Para conectar a diferentes sistemas RADIUS (o TACACS) se tendrán que utilizar tantas credenciales como sistemas queramos utilizar. En este aspecto, se trata de un protocolo de acceso remoto pero no de federación de identidad.

3.7.3. 802.1x

El 802.1x es un estándar de la Institute of Electrical and Electronics Engineers (IEEE) para la autenticación de dispositivos basada en puertos en una red conmutada. En este y otros muchos protocolos de esta índole, permiten la autenticación a dominios de identidad diferentes. No obstante, 802.1x no federa la identidad y se encargan únicamente de la operación de autenticación.

Resumen

La identidad, como concepto fundamental existente en nuestras vidas, ha sido objeto de muchas aproximaciones en su traslado al mundo virtual. Diferentes aportaciones han intentado, más o menos globalmente, solucionar diferentes aspectos de la gestión de la identidad y el acceso a los recursos.

Durante este módulo hemos podido ejemplarizar inicialmente algunos escenarios de uso de la federación de identidades, como concepto real que solventa alguno de los problemas existentes en la interacción entre diferentes dominios de identidad/seguridad. A partir de ello, hemos definido unos conceptos base que nos han permitido unificar términos que no siempre están claros en la literatura.

Seguidamente, hemos repasado los diferentes patrones de federación existentes y que conformarán el marco de relación básica entre las partes. Dependiendo de este modelo de relación, las técnicas, políticas, métodos y estándares pueden diferir, así como la propia estrategia de la empresa en cuanto a gestión de la identidad y el acceso se refiere. De esta forma, hemos podido acercarnos a los patrones *ad-hoc*, el *hub-and-spoke* y la Federation Network.

Seguidamente, y aun con la fragmentación existente en el mundo de la federación de identidades (o por lo menos en alguna de sus operaciones), hemos revisado las iniciativas más importantes que pretenden afrontar el problema con el objetivo de crear estándares y especificaciones para resolver los diferentes retos planteados en la gestión de la identidad y el acceso federado. Las más relevantes descritas en este módulo son las iniciativas de Microsoft/IBM, OASIS, Liberty Alliance y Internet2 con Shibboleth.

Por último, hemos revisado las tecnologías y protocolos más importantes del mundo de la federación de identidades. SAML ha ocupado gran parte de nuestro discurso, pues se trata de un protocolo que, poco a poco, se va imponiendo como medio de transmisión de la identidad y el control de acceso y también por la gestión de atributos que hace. Seguidamente hemos visto cómo la federación de identidades se transporta más allá del mundo de las personas, concretamente al de los servicios web, describiendo el estándar WSS. A continuación, y siguiendo con la descripción de tecnologías, hemos afrontado OpenID como método popular distribuido para la gestión de identidades, OAuth para la autorización, XACML para la definición y transmisión de políticas de identidad y gestión de acceso y SPML como lenguaje para el aprovisionamiento de sistemas de identidad. Finalmente, hemos completado este apartado tecnológico poniendo en contexto las tecnologías descritas con otras ya existentes en el mundo de los sistemas de información como Kerberos.

El viaje en el que nos hemos introducido a lo largo de este módulo constituye, en sí mismo, un reto emocionante y de constante evolución. La explosión de iniciativas y diferentes aproximaciones podrían hacer pensar en una incertidumbre futura, pero se ve ampliamente por la solidez de algunas iniciativas y sus posibilidades.

Actividades

1. Instalad OpenAM, que es un sistema de control de acceso de código libre, e intentad federarlo con alguno de los proveedores de servicio existentes de OpenID, como Google o Yahoo.

2. Las identidades de las personas siempre ha sido un bien muy preciado por las grandes compañías. Después de iniciativas como la de Microsoft Passport, Facebook connect quiere aprovechar la fuerza de una red social de más de 600 millones de usuarios para vincular los procesos de autenticación de terceros.

Tratad de buscar información acerca de Facebook connect, clasificadla en alguno de los patrones que hemos visto y haced algunas pruebas de autenticación desde vuestra propia web de pruebas.

3. Ha habido multitud de iniciativas de *single sign-on*: OpenSSO (ahora OpenAM), JBoss SSO, Ubuntu SSO, etc. Tratad de buscar las iniciativas existentes y agrupadlas en las corrientes que hemos visto durante el módulo. ¿Se parecen más a las de Oasis? ¿O quizás más a Liberty Alliance? ¿Son propietarias?

Glosario

array *m* Colección ordenada de objetos

atributo *m* Característica de una entidad

base 64 *f* Sistema de codificación donde la base es de 64 posiciones, en contra de la decimal habitual que es de diez.

CHAP *m* Iniciales de *challenge/response authentication protocol* que sirve para autenticar a dispositivos a través de un protocolo de reto/respuesta.

EAP *m* Iniciales de *extensible authentication protocol*, es un *framework* de autenticación normalmente utilizado en redes inalámbricas.

HMAC *m* Iniciales de *hash-based message authentication code*. En criptografía, una construcción para calcular un *message authentication code* (*MAC*).

HTTP *m* Iniciales de *hyper-transfer transport protocol*. Protocolo de nivel 4 para el transporte de servicios en una pila IP.

HTTP GET *m* Respuesta HTTP donde los parámetros van en la URL.

HTTP POST *m* Respuesta HTTP donde los parámetros se envían en el *payload* del paquete.

HTTPS *m* Sistema HTTP seguro.

IETF *m* iniciales de Internet Engineering Task Force, este organismo desarrolla y promueve estándares de Internet.

Java *m* Lenguaje de programación.

login *m* Proceso de autenticación a través de usuario y contraseña, generalmente.

MySQL *f* Base de datos de código libre.

.NET *m* Lenguaje de programación.

PAP *m* Iniciales de *password authentication protocol*. Protocolo para autenticarse contra un sistema remoto.

PHP *m* Lenguaje de programación.

PPP *m* Iniciales de *point to point protocol*, es un protocolo utilizado para la conexión a sistemas remotos.

Python *m* Lenguaje de programación

request/response *f* Petición/respuesta, corresponde a un modelo de programación interrogativo.

RFC *m* Iniciales de *request for comments*, estándares *de facto* colaborativos que emite IETF.

SHA-1 *m* Iniciales de *secure hash algorithm*, algoritmo criptográfico de *hash*.

SOAP *m* iniciales de *simple object access protocol*, este protocolo es utilizado para la llamada a objetos.

thread *m* Hilo de ejecución que comparte recursos computacionales con el proceso.

token *m* Elemento que sólo puede tener una entidad en un mismo momento y que es diferente de los demás *tokens*, es identificable.

URI *m* Iniciales de *uniform resource identifier*, dentro de una URL, camino unívoco al objeto referenciado.

URL *m* Iniciales de *uniform resource locator*. Identificador global único de un recurso en la web.

web service *m* Servicio disponible a través de HTTP/HTTPS.

XML *m* iniciales de *extensible markup language*. Un lenguaje de marcado sobre base de un fichero plano.

XRI *m* iniciales de *extensible resource identifier*. Es un nuevo sistema de identificación en Internet, diseñado específicamente para identidades digitales de dominio cruzado.

Bibliografía

(ISC)2 (2007). *Official (ISC)2 Guide to the CISSP CBK*. Nueva York: Auerbach Publications.

"Authentication, Authorization and Accounting (aaa)" (n.d.). *IETF*. Retrieved 2011-22-07 from http://datatracker.ietf.org/wg/aaa/charter/

Backhouse, J. (2006). "Interoperability of identity and identity management systems. Data Protection and Data Security". *Datenschutz und Datensicherheit* (vol. 30, núm. 9, págs. 568-570).

Backhouse, J.; Halperin, R. (2008). "Approaching interoperability for identity management systems". En: K. Rannenberg; D. Royer; A. Deuker. *The Future of Identity in the Information Society: challenges and* opportunities (págs. 245-268). Berlín y Heidelberg: Springer.

Baldwin, A.; Casassa, M.; Beres, Y.; Shiu, S. (2008). "On Identity Assurance in the Presence of Federated Identity Management Systems". Actas del 2007 ACM Workshop on Digital Identity Management (DIM'07) (págs. 27-35). Alexandria, VA, USA.

EduRoam (n.d.). *EduRoam*. Retrieved 2011-22-07 from http://www.eduroam.es

Harris, S. (2008). *All in one CISSP Exam Guide*. Nueva York: McGraw Hill.

Internet 2 (n.d.). *Internet 2*. Retrieved 2011-22-07 from http://www.internet2.edu

Internet 2 (n.d.). *Shibboleth*. Retrieved 2011-22-07 from http://shibboleth.internet2.edu

Keuleuven Univ. (n.d.). *SESAME in a Nutshell*. Retrieved 2011-19-06 from http://www.cosic.esat.kuleuven.be/sesame/html/sesame_what.html

Liberty Alliance (n.d.). *Liberty Alliance Project*. Retrieved 2011-07-07 from http://projectliberty.org/liberty/content/download/3063/20504/file/liberty-cross-framework-v1.1.pdf

Melgar, D.; Hondo, M.; Nadalin, A. (2002-01-12). *Web Services Security*: *Moving up the stack*. Retrieved 2007-22-07 from IBM DeveloperWorks: http://www.ibm.com/developerworks/library/ws-secroad/

MIT (n.d.). *MIT Kerberos Papers and Documentation web page*. Retrieved 2011-19-06 from http://web.mit.edu/kerberos/papers.html/

OASIS (n.d.). *OASIS*. Retrieved 2011-22-07 from Advanced Open Standards for Information Society: http://www.oasis-open.org

Open PERMIS Project (n.d.). *Open PERMIS Project*. Retrieved 2011-22-07 from http://www.openpermis.org/

Proyecto HERAS (n.d.). *Proyecto HERAS*. Retrieved 2011-22-07 from http://www.herasaf.org/

RFC 2865 (n.d.). *IETF*. Retrieved 2011-22-07 from http://www.faqs.org/rfcs/rfc2865.html

Satchell, C.; Shanks, G.; Howard, H.; Murphy, J. (2006). *Knowing Me Knowing You - User Perceptions of Federated Digital Identity Management Systems*. Sweeden: ECIS.

Stoneburner, G. (n.d.). *Underlining Technical Models for Information Technology Security*, *Recommendations of the National Institute of Standards and Techonology*. Retrieved 2011-19-06 from http://csrc.nist.gov/publications/nistpubs/800-33/sp800-33.pdf

Windley, P. J. (2005). *Digital Identity*. Sebastopol (California): O'Reilly Media, Inc.