Módulo 10 Tarea individual

Identificación en Internet

Máster en Desarrollo Seguro y DevSecOps

Campus Internacional Ciberseguridad

José María Canto Ortiz

Tabla de contenido

Tabla de ilustraciones

Introducción

Tras el reciente incidente de *ramsonware* padecido por la empresa CompuSec S.L., ya solucionado por nuestros compañeros de respuesta ante incidentes y dentro de la estrategia de mejora de su infraestructura de seguridad, se están realizando algunas recomendaciones que quedan recogidas una serie de documentos, en este que nos ocupa se trataran de explicar la identificación de usuarios a través de Internet y los posibles problemas a tener que solventar.

La empresa CompuSec S.L. en su proceso de crecimiento está desarrollando una serie de guías de como mejorar los distintos proyectos que tiene por delante y los que puedan venir en un futuro. Dado que se está desarrollando una serie de aplicaciones webs y APIs que quizás en un futuro puedan consultarse desde aplicaciones en dispositivos móviles, en este se tratará de explicar los problemas o ataques que puedan tener una mala implementación de alguno de los protocolos de autorización y autentificación en Internet, así como buenas prácticas a la hora de dicha implementación.

Como se ha indicado, están en desarrollo varias aplicaciones, todas ellas, obviamente, necesitarán que la persona se identifique. Para evitar que cada persona, ya sean usuarios internos de la empresa, como clientes, tengan que identificarse en cada una de estas aplicaciones, la mejor estrategia es la de introducir la autentificación mediante S*ingle Sign On (SSO)*, traducido como inicio de sesión único.

El inicio de sesión único es tipo de autentificación en el cual un usuario inicia sesión en un sistema y se le concede de forma automática acceso a otros servicios.

Esto se encuentra comúnmente en entornos empresariales donde los empleados acceden a numerosas aplicaciones y servicios a diario. En lugar de que un empleado cree un conjunto separado de credenciales para cada aplicación, simplemente inicia sesión una vez y puede acceder a cualquier aplicación que haya configurado el departamento de TI.

Los ejemplos claros de esta estrategia que a cualquiera se le pueden venir a la cabeza son Google, con un solo inicio de sesión accedemos a Gmail, Drive, Maps, etcétera y Microsoft igual para acceder a sus distintos servicios como Outlook, Learn, Office, Teams, etcétera; a esto quizás se podría añadir a Amazon, que iniciando una única vez podemos acceder a la propia tienda, a Prime o a Twitch.

Ilustración 1: Single Sign-On

El enfoque de inicio de sesión único posee algunos aspectos clave inherentes, entre los que nos encontramos:

- **Comodidad**: Se mejora la experiencia del usuario, ya que evita la necesidad de recordar múltiples contraseñas para diferentes aplicaciones.
- **Centralización**: Se centraliza el control de acceso, lo que facilita la administración y aplicación de políticas de seguridad de manera coherente en todas las aplicaciones y servicios conectados.
- **Riesgos**: Si la implementación de *SSO* no es adecuada, un ataque exitoso contra una aplicación o servicio podría comprometer el acceso a otras aplicaciones vinculadas.

Como se irá indicando más adelante, existen muchas formas diferentes de implementar *Single Sign-On*, ya que hay múltiples estándares y protocolos disponibles en el ámbito de la autentificación y la 'federación' de la identidad.

Cada uno de estos estándares, que serán analizados en detalle, están diseñados para abordar necesidades específicas en distintos contextos organizacionales y de seguridad, por lo que se tendrá que comprender de forma extensa sus características, sus ventajas y casos de uso para hacer una buena elección, garantizando un acceso seguro y eficiente a las aplicaciones y servicios de nuestro entorno digital.

Antes que entrar en detalle, se deben introducir una serie de conceptos que son necesarios conocer:

- **Identidad Federada**: se podría definir como un enfoque que permite a un usuario acceder a múltiples aplicaciones y servicios, utilizando las mismas credenciales de inicio de sesión a través de diferentes dominios y organizaciones. Esto se logra mediante la colaboración entre proveedores de identidad (IdP, Identity Providers) y

proveedores de servicios (SP, Service Providers) que habilitan la autentificación cruzada y el intercambio de información sobre la identidad.

- **Proveedor de Identidad (IdP)**: Un proveedor de identidad es una entidad que autentica y verifica la identidad de los usuarios. Luego emite tokens que certifican la autentificación exitosa del usuario y que utilizan los proveedores de servicios para permitir el acceso a sus recursos.

- **Proveedor de Servicios (SP)**: Un proveedor de servicios es una entidad que ofrece aplicaciones, servicios o recursos en línea que requieren autentificación y autorización. Los proveedores de servicios confían en los proveedores de identidad para autenticar a los usuarios y autorizar su acceso.

- **Tokens de Identidad**: Los tokens de identidad son mensajes o afirmaciones emitidas por el proveedor de identidad después de una autentificación exitosa. Estos tokens contienen información sobre el usuario autenticado y los permisos asociados, y son utilizados por los proveedores de servicios para permitir el acceso sin requerir una autentificación adicional.

- **Protocolos de Identidad y autentificación**: Son conjuntos de reglas y especificaciones que gobiernan cómo se realiza la autentificación y el intercambio de información de identidad entre proveedores de identidad y proveedores de servicios.

Protocolos de identidad y autentificación

Se van a describir, de la forma más clara y sencilla posible, algunos de estos protocolos de cara a la elección que realice la empresa sobre cual de ellos utilizar de cara a la implementación del inicio de sesión único.

JSON Web Token

JSON Web Token o *JWT* es un estándar abierto que define un formato compacto para transmitir información entre dos partes, de manera estructurada y segura. Este tipo de tokens se construye en formato JSON que es codificado para su transmisión, al estar en JSON los datos en destino se obtienen de forma sencilla, una vez descifrados.

Los *JWT* se utilizan comúnmente para la autentificación y la autorización en aplicaciones web y servicios API, y también se usan en sistemas de un solo inicio de sesión o *SSO*. Está diseñado para facilitar un intercambio ligero y seguro de información, adoptando un enfoque basado en el uso de datos en formato JSON, que resultan fácilmente interpretables tanto por máquinas como por los desarrolladores.

Estos tokens se compone de tres segmentos distintos, que están separados por puntos, estableciendo una estructura clara y coherente para su manipulación y verificación:

- **Encabezado (Header)**: Contiene información sobre cómo se ha creado y firmado el *JWT*. Suele constar de dos partes: el tipo de token, que es *JWT*, y el algoritmo de firma utilizado, como HMAC SHA256 o RSA.

- **Carga (Payload)**: Aquí se almacena la información específica que se desea transmitir, como datos de usuario, roles, permisos y un largo etcétera.

- **Firma (Signature)**: La firma se utiliza para verificar que el remitente del token sea quien dice ser y para garantizar que los datos del token no se hayan alterado en el camino. La firma se crea utilizando la información del encabezado y la carga, junto con una clave secreta conocida sólo por el emisor y el receptor del token.

A continuación se mostrará un pequeño ejemplo de generación de un *JWT.*

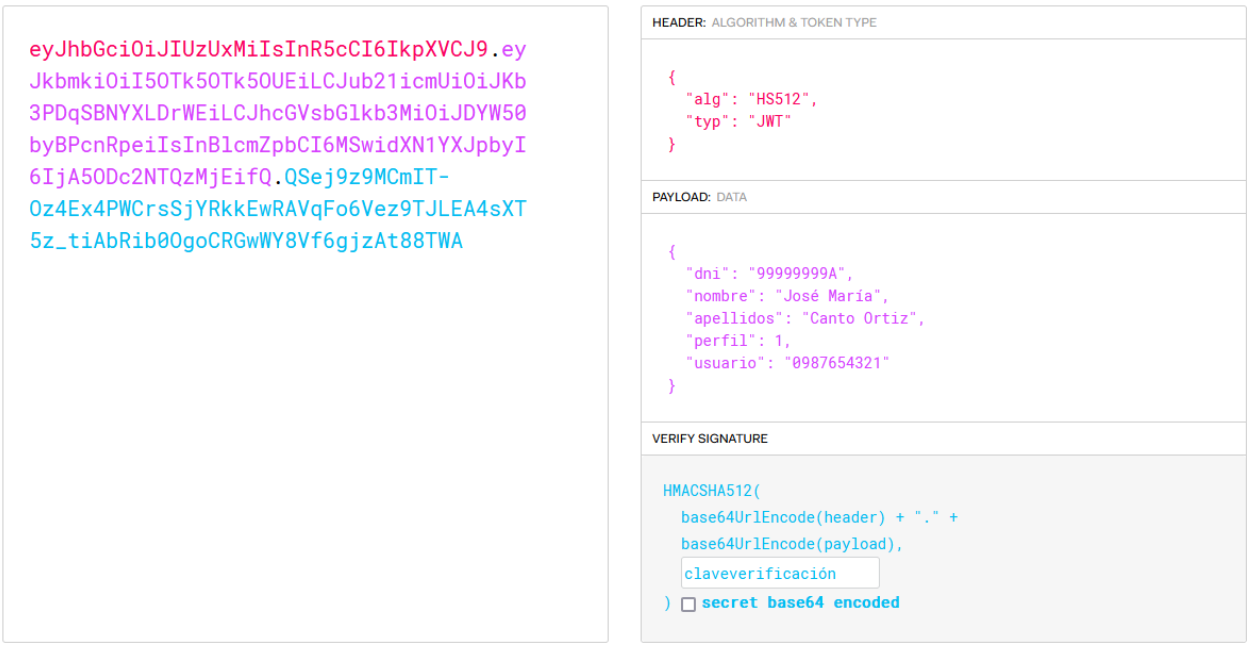

Ilustración 2: JWT codificado y descodificado

A la izquierda se puede observar el token codificado, que es el que se utilizaría para la transmisión y a la derecha se puede observar el token ya decodificado. Para que se puedan diferenciar mejor los segmentos que componen el token, están diferenciados por colores.

La transmisión del token se podría hacer pasando el valor codificado como un elemento de la cabecera, pasar dicho valor como un parámetro en la url de la petición u otras formas que se consideren.

La operación con *JWTs* se inicia con la creación del token por parte del proveedor de autentificación, el proveedor de identidad que nos provee la identidad, una vez que se ha verificado que somos quienes decimos ser. Cuando el cliente recibe el *JWT*, lo incluye en cada solicitud que hace al servidor de recursos, como prueba de autentificación. El servidor de recursos puede verificar la autenticidad del *JWT* y su integridad mediante la firma y luego usar la información contenida en el payload para tomar decisiones en cuanto a la autorización.

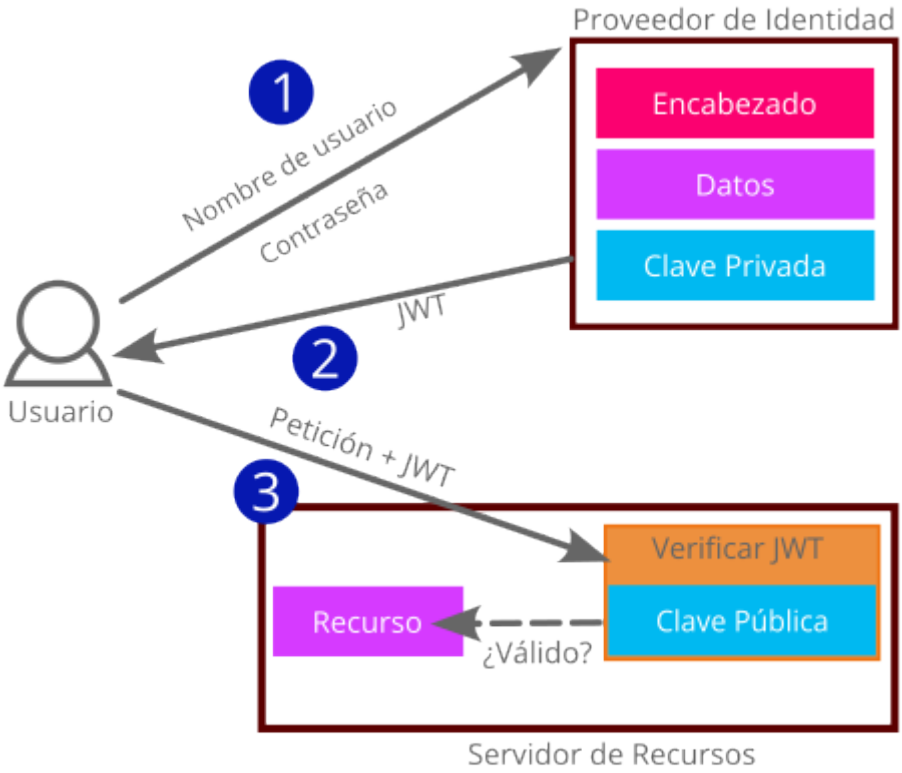

Ilustración 3: Flujo autorización JWT

JSON Web Token (JWT) es un estándar versátil y ampliamente utilizado para transmitir información de manera segura entre dos partes en aplicaciones web y servicios API, lo que lo convierte en una herramienta valiosa para la autentificación y autorización, pero pese a su utilidad, los *JWTs* no son adecuados para almacenar información confidencial o secreta, ya que la parte del payload no está cifrada, es legible por cualquiera que tenga acceso al token, aunque esto tiene solución introduciendo el concepto de *JSON Web Encryption* o *JWE*.

JSON Web Encryption o *JWE* es una extensión del estándar *JSON Web Token*, que proporciona una capa adicional de seguridad al cifrar el contenido del token. Mientras que *JWT* se enfoca en la representación de información en forma de objetos JSON, *JWE* agrega la capacidad de cifrar esos objetos para proteger aún más los datos sensibles durante su transmisión. La diferencia clave entre *JWT* y *JWE* radica en la adición del **cifrado**.

Esta no es la única extensión para *JWT* ya que, aunque no se ha mencionado antes, hemos utilizado *JSON Web Signature* (*JWS*), que es como se llama a la extensión de *JWT* que agrega la capacidad de firmar digitalmente la información contenida en el token, porque el estándar *JWT* solo define el formato para representar información en forma de objetos JSON, sin proporcionar mecanismos de firma ni cifrado.

Como se puede ver en la siguiente ilustración, *JWE* junto con *JWS* ofrecen un enfoque completo y seguro para la representación, firma y cifrado de información en aplicaciones que requieren autentificación, autorización y transmisión segura de datos.

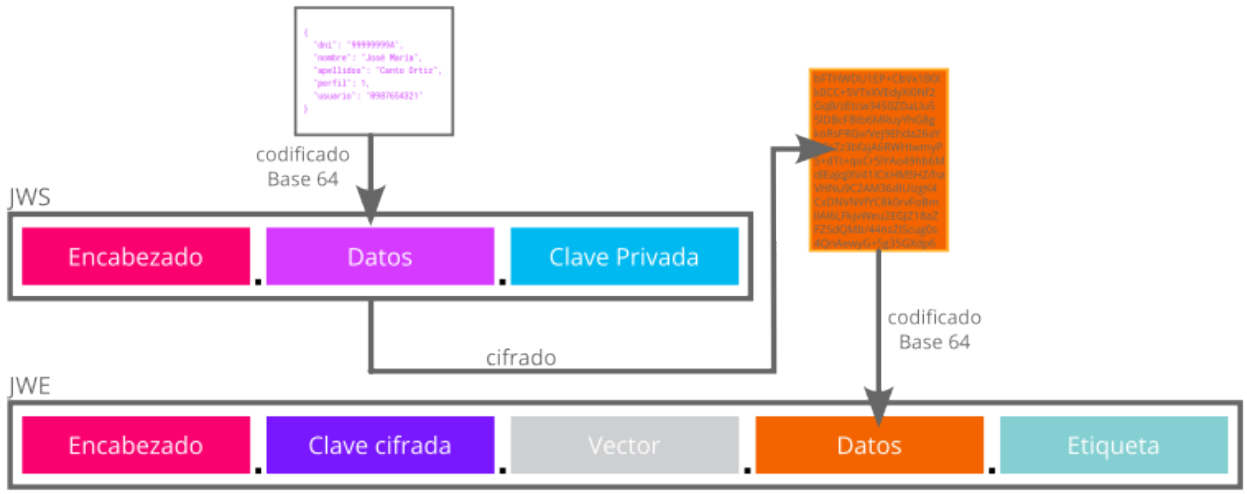

Ilustración 4: JWS y JWE

Open Authorization 2

El protocolo *Open Authorization 2.0*, también conocido como *OAuth 2.0*, definido en la RFC 6749, es un estándar ampliamente adoptado, cuyo objetivo es permitir que una aplicación de cualquier tipo, obtenga acceso limitado y autorizado a los recursos almacenados en otra aplicación, todo ello en representación de un usuario.

OAuth 2.0 facilita un proceso seguro y controlado de intercambio de autorización, en el que el usuario puede otorgar permisos específicos a una aplicación sin tener que compartir sus credenciales de inicio de sesión -nombre de usuario y contraseña-. Esto es crucial para mantener la seguridad y privacidad de las cuentas de los usuarios, ya que evita la necesidad de divulgar contraseñas o información confidencial a terceros.

Una situación típica en la que se aplicaría *OAuth 2.0,* es cuando una aplicación desea acceder a ciertos recursos o datos almacenados en otro servicio o plataforma en línea, como acceder a la lista de contactos de correo electrónico o a la información de redes sociales de un usuario. En lugar de solicitar al usuario sus credenciales de inicio de sesión para estos servicios externos, *OAuth 2.0* establece un proceso de autorización que permite a la aplicación obtener un *'token de acceso'* válido por un tiempo limitado, que actúa como una especie de permiso temporal para acceder a recursos específicos sin revelar la contraseña real del usuario.

OAuth introduce una serie de conceptos que es mejor explicar antes de continuar explicando el flujo del protocolo:

- **Propietario del recurso** *(resource owner)*: elemento capaz de conceder acceso a un recurso protegido. Si el propietario es una persona, se le denomina usuario final
- **Servidor de recursos** *(resource server)*: el servidor que aloja los recursos protegidos,

capaz de aceptar y responder a las solicitudes de recursos protegidos utilizando los tokens de acceso.

- **Cliente**: aplicación que realiza solicitudes de recursos protegidos en nombre del propietario del recurso y con su autorización.
- **Servidor de Autorización** *(authorization server)*: servidor que emite tokens de acceso al cliente tras autenticar correctamente al propietario del recurso y obtener la autorización.

El servidor de autorización puede ser el mismo servidor que el servidor de recursos o uno independiente. Un único servidor de autorización puede emitir tokens de acceso aceptados por varios servidores de recursos.

Flujo del protocolo OAuth 2.0

Para comprender en detalle cómo funciona este proceso de autorización, necesitamos explorar los pasos clave del flujo de información en el protocolo *OAuth 2.0.* A alto nivel, este proceso involucra interacciones entre la aplicación cliente, el propietario de los recursos, el servidor de recursos y el servidor de autorización, que juntos forman el ecosistema que permite a las aplicaciones acceder a los recursos protegidos en nombre del usuario.

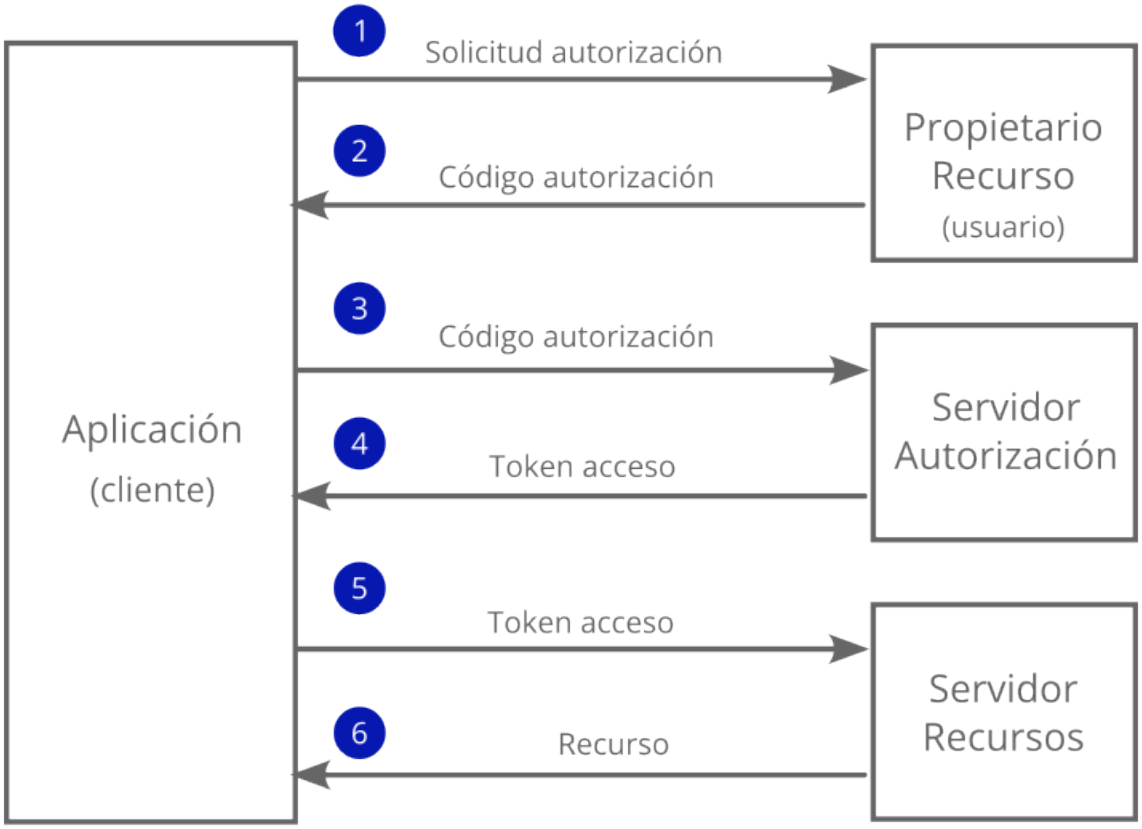

Ilustración 5: Flujo de OAuth 2.0

1. **Solicitud de autorización**: el cliente solicita autorización al propietario del recurso. La solicitud de autorización puede hacerse directamente al propietario del recurso, como se muestra en la imagen, o preferiblemente de forma indirecta a través del servidor de autorización como intermediario.

2. **Código de autorización**: el cliente recibe una concesión o código de autorización, que es una credencial que representa la autorización del propietario del recurso, expresada utilizando uno de los cuatro tipos de concesión definidos en esta especificación o utilizando un tipo de concesión de extensión. El tipo de concesión de autorización depende del método utilizado por el cliente para solicitar la autorización y de los tipos admitidos por el servidor de autorización.

3. **Solicitud del token de acceso**: el cliente solicita un token de acceso autenticándose con el servidor de autorización y presentando la concesión de autorización.

4. **Emisión del token de acceso**: el servidor de autorización autentica al cliente y valida la concesión de autorización y, si es válida, emite un token de acceso, puesto que sin él no se podrá acceder a los recursos protegidos.

5. **Solicitud de recursos**: el cliente solicita el recurso protegido al servidor de recursos y se autentica presentando el token de acceso.

6. **Autentificación del token de acceso**: el servidor de recursos valida el token de acceso y, si es válido y se tienen los permisos necesarios, sirve la solicitud.

Lo descrito en estos puntos y en la ilustración anterior, es la idea general pero el flujo real de este proceso varía según el tipo de concesión de autorización que se esté utilizando, el método específico que determina cómo se solicitan y se otorgan los tokens de acceso para acceder a los recursos.

Cada tipo de concesión de autorización en *OAuth 2.0*, ha sido concebido con el propósito de abordar distintos escenarios, y su elección dependerá de factores como el flujo de interacción entre las partes, el nivel de confidencialidad requerido y las necesidades en cuanto a la seguridad.

Tipos de autorizaciones

Para solicitar un **token de acceso**, el cliente obtiene la autorización del propietario del recurso. La autorización se expresa en forma de concesión o código de autorización, que el cliente utiliza para solicitar el token de acceso. *OAuth* define varios tipos de autorizaciones. También proporciona un mecanismo de extensión para definir tipos de concesión adicionales.

Código de autorización

El tipo de concesión de código de autorización es uno de los más seguros porque los datos sensibles, como el token de acceso y la información del usuario, no se envían a través del navegador. Toda la comunicación que tiene lugar a partir del intercambio del token es de servidor a servidor a través de un canal seguro e invisible para el usuario final. Esta concesión es especialmente apta para aplicaciones en el lado del servidor, donde no se expone el código fuente y se puede mantener la confidencialidad de la **clave secreta de cliente**, denominada de forma habitual como *'Client Secret'*, una clave utilizada por la aplicación cliente para identificarse ante el proveedor de servicios (SP).

Este flujo de autorización se basa en redirecciones, donde se le pregunta al usuario si consiente el acceso, y si acepta se concede a la aplicación cliente un 'código de autorización' que luego intercambia con el servicio *OAuth* para recibir un *'token de acceso',* con el cual realizar llamadas a la API. EL flujo de este tipo de autorización consta de los siguientes pasos:

- **Enlace de Código de Autorización**: En primer lugar, se le proporciona al usuario un enlace de código de autorización que podría ser como este: *'https://oauth-server.com/authorize?response_type=code&client_id=12345&redirect_uri=https://app.com/callback&scope=read&state=5555'*. Donde *response_type* indica que la aplicación está solicitando un código de autorización, *client_id* representa el cómo la API identifica la aplicación, *redirect_uri* revela hacia dónde redirigirá el servicio al navegador después de otorgar un código de autorización, *scope* especifica el nivel de acceso que la aplicación solicita y *state* (opcional) mantiene el contexto entre la solicitud y la respuesta para prevenir ataques de *CSRF*.

- **Autorización del Usuario**: Cuando el usuario hace clic en el enlace, debe iniciar sesión en el servicio para autenticar su identidad, a menos que ya tenga iniciada su sesión. Luego, el servicio le pedirá al usuario que autorice o deniegue el acceso de la aplicación a su cuenta.

- **Recepción del Código de Autorización por la Aplicación**: Si el usuario autoriza a la aplicación, el servicio redirige el navegador a la URI de redirección de la aplicación, que se especificó anteriormente, junto con un código de autorización. La redirección se verá algo así *'https://app.com/callback?code=1a1a1a&state=5555'*.

- **Solicitud del Token de Acceso por la Aplicación**: La aplicación solicita un token de acceso a la API al enviar el código de autorización junto con detalles de autentificación, incluido la clave secreta de cliente, al punto de acceso o *endpoint* de la API para el '/token'. En la petición *POST* se enviaría la siguiente información: *client_id=12345&client_secret=SEC&redirect_uri=https://app.com/callback&grant_type=authorization_code&code=1a1a1a*

- **Recepción del Token de Acceso por la Aplicación**: Si la autorización es válida, la API responderá con un token de acceso (y opcionalmente, un token de actualización) que se enviará a la aplicación. La respuesta completa tendrá un aspecto similar al siguiente json:
  {"access_token":"ACCESS_TOKEN","token_type":"bearer","expires_in":TIMESTAMP,"refresh_token":"REFRESH_TOKEN","scope":"read","info":{...}}

Ahora la aplicación está autorizada y puede utilizar el token para acceder a la cuenta del usuario a través de la API del servicio, con acceso limitado al alcance especificado, hasta que el token expire o se revoque, y en ese caso puede utilizar el token de actualización para solicitar nuevos tokens de acceso.

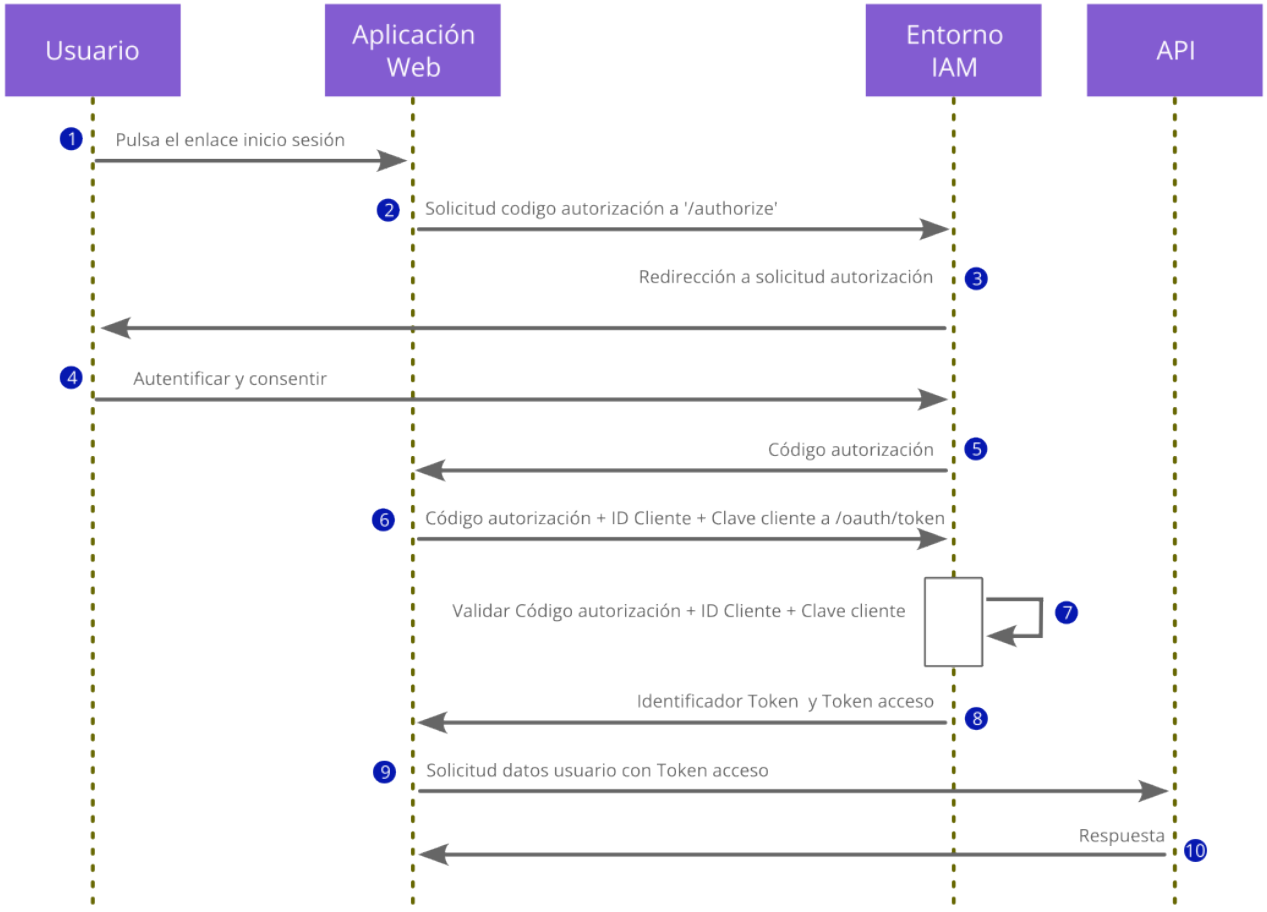

Ilustración 6: Flujo tipo de autorización: código de autorización

Cabe destacar en el caso que si un cliente público opta por emplear este tipo de concesión, existe la posibilidad de que el código de autorización sea interceptado. Hay una solución eficaz para atenuar este riesgo, que es la extensión *Proof Key for Code Exchange* (*PKCE*, conocida como "pixie").

PKCE

Proof Key for Code Exchange, que se podría traducir como clave de intercambio de código, es una extensión de seguridad para el flujo de autorización *OAuth 2.0*, que proporciona una capa adicional de protección contra ataques como la interceptación del código de autorización. *PKCE* fue diseñado principalmente para aplicaciones públicas o nativas que no pueden mantener confidencial la clave secreta de cliente *-client secrect-*, lo que pone en riesgo la seguridad del flujo de autorización.

La forma en que PKCE funciona es mediante la generación de un valor secreto llamado **código de verificación** (*code verifier*) en el cliente, antes de enviar la solicitud de autorización. Luego, se realiza un hash o transformación criptográfica de este código de verificación para crear un **código desafío** (*code challenge*), que se incluye en la solicitud de autorización.

Una vez que el servidor recibe la solicitud, verifica que el código de desafío coincide con el código originalmente generado y proporcionado por el cliente, asegurando que la solicitud es legítima y que no ha sido alterada.

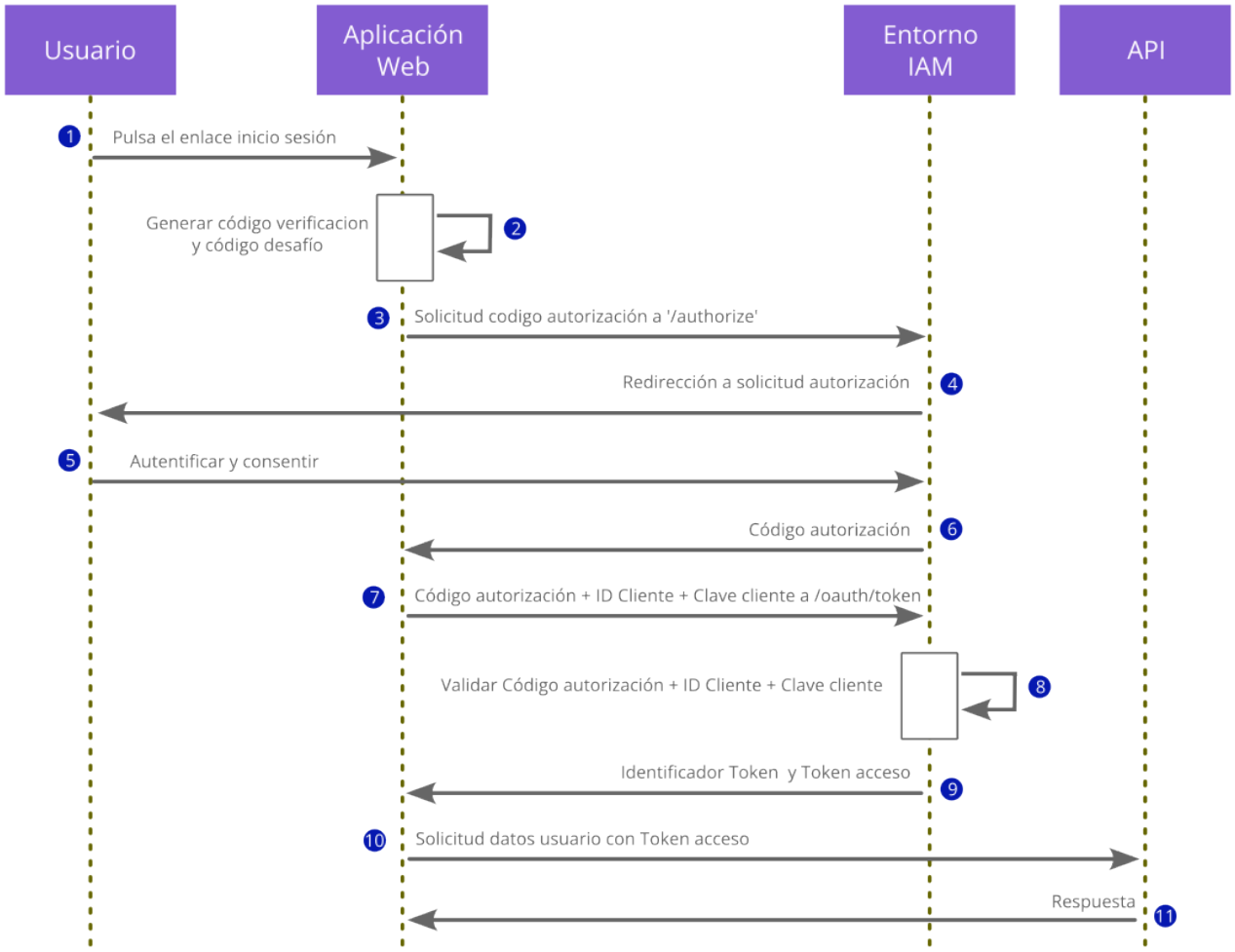

Ilustración 7: Flujo tipo de autorización: PKCE

Credenciales de cliente

El mecanismo de concesión de credenciales de cliente ofrece a una aplicación la capacidad de acceder a su propia **cuenta de servicio**. Esta concesión se utiliza por los clientes para obtener un token de acceso fuera del contexto de un usuario, es decir, acceder a recursos propios en lugar de acceder a los recursos de un usuario. El flujo se desenvuelve de la siguiente manera:

- **Solicitud del Token de Acceso**: La aplicación inicia el proceso solicitando un token de acceso, a través del envío de sus credenciales, es decir, su identificador de cliente *(client ID)* y su clave secreta de cliente *(client secret)*, al servidor de autorización.
- **Obtención del Token de Acceso**: Si las entradas proporcionadas por la aplicación son verificadas con éxito, el servidor de autorización responde suministrando un token de acceso. Con esto, la aplicación obtiene la autorización necesaria para hacer uso de su propia cuenta.

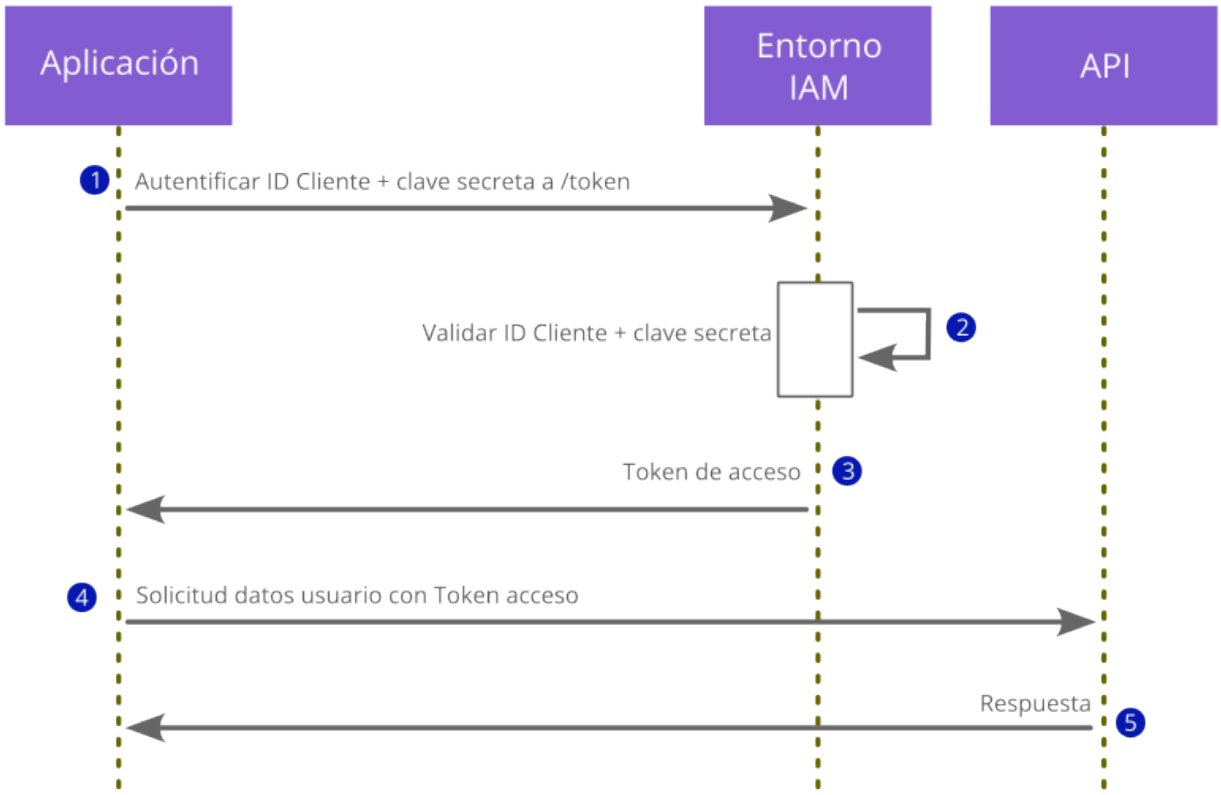

Ilustración 8: Flujo tipo de autorización: credenciales de cliente

Código del dispositivo

El tipo de concesión de código de dispositivo, proporciona un método para que dispositivos que carecen de un navegador web o tienen limitadas capacidades de entrada, puedan obtener un token de acceso y acceder a la cuenta de un usuario. La finalidad de este tipo de concesión es **facilitar** que los usuarios **autoricen** más cómodamente **aplicaciones** en dispositivos de este tipo para acceder a sus cuentas.

Una situación en la que esto podría ser útil sería cuando un usuario desea iniciar sesión en una aplicación de transmisión de vídeo, en un dispositivo que no dispone de un teclado convencional como un televisor inteligente o una consola de videojuegos. El flujo de datos sería el siguiente:

- **Solicitud de Autorización**: El usuario abre una aplicación en su dispositivo sin navegador o con limitaciones de entrada, y envía una solicitud *POST* a un punto de acceso, o *endpoint,* de autorización de dispositivo usando su identificador de cliente.

- **Obtención de Códigos**: Posteriormente se le devuelve un código de dispositivo único *(device_code)* que se utiliza para identificar a este; un código de usuario (*user_code*) que se puede introducir en una máquina más apta para la autentificación, como un portátil o un dispositivo móvil; la URL (*verification_uri*) que el usuario debe visitar para indicar el código de usuario y autentificar su dispositivo; el tiempo de vida (*expires_in*) en segundos para código de dispositivo y código de usuarioy un intervalo de sondeo (*interval*).

- **Consentimiento**: Una vez se ingresa el código de usuario en la URL especificada y se inicia sesión en la cuenta, se le presenta al usuario una pantalla de consentimiento en la que puede autorizar o no el acceso.

- **Sondeo Continuo**: Mientras el usuario visita la URL de verificación e ingresa el código, la aplicación comienza a sondear al servidor de autorización, de acuerdo al intervalo, para obtener un token de acceso, y continúa sondeando hasta que el usuario completa el proceso dando su consentimiento o hasta que el user_code expira.

- **Obtención del Token de Acceso**: Cuando el usuario completa con éxito el proceso, el servidor responde con el token de acceso (y opcionalmente, un token de actualización), que la aplicación del dispositivo puede usar para llamar a una API y acceder a información sobre el usuario.

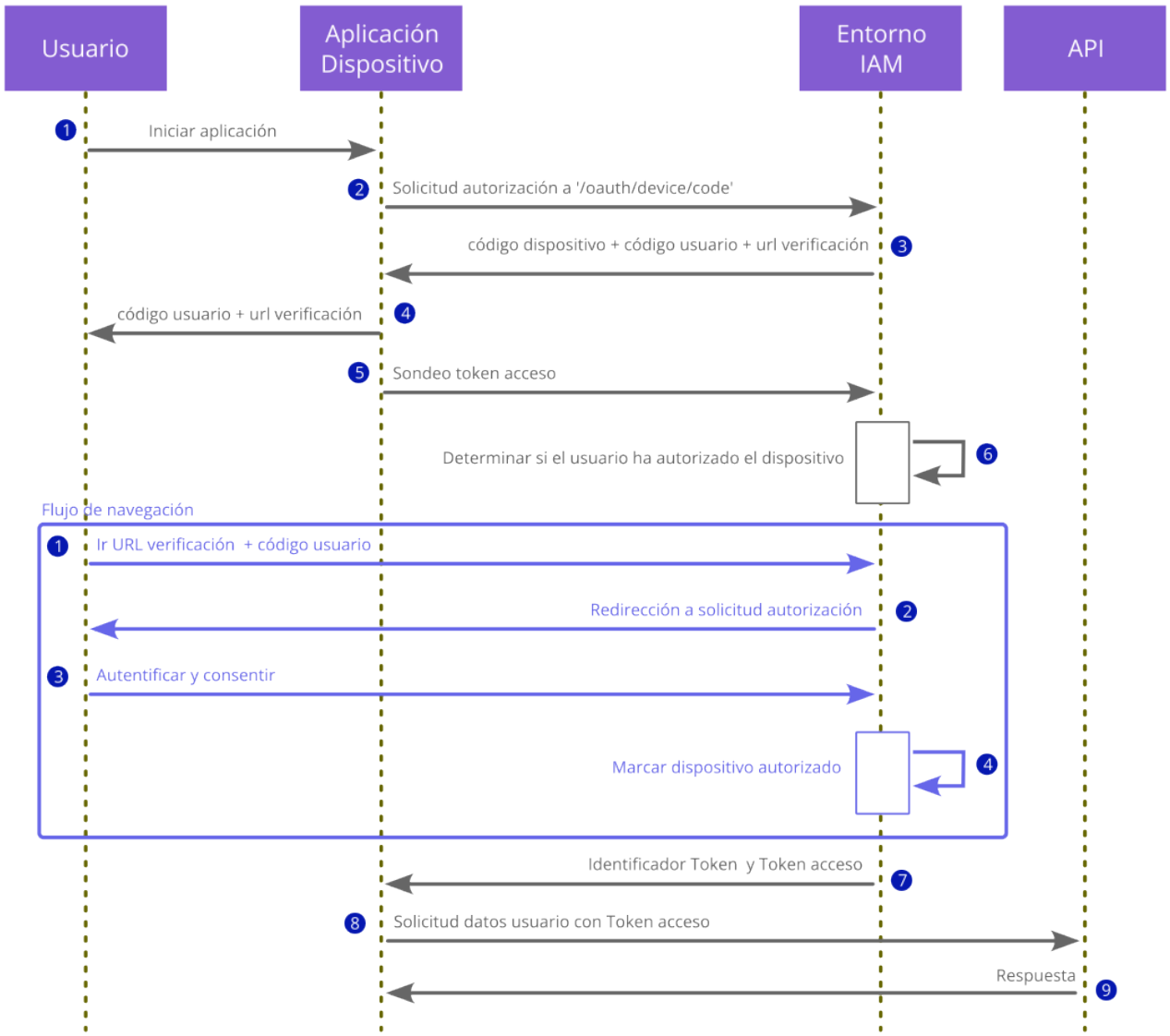

Ilustración 9: Flujo tipo de autorización: credenciales de cliente código de dispositivo

Token de refresco

Como se ha indicado en puntos anteriores, cuando un cliente obtiene un token de acceso y un **token de actualización** a través del flujo de autorización, puede utilizar este último para **solicitar** un **nuevo token** de acceso cuando el primero caduque, ya que los tokens de acceso tienen una fecha de expiración por lo que solo son válidos por un período de tiempo limitado.

En esta concesión el cliente realiza una solicitud *POST* a la URL del punto de acceso o *endpoint '/token'*, estableciendo el parámetro *'grant_type'* al valor *'refresh_token'*. Luego, el servidor de autorización verifica la validez del token de actualización y, si es válido, emite un nuevo token de acceso prolongando la sesión del usuario sin requerir que vuelva a introducir sus credenciales.

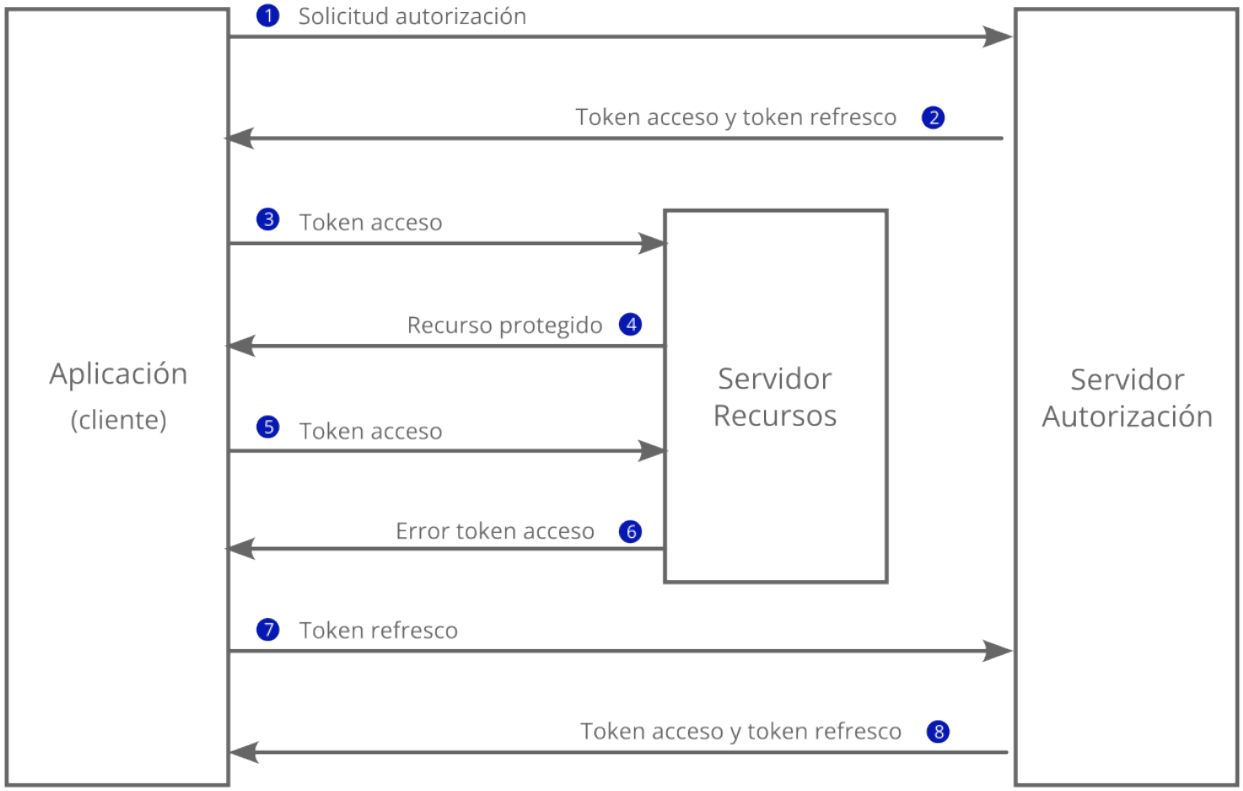

Ilustración 10: Flujo tipo de autorización: token de refresco

Resource owner password credentials

Este tipo que quizás se podría traducirlo como credenciales de propietario del recurso, es adecuado para clientes capaces de obtener las credenciales del propietario de los recursos (nombre de usuario y contraseña, generalmente utilizando un formulario interactivo), aunque también se utiliza para migrar clientes existentes que utilizan esquemas de autentificación directa como autentificación básica de HTTP, convirtiendo las credenciales almacenadas en un token.

Dado que las credenciales se envían al *backend* y pueden almacenarse para uso futuro antes de ser intercambiadas por un token de acceso, es imperativo que la aplicación trate esta información de forma confiable, pero incluso si se cumple esta condición, este flujo **solo** se **recomienda** cuando **no se puedan usar otros** basados en redirecciones.

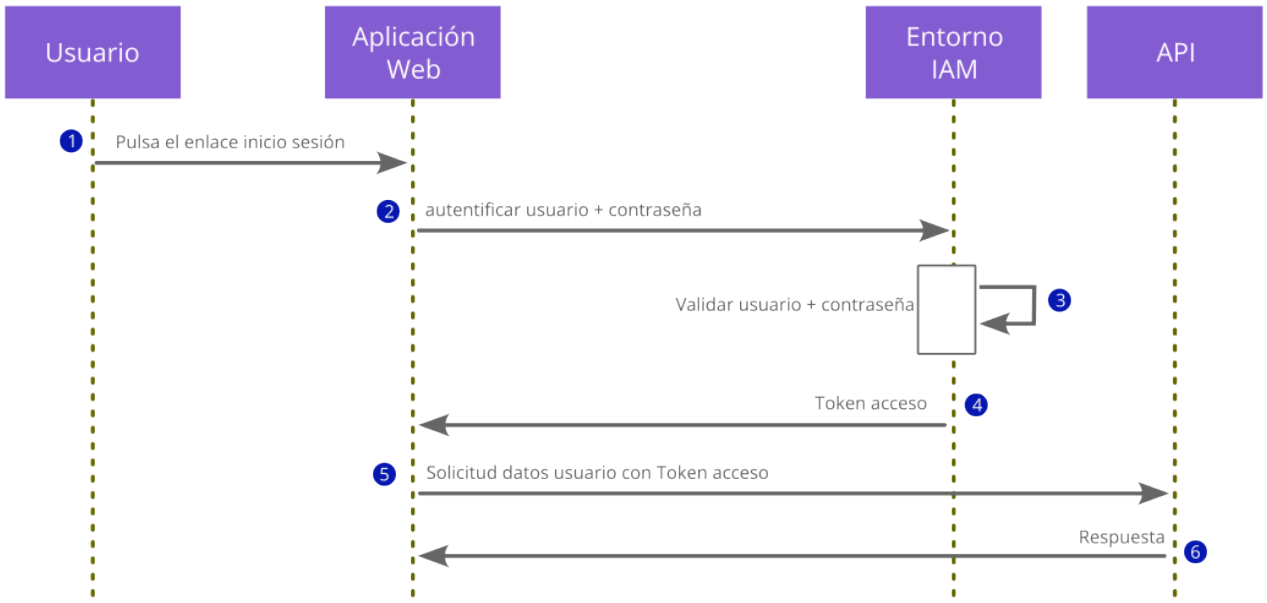

Ilustración 11: Flujo tipo de autorización: credenciales de propietario del recurso

Implícita

El tipo de concesión implícita es una forma heredada pero sencilla de obtener un token de acceso para acceder a los recursos de un usuario. A diferencia del flujo de código de autorización, donde se obtiene un código de autorización primero y luego se intercambia por un token de acceso, en la concesión implícita el cliente recibe directamente el token de acceso después de que el usuario otorga su consentimiento.

La razón por la que las aplicaciones cliente no siempre usan la concesión implícita es porque es **menos segura**. En este flujo, toda la comunicación ocurre a través de **redireccionamientos** del navegador, **sin un canal seguro** en segundo plano como en el flujo de código de autorización, lo que expone el token de acceso y los datos del usuario a posibles ataques. El flujo de información en esta concesión es el siguiente:

- **Solicitud de Autorización**: Es similar al primer paso en el flujo del código de autorización, pero el parámetro *response_type* se define como *'token'*.
- **Inicio de Sesión del Usuario y Consentimiento**: El usuario inicia sesión y decide si otorga o no el permiso solicitado.
- **Concesión del Token de Acceso**: Si el usuario otorga su consentimiento, el servicio *OAuth* redireccionará el navegador del usuario a la *URL* de redireccionamiento especificada en la solicitud de autorización. Sin embargo, en lugar de enviar un código de autorización como parámetro de consulta, enviará el token de acceso y otros datos específicos como parte de la URL.
- **Llamadas a la API**: Una vez que la aplicación cliente haya extraído con éxito el token de acceso, puede utilizarlo para realizar llamadas a la API del servicio *OAuth*.

- **Concesión de Recursos**: El servidor de recursos verificará que el token sea válido y pertenezca a la aplicación cliente actual. Si es así, enviará los recursos solicitados, es decir, los datos del usuario según el alcance asociado al token.

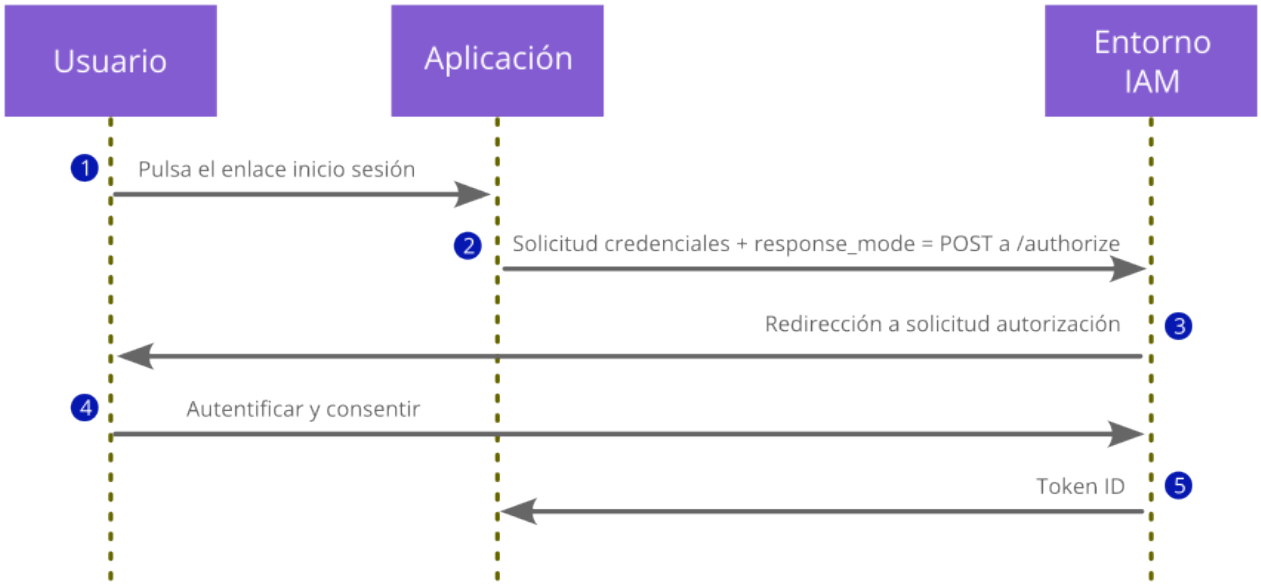

Ilustración 12: Flujo tipo de autorización: implícita

OpenID Connect

OpenID Connect (*OIDC*) es un protocolo de autentificación basado en la familia de especificaciones de *OAuth 2.0,* diseñado exclusivamente para la autorización. Este estándar agrega información de inicio de sesión (autentificación) y perfil (identidad) sobre la persona que ha iniciado sesión, lo que permite a las aplicaciones obtener detalles precisos sobre el usuario autenticado, y proporciona una base sólida para implementar soluciones de inicio de sesión único (*SSO*) y acceso seguro a recursos.

OIDC maneja la autentificación a través de *JSON Web Tokens* (*JWTs*), entregados mediante el protocolo *OAuth 2.0*, para facilitar la entrega segura de información de identidad a las aplicaciones autorizadas, lo que lo convierte en un protocolo clave para acceso seguro a recursos en línea. Al combinar la autorización y autentificación, *OIDC* mejora la seguridad y la experiencia del usuario al permitir un flujo de autentificación seguro y transparente en una variedad de aplicaciones y servicios interconectados.

Este protocolo es comúnmente utilizado por **redes sociales**, ya que permite a los desarrolladores usarlas como proveedores de identidad. Con *OpenID Connect*, una aplicación primero envía una solicitud a un proveedor de identidad, este autentica al usuario y, después de una verificación exitosa, le solicita que otorgue, a la aplicación que inició la solicitud, acceso a los datos. Una vez que el usuario acepta compartir los datos, el proveedor de identidad genera un token llamado *id_token*, que contiene información de la identidad del usuario, y lo devuelve a la aplicación.

En resumen, *OpenID Connect* es una capa de identidad construida sobre *OAuth 2.0* que proporciona autentificación federada para aplicaciones web, ofreciendo una forma segura y sencilla de integrar la autentificación de terceros en aplicaciones, lo que facilita el proceso de inicio de sesión para los usuarios y mejora la seguridad general de los sistemas.

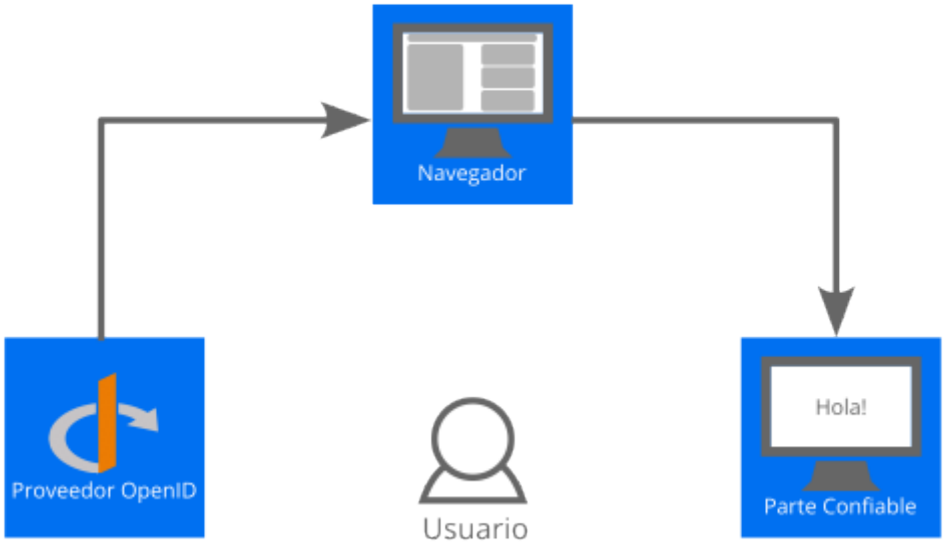

Ilustración 13: Flujo protocolo OpenID Connect

OIDC, a diferencia de *OAuth 2.0*, permite a las aplicaciones web autenticar a los usuarios de manera segura y obtener información sobre su identidad, además de acceder a recursos protegidos en su nombre y permitir a cualquier entidad desempeñar el rol de proveedor de identidad.

El protocolo sigue una serie de pasos que involucran la autentificación del usuario por parte de un proveedor *OpenID*, la generación de tokens de identidad y acceso, y la comunicación entre el cliente y el proveedor para intercambiar información relevante. El proceso general de OIDC es el siguiente:

- **Acceso vía Web**: El proceso comienza cuando un usuario ingresa a una página web o aplicación a través de su navegador.
- **Inicio de Sesión**: El usuario decide iniciar sesión en la aplicación y proporciona sus credenciales de acceso (nombre de usuario y contraseña).
- **Solicitud al Proveedor OpenID**: La aplicación envía una solicitud al proveedor *OpenID* para autenticar al usuario y obtener información de su identidad.
- **autentificación y Autorización**: El proveedor *OpenID* verifica las credenciales y, si son válidas, solicita y obtiene la autorización del usuario para compartir su información de identidad con el cliente.

- **Emisión de Tokens**: Una vez que el usuario ha sido autenticado y ha otorgado su consentimiento, el proveedor *OpenID* genera un token de identidad (*id_token)*, que contiene información sobre la identidad del usuario, y posiblemente un token de acceso (*access_token*), que permitirá al cliente acceder a recursos protegidos en nombre del usuario.

- **Uso de Tokens**: El cliente puede decidir enviar el token de acceso al dispositivo del usuario, útil si necesita acceder a servicios o recursos adicionales, como APIs.

- **Información del Usuario**: Si el cliente necesita información adicional sobre el usuario, puede utilizar un punto de acceso específico (*UserInfo*) para solicitar atributos como el nombre, dirección de correo electrónico, etcétera.

Flujos del protocolo OpenID Connect

Al igual que *OAuth 2.0*, *OIDC* ofrece varios tipos de concesiones de autorización, elementos fundamentales que definen cómo los usuarios y las aplicaciones interactúan durante el proceso de autentificación, para permitir la obtención de tokens en nombre de un usuario autenticado.

En este estándar se pueden seguir tres caminos: flujo de código de autorización, flujo implícito y flujo híbrido; dos de los cuales ya se han explicado con anterioridad, aunque se introducen ciertas variaciones, por la diferencia principal de los protocolos, y es que a pesar de que los flujos a alto nivel son similares, un flujo de *OpenID Connect* resulta en un token de identificación además de cualquier token de acceso o de actualización.

Flujo de código de autorización

En el flujo de código de autorización, muy parecido al de OAuth 2.0, los pasos para obtener un token de acceso y un token id, son los siguientes:

- **Solicitud de autentificación**: El cliente prepara una solicitud de autentificación *OpenID* (*https://server.example.com/authorize? response_type=code&scope=open_id_profile_email&client_id=12345&state=5555&redirect_uri=https://client.example.org /cb*), que esencialmente es una solicitud de autorización de *OAuth 2.0* para acceder a la identidad del usuario, la cual contiene los parámetros deseados, indicando el valor *'openid'* en el parámetro de alcance (*scope*), ya que es el valor necesario si se quiere iniciar el flujo de autorización openid.

- **Envío de Solicitud**: El cliente envía la solicitud al servidor de autorización, que valida todos los parámetros de *OAuth 2.0* según la especificación, y valida que exista un parámetro de alcance con al menos el valor *'openid'*.

- **Inicio de Sesión del Usuario**: Si la solicitud es válida, el servidor de autorización intenta autenticar al usuario final, aunque existen casos en donde el servidor de

autorización no debe intentarlo. Si el usuario aún no está autenticado o si está autenticado, pero se envía el parámetro opcional *'prompt'*, que indica una nueva autentificación y consentimiento, con el valor *'login'* en la solicitud, el servidor de autorización debe seguir el proceso. Sin embargo, si se envía *'prompt'* con el valor *'none'* se debe generar un error.

- **Consentimiento**: Después de autenticar al usuario final, el servidor de autorización, a través de un diálogo interactivo, obtiene su decisión en cuanto a otorgar el acceso.
- **Redirección al Cliente**: Una vez se permite el acceso, el servidor de autorización redirige al usuario de vuelta al cliente con un código de autorización *(code)*.
- **Envío del Código de Autorización**: El cliente enviará ese código de autorización al punto de acceso del token mediante una petición *POST (grant_type=authorization_code&code=ABCDE&redirect_uri=https://client.example.org/cb)*, y solicitará el token id, el token de acceso y, opcionalmente, el token de actualización.
- **Validación de la Solicitud**: El servidor de autorización valida la solicitud de token y envía una respuesta al cliente con dicha información *({'access_token': '1111', 'token_type': 'Bearer', 'refresh_toke n': '2222', 'expires_in': 3600, 'id_token': '3333'})*.
- **Identidad del Usuario**: Finalmente, el cliente verifica el *id_token* y obtiene la identidad del usuario.

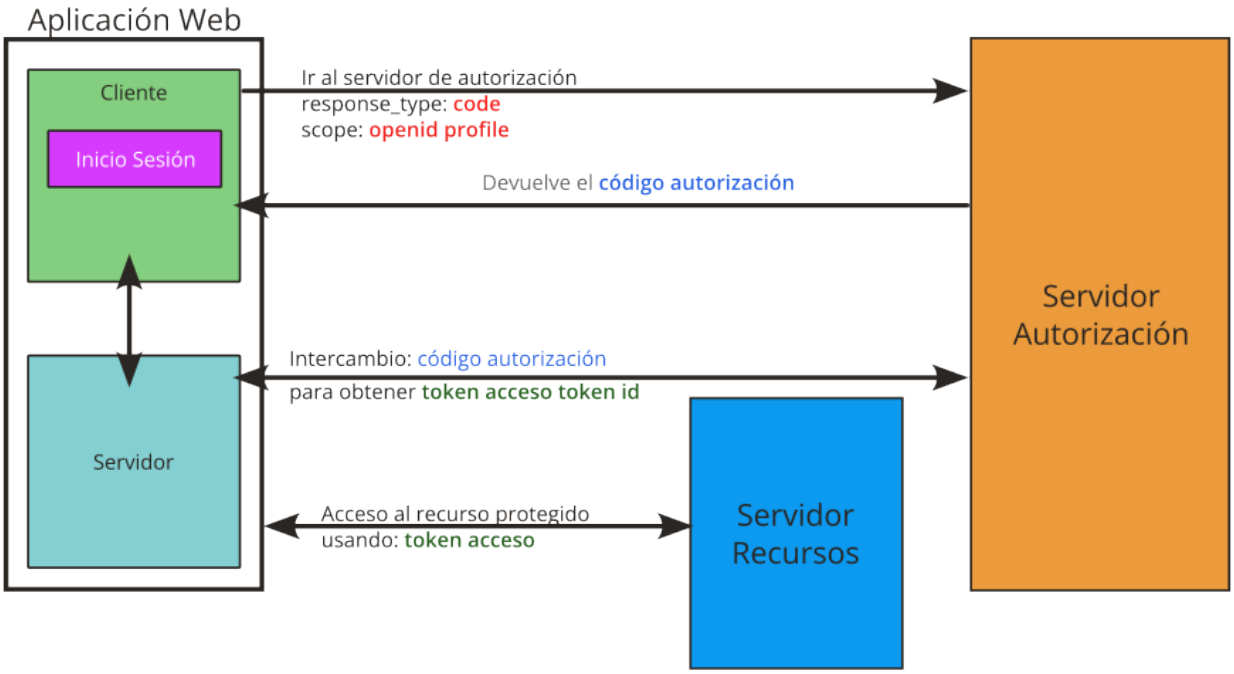

Ilustración 14: Flujo de código de autorización en OpenID Connect

Flujo implícito

En el flujo implícito, los tokens son emitidos en el punto de acceso, o *endpoint*, de autorización, no en el de token. Como se pudo ver al explicar este mismo flujo en *OAuth 2.0*, no se genera un código de autorización, solo se generan tokens, que se obtienen de la siguiente forma:

- **Solicitud de autentificación**: El cliente prepara una solicitud de autentificación *OpenID* (*https://server.example.com/authorize?response_type=id_token token&client_id=12*345&redirect_uri=https://client.example.org/cb&scope=openid profile&state=5555&*nonce=hjhjhj*), y la envía al servidor de autorización.

- **Inicio de Sesión del Usuario**: El servidor de autorización autentica al usuario final, y obtiene el consentimiento de acceso a recursos por parte del usuario, redirigiéndolo de vuelta al cliente con un token de identidad y, si se solicita, un token de acceso.

- **Identidad del Usuario**: El cliente valida el *id_token* y obtiene la identidad del usuario.

Como se puede apreciar este flujo destaca por su sencillez, aunque se incluyen algunos parámetros y valores extra en la primera petición, como *'nonce'*, utilizado para asociar una sesión de cliente con un token id y mitigar ataques de repetición. Además, se añaden los valores *id_token* y *token* en *response_type*, para indicar que se solicita un token de identidad y un token de acceso.

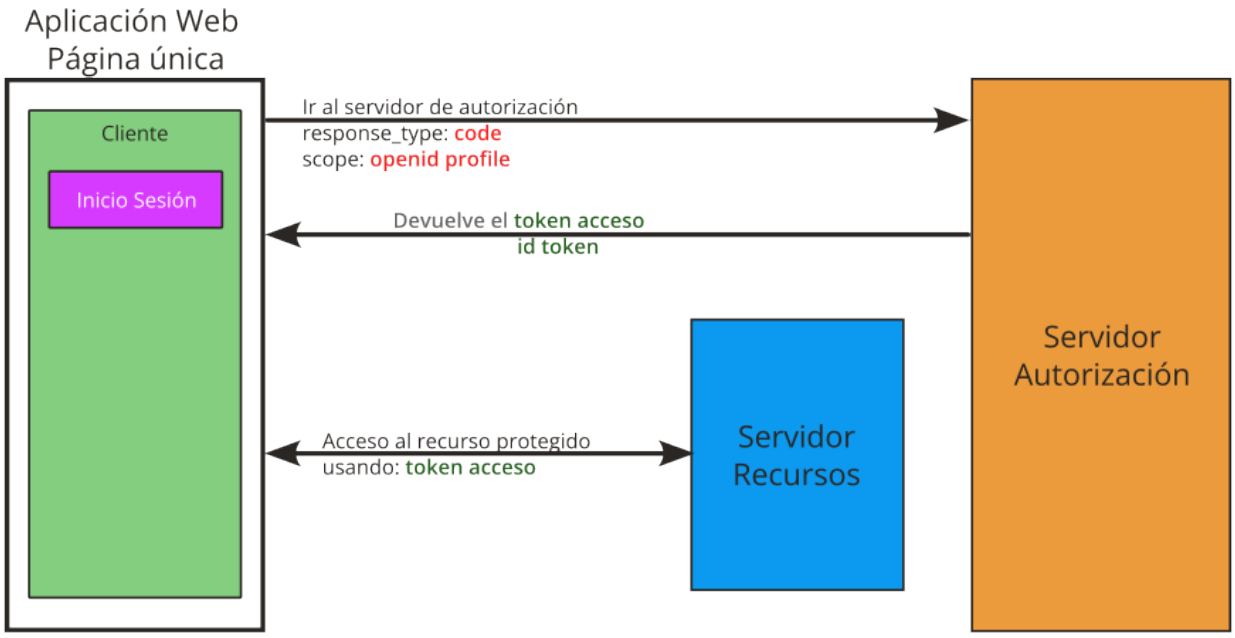

Ilustración 15: Flujo implícito en OpenID Connect

Flujo híbrido

El flujo híbrido combina elementos de los dos anteriores, al generar un código en el punto de acceso de autorización y obtener tanto el *token id* como el *token de acceso* según sea necesario, útil cuando se requiere que las aplicaciones reciban tokens separados para el *front-end* y el *back-end*. De esta forma, se puede recibir un token para el acceso inmediato a información relacionada con el usuario, mientras se mantiene el acceso de forma segura a otros tokens. El flujo de operación sería el siguiente:

- **Inicio de Sesión del Usuario**: El usuario inicia el proceso haciendo clic en 'Iniciar sesión' dentro de la aplicación.
- **Redirección al Servidor**: La aplicación redirige al usuario hacia el servidor de autorización, usando el punto de acceso '*/authorize*'. En esta redirección se incluyen los parámetros 'response_type', para indicar el tipo de token solicitado, y 'response_mode' establecido al valor 'form_post', para la seguridad en la respuesta.
- **autentificación del Usuario**: El usuario se autentica utilizando una de las opciones de inicio de sesión configuradas y da su consentimiento en cuanto al acceso a recursos.
- **Redirección al Cliente**: Después de autenticarse, el servidor de autorización redirige al usuario de regreso a la aplicación. En este punto, se proporciona un código de autorización, válido para un solo uso y, un token id, un token de acceso o ambos.
- **Envío de Solicitud**: La aplicación envía el código de autorización, su id de cliente y el material de autentificación, la clave secreta de cliente o una clave privada *JWT*, al punto de acceso *'/token'* del servidor.
- **Verificación de la Aplicación**: El servidor de autorización verifica la validez del código de autorización, el id de cliente y la identidad de la aplicación.
- **Segundo Token ID y Token de Acceso**: Si la verificación es exitosa, el servidor de autorización responde proporcionando un segundo token id y un token de acceso, y en algunos casos, un token de actualización.
- **Llamadas a la API**: La aplicación ahora sí puede utilizar el token de acceso para realizar llamadas a una API y acceder a información del usuario.

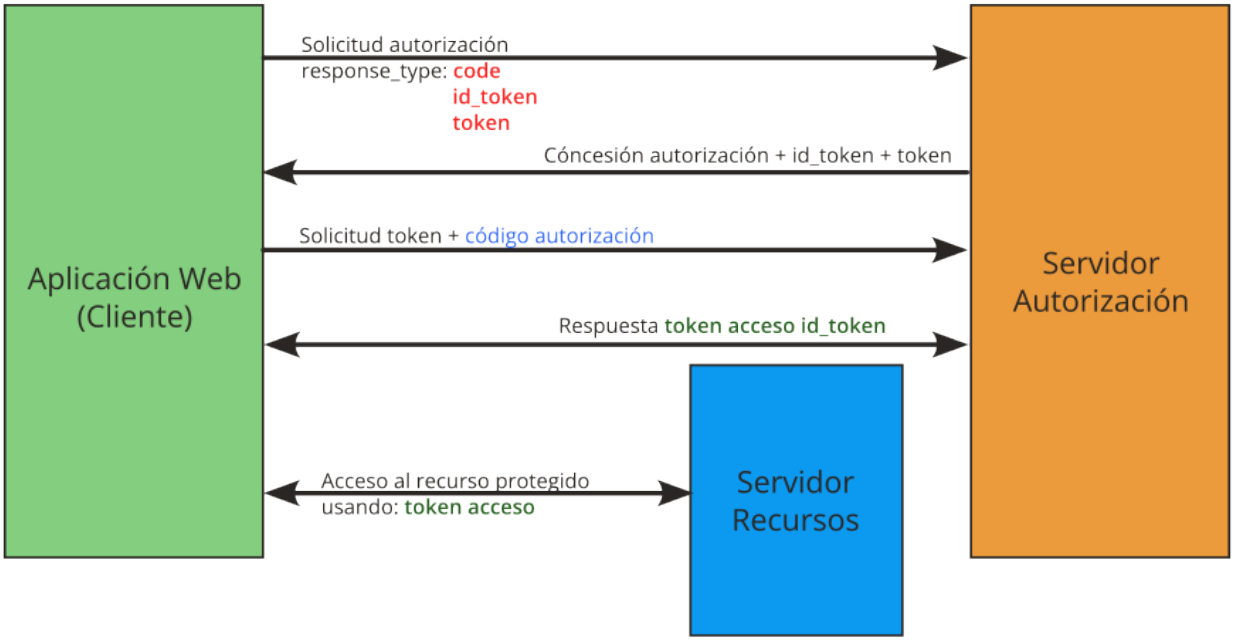

Ilustración 16: Flujo híbrido en OpenID Connect

Fast Identity Online 2

La *FIDO Alliance*, una organización sin ánimo de lucro que reúne a empresas líderes en tecnología, proveedores de servicios, instituciones financieras y otros actores de la industria con el objetivo de desarrollar y promover estándares abiertos para mejorar la autentificación y la seguridad en línea, ha publicado tres conjuntos de especificaciones que juntas se conocen como *Fast Identity Online 2* o *FIDO2*:

- ***FIDO Universal Second Factor* (*FIDO U2F*)**: permite agregar un segundo factor a la autentificación por medio del uso de dispositivos físicos como llaves de seguridad USB.
- ***FIDO Universal Authentication Framework* (*FIDO UAF*)**: permite una autentificación más avanzada y conveniente al utilizar métodos biométricos -como huellas dactilares o reconocimiento facial-, mejorando la seguridad y la experiencia del usuario al eliminar la necesidad de recordar contraseñas. Con *FIDO UAF* también se pueden combinar múltiples mecanismos de autentificación, como huellas dactilares más PIN.
- ***Client to Authenticator Protocols* (*CTAP*)**: Estos protocolos, que complementan la especificación de *autentificación Web (WebAuthn)* del W3C, permiten la comunicación segura entre el cliente (navegador web) y los autentificadores (llaves de seguridad).

FIDO2 es un estándar que utiliza la criptografía de clave pública para garantizar un sistema de autentificación seguro y conveniente, eliminando la necesidad de contraseñas tradicionales cuyo uso introduce amenazas como *phishing* y ataques de fuerza bruta. Con la implementación de este estándar se opta por emplear una clave privada y una clave pública para validar la identidad de cada usuario.

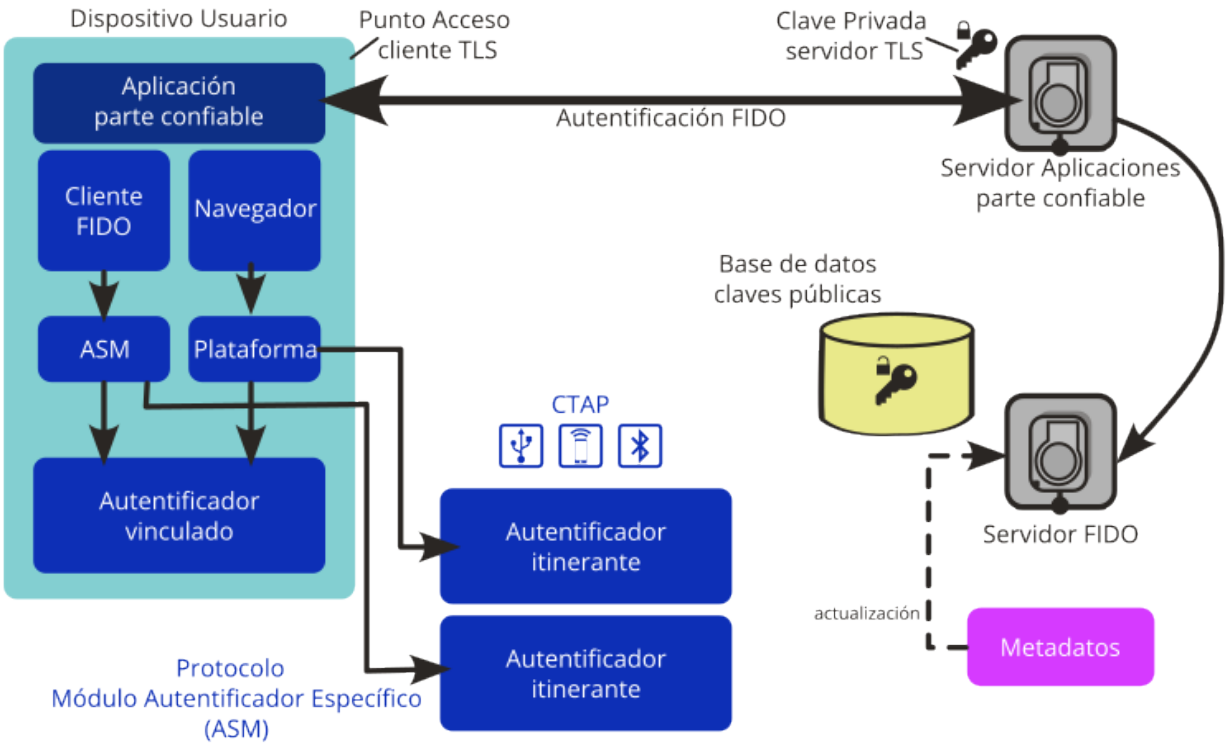

Ilustración 17: Esquema de Fast Identity Online 2

Antes a utilizar la autentificación *Fast Identity Online 2*, se debe completar el registro en servicios compatibles y ya hay grandes plataformas como Apple, Google y Microsoft que respaldan FIDO, aunque no son su única fuerza impulsora. La *FIDO Alliance* colabora junto a cientos de empresas en todo el mundo para hacer que la autentificación más simple y sólida, sea una realidad. Como primer paso, debemos configurar su uso siguiendo estas indicaciones:

- **Registro**: completar el correspondiente formulario de registro en el servicio que permita esta opción y seleccionar un autentificador.
- **Generación de Claves**: una vez se produzca el registro, el servicio generará un par de claves de autentificación -clave privada y clave pública-.
- **Almacenamiento en Dispositivo**: El autentificador enviará la clave pública al servicio, mientras que la clave privada que contiene información sensible permanecerá en el dispositivo.
- **Comunicación**: Una vez habilitada esta opción de comunicación segura, las claves configuradas se almacenan permanentemente para autentificaciones futuras.

Al completar la configuración, la próxima vez que se inicie sesión en uno de los servicios que admiten el estándar *FIDO2*, simplemente se le proporciona el nombre de usuario y correo electrónico al servicio, que presentará un desafío criptográfico y usando el

autentificador *FIDO2* se firmará para que el servidor del servicio lo verifique y otorgue el acceso.

Es importante recordar que, en este proceso de inicio de sesión seguro en la web, no se intercambian secretos con los servidores. La pieza crucial de información, que es la clave de seguridad *FIDO2*, siempre permanece en el dispositivo.

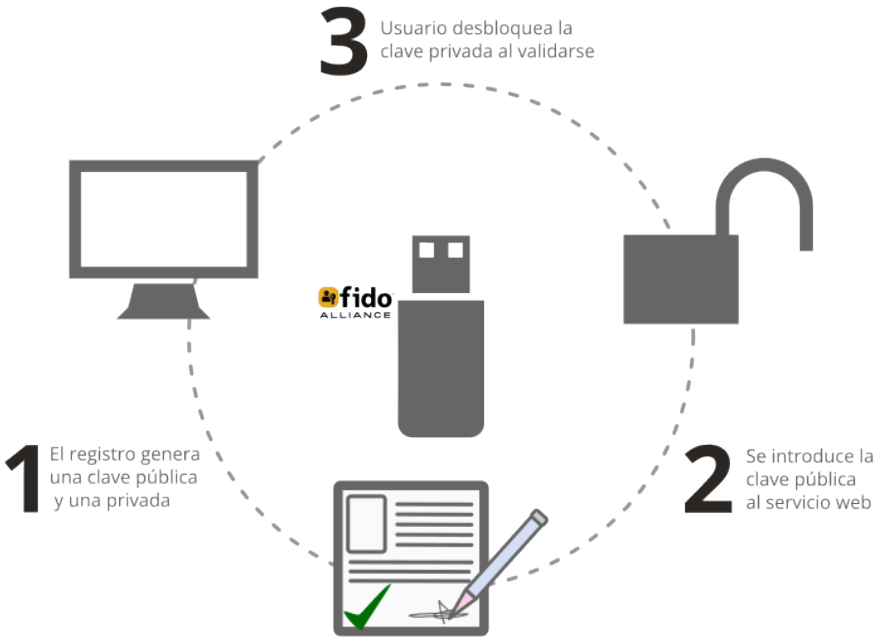

Ilustración 18: Proceso de registro mediante FIDO2

Ataques a protocolos de identidad y autentificación

Siempre pueden surgir nuevas formas de atacar a estos protocolos, aquí se tratarán de explicar las más comunes.

JSON Web Token

A continuación se incluyen algunos de los ataques que podrían afectar a *JWT* con una mala implementación del protocolo.

Fallo al verificar la firma

Este ataque se refiere a una incorrecta implementación de *JWT*, donde los desarrolladores confunden los métodos de decodificación y verificación proporcionados por muchas bibliotecas.

En este contexto, las bibliotecas a menudo ofrecen dos métodos diferentes para manejar los tokens:

- **decode()**: Este método solo decodifica el token de su codificación *base64url* sin verificar la firma. En otras palabras, convierte el token en una estructura legible, pero no verifica si el token ha sido alterado o si la firma es válida.

- **verify()**: Este método decodifica el token y verifica su firma. Verifica la autenticidad del token asegurando que no haya sido manipulado y que la firma se corresponda con la clave secreta del emisor.

El problema surge cuando los desarrolladores confunden o mezclan incorrectamente estos métodos. Si accidentalmente utilizan solo el método de decodificación (*decode()*) en lugar del método de verificación (*verify()*), el token no se autenticará y la aplicación aceptará cualquier token, incluso si su firma es incorrecta. Otra posibilidad también es que los desarrolladores deshabiliten la verificación de firmas para realizar pruebas y olviden volver a habilitarla en producción, permitiendo el acceso no autorizado a cuentas o la escalada de privilegios.

Para prevenir este tipo de ataque, es fundamental que utilicemos correctamente los métodos de decodificación y verificación proporcionados por las bibliotecas para implementar JWT, y que mantengamos la seguridad en la validación de los tokens en todas las etapas de desarrollo y producción.

Permitir que no se indique algoritmo

En el contexto de *JWT*, se utilizan algoritmos para generar la firma digital que asegura la autenticidad e integridad. Sin embargo, uno de los algoritmos aceptados por el estándar es el conocido como *'None'*, que indica que el token no está firmado en absoluto. Esto significa que el contenido no está protegido por una firma digital.

La vulnerabilidad radica en la aceptación de varios tipos de algoritmos para generar una firma en el token, mejor dicho, la posibilidad de cambiar un algoritmo de firma legítimo a *'None'*, lo que permite modificar el contenido de este sin que se detecte una alteración.

Suplantación de JWKS

El ataque de suplantación de *JWKS*, o *JWKS Spoofing*, es una técnica utilizada para manipular el proceso de verificación de tokens *JWT*, al aprovechar la propiedad o parámetro *'jku'* (JSON Web Key URL) de la cabecera del token. Cuando un token *JWT* incluye este parámetro en su cabecera, significa que el servidor de autentificación utiliza una URL específica para cargar las claves públicas necesarias en la verificación.

En la ejecución de este ataque, se manipula el *JWT* para que el valor de *'jku'* apunte a una URL controlada, ya que de esta forma el atacante puede observar las interacciones HTTP y eso le proporciona información sobre cómo está tratando de cargar las claves públicas el servidor. Posteriormente, habiendo generado un nuevo par de claves, inyecta la URL controlada en el token y crea un conjunto de claves *JWKS (JSON Web Key Set)*, que contienen la

nueva clave pública y el token firmado con la nueva clave privada, presentándose como si fuera el conjunto legítimo de claves que utiliza el servidor.

Cuando el servidor de autentificación recibe el token manipulado y trata de cargar las claves públicas desde la URL controlada, obtiene las claves falsas generadas por el atacante, lo que puede llevar a que el servidor verifique el token como válido incluso cuando no lo es, ya que las claves falsas coinciden con la firma en el token manipulado.

En resumen, el ataque de suplantación de *JWKS* se basa en manipular el parámetro '*jku*' de un token *JWT* para engañar al servidor de autentificación y lograr que use claves públicas falsas generadas por el atacante, lo que lleva a la validación errónea de tokens.

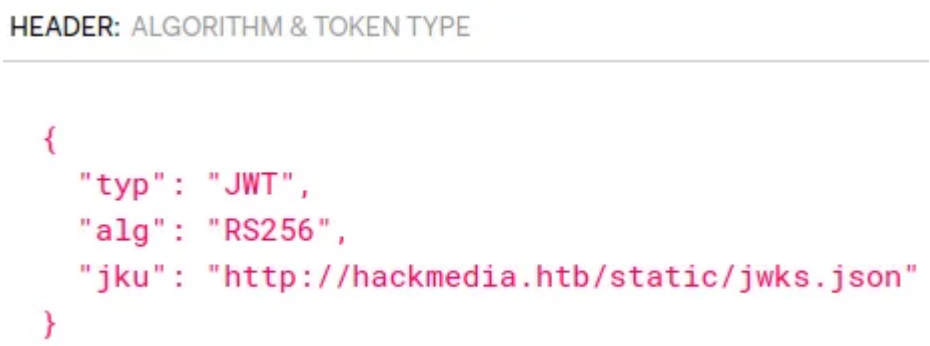

Ilustración 19: manipulación parámetro jku

Inyecciones en el parámetro KID

El parámetro *'kid'* se utiliza comúnmente para identificar una clave específica en una base de datos o sistema de archivos, que luego se emplea para verificar la firma del token. Si este parámetro se logra manipular o inyectar con datos maliciosos, puede abrir la puerta a ejecutar otra serie de ataques en la aplicación.

Un atacante podría manipular este parámetro de manera que la aplicación utilice una clave comprometida o inexistente para verificar la firma del token, dando como resultado la aceptación de un token no válido y la posibilidad de ejecutar ataques más avanzados, como ejecución remota de código (*RCE*), inyección de SQL (*SQLi*) o inclusión de archivos locales (*LFI*).

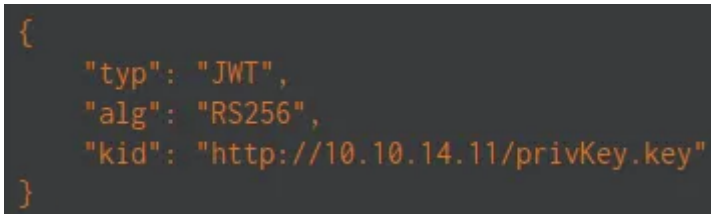

Ilustración 20: Manipulación del parámetro "kid"

Es importante tener en cuenta que prevenir esta vulnerabilidad implica implementar medidas de seguridad sólidas para validar y proteger adecuadamente el parámetro 'kid' y las

claves asociadas en la aplicación. Esto podría incluir la validación estricta del parámetro, el uso de mecanismos de autentificación sólidos para acceder a las claves y la implementación de prácticas seguras de almacenamiento y recuperación de las mismas.

Open Authorization 2

La adopción generalizada de este estándar, si bien ha impulsado significativamente la eficiencia y la seguridad en la gestión de la identidad y el acceso, también ha generado un entorno propicio para la exploración de posibles vectores de ataque, sobre todo, como hemos visto, por la diversidad de tipos de concesiones (*grant types*), la necesidad de coordinación entre distintas entidades durante el flujo de autorización y otras variables inherentes a su implementación.

En este contexto, el crecimiento del ecosistema de aplicaciones y servicios compatibles con OAuth 2.0 amplía la superficie de ataque, ya que cada nueva integración puede introducir configuraciones defectuosas o confusiones en el proceso de implementación.

PRE-ACCOUNT TAKEOVER

Uno de los problemas más comunes que suele presentarse es una vulnerabilidad que permite tomar el control de cuentas de usuario, aprovechándose de la falta de verificación de correos electrónicos durante la creación de cuentas. Los atacantes buscan vincular cuentas a direcciones de correo electrónico de víctimas para obtener acceso no autorizado a la información de esos usuarios.

Imaginemos que estamos utilizando una aplicación que nos permite iniciar sesión con nuestra cuenta de Google, pero también podemos crear una cuenta dentro de la misma aplicación usando un correo y contraseña. Existen dos formas en las que un atacante puede aprovechar esta situación:

- **Pre-Registro**: Si la aplicación no verifica si un correo ya está registrado antes de que alguien cree una cuenta, un atacante podría crear una cuenta falsa con el correo del usuario antes de que este se registre. Luego, cuando el usuario intente usar su cuenta de Google para entrar, la aplicación encuentra el correo registrado, cree que es el del usuario y vincula su cuenta de Google con la cuenta falsa que creó el atacante, pudiendo así acceder este a información sensible.

- **Modificación de Correo Electrónico**: Si la aplicación no verifica bien los correos electrónicos, un atacante podría registrarse usando una dirección falsa y luego cambiarla por la del usuario. De nuevo, la aplicación podría creer que el correo es el suyo y vincularlo con la cuenta falsa.

IMPROPER VALIDATION OF REDIRECT_URI

Una vulnerabilidad recurrente en los servidores de autorización *OAuth 2.0* es la insuficiente validación de las *URIs* de redirección, que las aplicaciones cliente almacenan como ubicaciones confiables para la devolución de llamadas. Durante las solicitudes de autorización, el cliente suministra una URL de redirección válida como parte de los parámetros de la solicitud, especificando hacia dónde se redirigirá al usuario junto con su código de autorización o token de acceso, dependiendo del flujo en uso.

La vulnerabilidad emerge cuando el servidor de autorización no verifica adecuadamente si la URL de redirección, proporcionada por el cliente, es una de las URLs permitidas y confiables, una situación que puede llevar a la exposición del token. Tomemos como ejemplo el flujo implícito de *OAuth 2.0*: si el servidor de autorización no valida correctamente el dominio en la URL de redirección, un atacante podría dirigir un token de acceso a su propio servidor. Además, en ciertas ocasiones, los servidores de autorización únicamente validan el dominio de la URL, omitiendo considerar la ruta completa, lo que permite a un atacante dirigir al usuario a un punto de acceso de redirección abierto (*Open Redirect*) en el dominio confiable de la aplicación cliente, que a su vez puede ser explotado para llevar a cabo un ataque de redirección.

Los atacantes pueden aprovechar estas debilidades de validación al manipular diversas facetas de las URLs de redirección: pueden añadir subdominios, usar *'localhost'* en el nombre del dominio, modificar la ruta de la URL o explorar problemas relacionados con la interpretación de URLs. Estas acciones buscan explotar fallas en la validación para potencialmente obtener tokens de acceso.

IMPROPER SCOPE VALIDATION

Esta vulnerabilidad se refiere a una situación en la que un atacante aprovecha una deficiencia en la validación del alcance (*scope*), durante el proceso de autorización en el flujo de *OAuth*, con el propósito de acceder a otros recursos no autorizados.

Durante el flujo de *OAuth*, el acceso solicitado depende del *scope* definido en la solicitud de autorización. El token generado concede a la aplicación cliente la facultad de acceder a recursos conforme al alcance aprobado por el usuario, y si este no se valida de forma rigurosa, podría permitir al atacante acceder a detalles más allá de los que se supone se ha autorizado.

En síntesis, esta vulnerabilidad reside en la omisión de una adecuada validación del alcance o *scope* en una solicitud de autorización en el contexto de *OAuth*. Y como resultado, un atacante puede manipular el parámetro, obteniendo acceso a información adicional, crítico tanto para la seguridad como para la privacidad de la aplicación y sus usuarios.

ACCESS TOKEN LEAKAGE

Esta vulnerabilidad se refiere a la posibilidad de que, durante el proceso de autentificación en el flujo de OAuth, los parámetros, incluyendo el token de acceso, se almacenen en el historial, lo que puede acarrear la exposición de información a cualquier individuo con acceso al registro histórico del navegador.

La defensa frente a esta vulnerabilidad reside en la imperiosa necesidad de gestionar los parámetros de *OAuth* y los tokens de manera segura, evitando el registro de estos.

PKCE DOWNGRADE

Este ataque guarda relación con la implementación de clave de intercambio de código o *Proof Key for Code Exchange (PKCE)*, explicada en los flujos de *OAuth*, una extensión diseñada con el propósito de mejorar la seguridad en el proceso de autorización.

En el contexto de *PKCE*, cuando un cliente inicia una solicitud de autorización, incorpora un parámetro denominado *'code_challenge_method'* en dicha petición, generalmente configurado como *'S256'*. Este parámetro indica que el 'código desafío' *(code challenge)* presentado, es el resultado del cálculo del hash *SHA-256* de dicho código. No obstante, la especificación de *PKCE* requiere que los servidores de autorización admitan también un valor *'code_challenge_method'* de *'plain'*, que indica que el *code challenge* es igual al *code verifier*, dando soporte a clientes que no pueden manejar *SHA-256* por razones técnicas.

El ataque potencial que aquí surge se refiere a la posibilidad de que un atacante degrade *'S256'* a *'plain'*, durante el intercambio del código de autorización. Aunque por lo general, *PKCE* protegería en esta situación, un atacante podría intentar ejecutar un ataque de degradación, alterando el valor del método enviado en la solicitud de intercambio del token.

Este tipo de ataque facultaría a un atacante, que ha conseguido interceptar el flujo de autorización de un usuario, a rebajar la seguridad proporcionada por *PKCE*, obteniendo así acceso no autorizado a recursos. Para evitar este ataque, es fundamental que los servidores de autorización mantengan un registro y validación coherente tanto del método como del valor del desafío a lo largo de todo el proceso.

OpenID Connect

La especificación de *OpenID Connect* es mucho más estricta que la del protocolo básico de *OAuth 2.0*, lo que significa que, por lo general, hay menos posibilidades de implementaciones que introduzcan fallos.

Dicho esto, como *OpenID Connect* es una capa que se superpone a *OAuth*, la aplicación del cliente o el servicio *OAuth* aún pueden ser vulnerables a algunos de los ataques basados en *OAuth* que analizamos anteriormente, por lo que tendremos que enfocarnos en proteger los diversos elementos que forman parte del estándar, para prevenirlos.

Improper handling of nonce claim

Este ataque, que se podría traducir como manipulación incorrecta de la solicitud *nonce*, se relaciona con el propósito del parámetro *'nonce'* de prevenir ataques de repetición, ya que, si el valor no se maneja correctamente, un atacante podría obtener acceso no autorizado aprovechando respuestas de autentificación previas, presentándose como un usuario legítimo sin tener que iniciar el proceso de autentificación.

El fallo podría ser aprovechado por un atacante en cualquiera de estas situaciones:

- **Parámetro Ausente**: Si el parámetro *'nonce'* no se incluye como parte en el proceso de autentificación y autorización, no hay forma de asociar una sesión de cliente con un *token id* específico.
- **Valor Estático**: Si se utiliza un valor estático para *'nonce'* que nunca cambia tras múltiples solicitudes, un atacante podría capturar una respuesta de autentificación y luego reutilizarla.
- **Parámetro no Validado**: Si no se valida adecuadamente el valor de *'nonce'*, un atacante podría reenviar la petición.
- **Valor Accesible**: Si el valor de *'nonce'* es accesible en texto plano para un atacante en el lado del cliente, podría capturarlo y usarlo para reenviar una respuesta de autentificación en un intento de obtener acceso no autorizado.

Fast Identity Online 2

En la búsqueda constante por fortalecer la seguridad en internet, *FIDO2* emerge como un verdadero seguro en el ámbito de la autentificación, ya que este conjunto de estándares ha revolucionado la forma en que accedemos a servicios en línea al ofrecer un enfoque sin contraseñas y basado en claves públicas.

Sin embargo, incluso esta implementación ha encontrado desafíos. *FIDO2,* a pesar de su robustez, no es inmune a la posibilidad de ataques y escenarios donde la existencia de una serie de condiciones lleva a la explotación exitosa.

Timing attacks on fido authenticator privacy

Recientemente, se ha descubierto la existencia de ciertos autentificadores *FIDO2* sobre los cuales se pueden ejecutar ataques basados en tiempo, que permiten a los atacantes vincular cuentas de usuario almacenadas en esos autentificadores vulnerables, representando una seria preocupación en términos de privacidad.

Desde una perspectiva criptográfica, el protocolo implica un simple desafío-respuesta en el que se utiliza un determinado algoritmo de firma digital.

Para proteger la privacidad del usuario, se utilizan pares de claves únicas por servicio, aunque también se debe manejar la limitación de la memoria, tarea que se lleva a cabo utilizando diversas técnicas que hacen uso de un parámetro especial llamado *'key handle'*. Este parámetro es enviado por el servicio al token, parámetro con el cual el token puede producir de manera segura una clave.

La vulnerabilidad se localiza en la forma en que se implementa el procesamiento de esos *'key handles'*. Los autentificadores vulnerables muestran una diferencia en el tiempo necesario para procesar el *'key handle'* en servicios diferentes, permitiendo a los atacantes vincular de forma remota cuentas de usuario en múltiples servicios.

Medidas de prevención

Las medidas aquí descritas pueden ayudar a una correcta implementación del protocolo escogido. Aparte de las aquí incluidas, siempre pueden existir más, pero se debe tener en consideración si van a ser un beneficio o un perjuicio incluirlas en cualquier desarrollo.

JSON Web Token

La implementación segura de este estándar implica que se deben considerar una serie de puntos clave para garantizar la integridad, autenticidad y confidencialidad de los datos transmitidos y almacenados, dirigiendo la atención a los siguientes aspectos con el objetivo de evitar ataques:

- **Elección de Algoritmo**: Seleccionar algoritmos de firma y cifrado robustos y seguros que se adapten a nuestras necesidades de seguridad y cumplimiento.
- **Gestión de Claves**: Implementar prácticas sólidas de generación, almacenamiento y rotación de claves criptográficas utilizadas en la firma y cifrado de JWT.
- **Firma y Verificación**: Verificar cuidadosamente la firma digital del token para asegurarse de que no ha sido alterado desde su emisión.
- **Control de Expiración**: Establecer tiempos de expiración adecuados para los tokens JWT, limitando su tiempo de vida y minimizando la ventana de oportunidad para ataques.
- **Autorización y Claims**: Utilizar los claims (reclamaciones) de JWT, piezas de información que se incluyen en el token para describir ciertos aspectos del titular y su autorización, para limitar los privilegios de acceso.
- **Seguridad de la Comunicación**: Transmitir los tokens JWT a través de canales seguros, como conexiones HTTPS, para evitar la interceptación y la retransmisión manipulada.

- **Prevención de Inyecciones**: Validar y escapar adecuadamente los datos que se incluyen en los claims para prevenir ataques de inyección de código, como inyección de SQL o *Cross-Site Scripting* (XSS).

Open Authorization 2

Para contrarrestar las amenazas necesitamos un enfoque exhaustivo, que involucre no solo la comprensión detallada de las especificaciones de *OAuth 2.0*, sino también la aplicación diligente de mejores prácticas en su configuración e implementación:

- **Whitelist de redirect_uri**: Las aplicaciones deberían tener una lista blanca (whitelist) de posibles entradas válidas para el parámetro redirect_uri, realizando una coincidencia exacta de la URI en lugar de una coincidencia por patrón, lo que evita que los atacantes accedan a otras páginas en los dominios de la lista blanca. Esta práctica resulta fundamental para evitar el ataque de *improper validation of redirect_uri o* validación incorrecta de la url de redirección.

- **Parámetro 'state'**: El valor del parámetro *'state'*, necesario para prevenir ataques de *Cross-site request forgery (CSRF)* -falsificación de petición en sitios cruzados -, debe ser único, no adivinable, y estar asociado con la sesión del usuario, validando que el valor recibido durante el procesamiento de la solicitud sea el mismo que el enviado.

- **Parámetro 'scope'**: Se debe proporcionar la variable *'scope'* con el acceso que se ha solicitado y no proporcionar detalles adicionales, ya que es una buena medida el minimizar la cantidad de información a la que da acceso el propietario de los recursos. Además, el servicio debe validar la variable 'scope' frente a la que solicitó el token de acceso para prevenir manipulaciones. Con esta práctica evitaríamos en gran medida el ataque de *improper scope validation* o validación incorrecta del alcance.

- **Reglas de Creación de Cuentas**: Los usuarios no deben tener la opción de crear una cuenta si ya se ha creado otra con la misma dirección de correo electrónico. Si esto sucede, se debería informar del suceso al usuario y solicitar el cambio de contraseña. Como pasaría en el ataque de ***pre-account takeover***.

- **Tokens de Acceso Reutilizables**: El token de acceso *OAuth* no debe ser reutilizable, y tiene que estar asociado con un solo usuario, caducando cuando este cierre sesión en la aplicación. Esta medida debe ayudar a prevenir el ataque de ***access token leakage*** o filtración del token de acceso.

- **Seguridad de los Tokens de Acceso**: Los usuarios otorgan su consentimiento para acceder a sus datos en su nombre, a través del token de acceso, por lo tanto, se debe garantizar la seguridad de los tokens en el envío y almacenamiento, utilizando exclusivamente canales seguros y cifrados, previniendo la exfiltración de información. Esta medida también debe de ayudar a prevenir el ataque de ***access token leakage.***

- **Validación de la Cabecera Host**: El servicio debe validar la cabedera *'host'* para comprobar que se trata de un dominio permitido, incluido en una lista blanca, y rechazar cualquier solicitud con un equipo no confiable. También se recomienda deshabilitar el soporte para cabeceras como '*X-Host*' o *'X-Forwarded-Host'*, asegurando que la cabecera no se pueda manipular.

OpenID Connect

En lo que respecta a este protocolo, se podrían poner en práctica las siguientes medidas, para intentar evitar ataques, algunas de ellas ya se han indicado en apartados anteriores:

- **Lista blanca de redirecciones**: utilizar una lista blanca de *URLs* de redirección permitidas para evitar redirecciones maliciosas durante el proceso de autentificación.
- **Validación estricta de URLs**: validar las *URLs* de redirección proporcionadas por el cliente para asegurarte de que están dentro de los dominios permitidos.
- **Cookies seguras**: configurar las cookies con las etiquetas *HttpOnly* y *Secure* para prevenir accesos no autorizados desde scripts y asegurar su transmisión solo a través de HTTPS.
- **Timeout de sesión**: establecer tiempos de expiración adecuados para las sesiones y tokens, y realiza rotación de tokens cuando sea necesario.
- **Firmas y algoritmos seguros**: Asegurarse que los *tokens ID* estén firmados utilizando algoritmos seguros, como mínimo *RS256*. Valida la firma del *token ID* para confirmar su autenticidad.
- **Validación de campos**: verificar todos los campos del *token ID*, incluyendo `iss` (emisor), `aud` (audiencia), `exp` (expiración) y `iat` (emitido en).
- **HTTPS obligatorio**: aunque este se puede presuponer, utilizar HTTPS para todas las comunicaciones entre el cliente, el servidor de autorización y el recurso protegido. Esto previene ataques de intermediarios (*man-in-the-middle*).
- **PKCE (*Proof Key for Code Exchange*)**: implementar *PKCE*, especialmente en aplicaciones móviles y de una sola página (*SPA*), para proteger el flujo de autorización contra ataques de interceptación de código de autorización.
- **MFA**: a ser posible, implementar autentificación multifactor para añadir una capa adicional de seguridad en el proceso de autentificación.
- **Alcances y permisos mínimos**: solicitar y conceder solo los permisos necesarios para reducir la exposición en caso de que el token sea comprometido.
- **Registro de eventos**: Registra todos los eventos relacionados con la autentificación y

autorización para detectar comportamientos sospechosos

Fast Identity Online 2

Aunque FIDO2 es probablemente el protocolo más seguro de todos, el hecho que se utilicen dispositivos externos, pueden hacer que el origen del ataque venga por problemas o vulnerabilidades de estos dispositivos más que por la propia implementación del protocolo. Esto hace que haya más factores a tener en cuenta y revisar. La implementación y mantenimiento de medidas de prevención contra ataques ayuda a maximizar la seguridad al utilizar el protocolo *FIDO2*, protegiendo tanto a los usuarios como a los sistemas.

- **Mantenimiento de software actualizado**: Asegurar que el software que interactúa con los autentificadores *FIDO2* esté siempre actualizado con los últimos parches de seguridad y mejoras.

- **Utilizar autentificadores certificados**: Implementar autentificadores *FIDO2* que hayan sido certificados por la *FIDO Alliance* para garantizar que cumplen con los estándares de seguridad más recientes.

- **Procedimientos de recuperación de cuenta**: Establecer procedimientos claros y seguros para la recuperación de cuentas en caso de pérdida de un autentificador *FIDO2*, asegurando que estos procedimientos no comprometan la seguridad.

- **autentificadores de respaldo**: Proveer y registrar autentificadores de respaldo para los usuarios, de manera que puedan acceder a sus cuentas incluso si se pierde el autentificador principal.

- **Pruebas de seguridad regulares**: Realizar pruebas de seguridad periódicas en los autentificadores *FIDO2* y los sistemas relacionados para identificar y corregir vulnerabilidades.

- **Control de acceso físico**: Proteger los dispositivos físicos (como tokens USB o dispositivos biométricos) mediante políticas de seguridad que limiten el acceso físico a usuarios autorizados.

- **Almacenamiento seguro**: Guardar los dispositivos de autentificación en lugares seguros cuando no se estén utilizando, para prevenir robos o pérdida.

- **Monitorización de autenticaciones**: Implementar sistemas de monitoreo para detectar patrones de autentificación inusuales o sospechosos que puedan indicar intentos de acceso no autorizados.

- **Alertas y notificaciones**: Configurar alertas para actividades inusuales relacionadas con los autentificadores FIDO2, permitiendo una respuesta rápida a posibles incidentes de seguridad.

Conclusiones

La elección del protocolo a utilizar para la autentificación, dependerá de varios factores, como pueden ser: la experiencia del equipo de desarrollo con el protocolo seleccionado; el tiempo que se tenga para desarrollar el proyecto, las configuraciones de servidores y demás infraestructura a desplegar, etcétera.

Si aunque el equipo de TI, tanto desarrollo como sistemas, no tiene experiencia en implementar ***FIDO2***, pero se dispone de **tiempo suficiente**, tanto para realizar la programación correspondiente como para las configuraciones necesarias en dispositivos y servidores, entonces sería la opción a utilizar.

Si el **tiempo** es un factor **determinante** y no se ha utilizado nunca dicho protocolo, entonces es sería recomendable utilizar tanto *OpenID Connect* como *Oauth 2.0*. La elección ya dependerá de lo cómodo que se esté con estos protocolos.

Eso si, la selección del protocolo, aunque cualquiera de ellos se considera seguro, no hace que la implementación sea segura, hay que tener en cuenta siempre las recomendaciones indicadas en los puntos anteriores como cualquier otra que mejore la seguridad del código, ya sea medidas para evitar la inyección SQL o si se utiliza LDAP, medidas para evitar la inyección de LDAP, etcétera.

Junto con el protocolo para la autentificación podría introducirse la utilización de la **autentificación adaptativa**, que no es más que autentificar a un usuario en función del nivel de riesgo presentado por un intento de inicio de sesión. se basa en la evaluación del riesgo de cada solicitud de autentificación y tomar medidas adicionales cuando se considera necesario.

En situaciones normales, de bajo riesgo, los usuarios pueden iniciar sesión sin obstáculos, reduciendo la complejidad para aquellos que inician sesión en sus cuentas desde ubicaciones, sistemas y horarios de confianza. Sin embargo, si se detecta un comportamiento inusual o sospechoso, se activa la autentificación escalonada y se solicita al usuario autentificación adicional, como el envío de notificaciones para verificar la identidad, el uso de códigos de un solo uso o la verificación a través de un dispositivo. En el caso de que no se hayan cumplido los distintos métodos de autentificación en un determinado espacio temporal, se procedería a bloquear la cuenta.

Y todo ello, siendo debidamente monitorizado y registrado.