Autenticación de Microservicios

Autenticación de microservicios basados en OAuth

Paul Andrés Rodríguez Chiriboga
Maestría en Ciberseguridad y Privacidad
Seguridad Empresarial

Nombre Tutor/a de TF: Manuel Jesús Mendoza Flores

Fecha: octubre de 2022

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Autenticación de microservicios basados en OAuth*
Nombre del autor:	*Paul Andrés Rodríguez Chiriboga*
Nombre del consultor/a:	*Manuel Jesús Mendoza Flores*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	*01/2023*
Titulación o programa:	*Maestría en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*Seguridad Empresarial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Microservices, Authentication, OAuth, Security*

Resumen del Trabajo

Las arquitecturas de microservicios son de amplio uso en muchas empresas en la actualidad que buscan guiar el desarrollo del software bajo la separación de funcionalidades. No obstante, estas arquitecturas suponen un conjunto de retos a tener en cuenta por los equipos de desarrollo, y en donde la seguridad es uno de los puntos con mayor importancia. En este sentido, OAuth se plantea como una alternativa para autenticación y autorización que puede ser usado para asegurar el entorno de microservicios. Sin embargo, este protocolo tiene debilidades para las cuales se debe tener en cuenta consideraciones de seguridad con el objetivo de fortalecer el entorno. Este conjunto de amenazas y consideraciones de seguridad se encuentra recopilado en el documento RFC-6819. En el presente trabajo, se procede a planificar, diseñar, desarrollar y ejecutar un entorno construido en base a una arquitectura de microservicios que hagan uso del protocolo de autorización de OAuth 2.0. De la misma manera, se analiza y configura el entorno siguiendo los lineamientos de seguridad expuestos en el documento RFC-6819.

Abstract

Microservice architectures are nowadays widely used in many companies looking to lead software development under the separation of functionalities. Nevertheless, these architectures pose a set of challenges to be taken into account by development teams, where security is one of the main focuses of greatest importance. In this way, OAuth is considered as an authentication and authorization alternative that can be used to secure the microservices environments. However, this protocol has weaknesses and security considerations must be taken into account in order to strengthen the environment. This set of threats and security considerations are compiled in the RFC 6819 document. In the present document, we proceed to plan, design, develop and execute an environment built based on a microservice architecture that uses of the OAuth 2.0 authorization protocol. In the same way, the environment is analyzed and configured following the security guidelines exposed in the RFC 6819 document.

Índice

1. Introducción 1
1.1. Contexto y justificación del Trabajo 1
1.2. Objetivos del Trabajo 2
1.3. Impacto en sostenibilidad, ético-social y de diversidad 2
1.4. Enfoque y método seguido 3
1.5. Planificación del Trabajo 4
1.6. Breve sumario de productos obtenidos 9
1.7. Breve descripción de los otros capítulos de la memoria 9
2. Marco teórico 10
2.1 Microservicios 10
2.1.1 Aplicaciones Monolíticas 10
2.1.2 Arquitectura de Microservicios 11
2.1.3 Aplicación Monolítica vs Arquitectura de Microservicios 13
2.2 Protocolo OAuth 14
2.2.1 OAuth 1.0 15
2.2.1.1 Flujo de autorización 16
2.2.2 OAuth 2.0 17
2.2.2.1 Componentes 17
2.2.2.2 Tipos de Concesión de Autorización 18
2.2.2.3 Tokens 19
2.2.3 Flujo de autorización abstracto 20
2.2.4 Ventajas del uso de OAuth 2.0 21
2.3 Documento RFC-6819 21
2.3.1 Concepto 21
2.3.2 Flujo de autorización OAuth 2.0 por códigos de autorización 21
2.3.3 Características de los despliegues 23
2.3.4 Supuestos de los ataques 23
2.3.5 Modelo de Amenazas y Recomendaciones 24
2.3.5.1 Dirigidas al Cliente 24
2.3.5.2 Endpoint de Autorización 27
2.3.5.3 Endpoint de Token 28
2.3.5.4 Flujo: Obteniendo Autorización 30
2.3.5.4.1 Concesión: Código de Autorización 30
2.3.5.4.2 Concesión: Implícita 33
2.3.5.4.3 Concesión: Credenciales de contraseña del propietario del recurso 35
2.3.5.4.4 Concesión: Credenciales del cliente 37
2.3.5.5 Flujo: Actualizando un token de acceso 38
2.3.5.6 Flujo: Accediendo a un recurso protegido 39
3. Desarrollo e Implementación 41
3.1 Patrones en Arquitecturas de Microservicios 41
3.2 Definiendo una arquitectura de microservicios 42
3.3 Diseño de la arquitectura de microservicios 44

3.4 Desarrollo de la arquitectura de microservicios 46
3.4.1 Desarrollo del API Gateway 46
3.4.2 Desarrollo del Servidor de Autorización 47
3.4.3 Desarrollo de los Microservicios 49
3.5 Ejecución de la arquitectura de microservicios 50
4. Seguridad en el entorno de microservicios usando el protocolo OAuth 54
4.1 Conceptos básicos de Keycloak 54
4.2 Seguridad de los clientes 55
4.2.1 Entropía en los secretos del cliente 56
4.2.2 Clientes públicos y de baja confianza 57
4.2.3 Secretos específicos para despliegues específicos 60
4.3 Seguridad de los usuarios 61
4.3.1 Número de intentos de acceso permitidos 61
5. Conclusiones y trabajos futuros 63
6. Glosario 66
7. Bibliografía 67
8. Anexos 68
Anexo 1: Configuración básica del Servidor de Autorización usando Keycloak 68

Lista de figuras

Figura 1: Cronograma de Actividades 8
Figura 2: Arquitectura monolítica 11
Figura 3: Arquitectura de Microservicios 11
Figura 4: Diferencia entre Arquitectura Monolítica y Microservicios. 14
Figura 5: Roles de delegación de identidad. 15
Figura 6: Flujo de Autorización de OAuth 1.0. 16
Figura 7: Componentes OAuth 2.0. 17
Figura 8: Flujo de código de Autorización OAuth 2.0 18
Figura 9: Flujo de Concesión Implícita. 18
Figura 10: Flujo de concesión por credenciales de contraseña del propietario del recurso. 19
Figura 11: Flujo de concesión por credenciales del cliente. 19
Figura 12: OAuth 2.0 - Flujo de Autorización básico 20
Figura 13: Flujo de Autorización Completo OAuth 2.0. 22
Figura 14: Flujo y datos usados por el cliente en un entorno OAuth 2.0. 25
Figura 15: Datos usados por el endpoint de autorización en un flujo OAuth 2.0. 27
Figura 16: Flujo y datos usados por el endpoint de token en un flujo OAuth 2.0. 29
Figura 17: Diagrama de concesión por código de Autorización. 31
Figura 18: Diagrama de concesión Implícita. 34
Figura 19: Diagrama de Concesión por Credenciales de contraseña del Propietario del Recurso. 36
Figura 20: Diagrama de concesión por credenciales del cliente. 37
Figura 21: Flujo de actualización de un token de acceso. 38
Figura 22: Flujo de acceso a un recurso protegido. 39
Figura 23: Lenguaje de patrones de arquitecturas de microservicios. 41
Figura 24: Puerta de enlace API (API Gateway). 42
Figura 25: API Gateway como un cliente OAuth 2.0. 43
Figura 26: API Gateway como un servidor de recursos OAuth 2.0. 43
Figura 27: Diagrama de secuencia de una arquitectura de microservicios con un API Gateway como un cliente OAuth. 44
Figura 28: Funcionamiento de los componentes a desarrollar. 45
Figura 29: Servidor de Autorización con Keycloak. 51
Figura 30: Ejecución de la arquitectura de microservicios hacia el microservicio 1 52
Figura 31: Ejecución de la arquitectura de microservicios hacia el microservicio 2 52
Figura 32: Ejecución de la arquitectura de microservicios hacia el microservicio 3 52
Figura 33: Ejecución conjunta de toda la infraestructura de OAuth2.0 y microservicios 53
Figura 34: Registro de clientes en servidor Keycloak 55
Figura 35: Tipos de clientes en servidor Keycloak 56
Figura 36: Secretos del cliente de Keycloak 56
Figura 37: Nivel de entropía del secreto del cliente 57
Figura 38: API Gateway como un cliente público 57
Figura 39: Solicitud de acceso para el API Gateway como cliente público 58
Figura 40: Acceso a recursos protegidos por el microservicio 2 usando un token de acceso 59
Figura 41: Solicitud de acceso denegado por falta del secreto del cliente 59
Figura 42: Registro de un cliente para el API Gateway para Insomnia 60
Figura 43: Deshabilitado de un cliente específico 61
Figura 44: Solicitud de autorización rechaza por un cliente deshabilitado 61
Figura 45: Configuración de seguridad contra ataques de fuerza bruta 62

Lista de tablas

Tabla 1: Cronograma de Actividades **8**
Tabla 2: Beneficios de ambas Arquitecturas **13**
Tabla 3: Inconvenientes de ambas Arquitecturas **14**
Tabla 4: Modelo de amenazas y recomendaciones para el Cliente OAuth2.0 **27**
Tabla 5: Modelo de amenazas y recomendaciones para el endpoint de autorización OAuth2.0 **28**
Tabla 6: Modelo de amenazas y recomendaciones para el endpoint de token OAuth2.0 **30**
Tabla 7: Modelo de amenazas y recomendaciones para la concesión por código de autorización OAuth2.0. **33**
Tabla 8: Modelo de amenazas y recomendaciones para la concesión Implícita OAuth2.0. **35**
Tabla 9: Modelo de amenazas y recomendaciones para la concesión por contraseña del propietario del recurso OAuth2.0. **37**
Tabla 10: Modelo de amenazas y recomendaciones para el flujo de actualización de tokens de acceso OAuth2.0. **39**
Tabla 11: Modelo de amenazas y recomendaciones para el flujo de acceso a recursos protegidos OAuth2.0. **40**
Tabla 12: Definición de componentes a desarrollar. **45**
Tabla 13: Tecnologías usadas para el desarrollo del API Gateway **46**
Tabla 14: Tecnologías usadas para el desarrollo del Servidor de Autorización **48**
Tabla 15: Tecnologías usadas para el desarrollo de los Microservicios **49**
Tabla 16: Compilación y ejecución de componentes. **50**
Tabla 17: Características del despliegue de los componentes **51**
Tabla 18: Consideraciones de seguridad implementadas acorde al RFC 6819 **64**
Tabla 19: Tabla de cobertura de objetivos **65**

1. Introducción

Cuando se pretende desarrollar una aplicación de lado del servidor, tenemos dos grandes paradigmas que últimamente han dominado el desarrollo moderno actual: Las Arquitecturas Monolítica y las de Microservicios. Sin embargo, la evolución de las arquitecturas de software ha sido impulsada por la necesidad de lograr una mejor separación de funcionalidades (Blinowski, Ojdowska, & Przybylek, 2022).

De allí que deviene la idea de dividir una aplicación en unidades pequeñas, pequeños servicios interconectados que ejecuten varias tareas en conjunto en lugar de un solo servicio monolítico. Cada uno de estos pequeños servicios, se le denomina microservicios (Kharenko, 2015).

1.1. Contexto y justificación del Trabajo

Con la transición de sistemas monolíticos hacia microservicios, muchos problemas comenzaron a emerger. Uno de ellos son los sistemas de autenticación y autorización en microservicios. En aplicaciones monolíticas sólo hacía falta de un único proceso de autenticación y autorización; sin embargo, en sistemas compuestos por microservicios, es necesario sistemas más complejos que manejen las relaciones entre microservicios y sus niveles de acceso (Triartono, Muldina Negara, & Sussi, 2019).

En este sentido y, dado los masivos ataques reportados en compañías que adoptan las arquitecturas de microservicios, como Netflix o Amazon, lidiar con la seguridad en estas arquitecturas es algo necesariamente urgente (Hannousse & Yahiouche, 2020).

Para ello, se han desarrollado algunos enfoques que pretenden verificar la identidad de un microservicio que busca tener acceso a recursos restringidos, entre los que se encuentran los métodos basados en JWT (JSON Web Token), el uso de API Gateway como único punto de entrada al sistema mediante una petición verificada, la realización de Single Sign-On (SSO) utilizando un proveedor de identidad, el uso del protocolo OAuth, entre otros (Yang, Huo, Li, & Zhu, 2021).

Por otra parte, el uso de OAuth nos permite obtener autorización para el acceso a un recurso desde un servicio; por tal motivo, es un protocolo válido de autenticación y autorización que se puede aplicar a microservicios (Triartono, Muldina Negara, & Sussi, 2019). Mientras OAuth 1.0 es un protocolo estandarizado que define el modelo teórico de autenticación y autorización, OAuth 2.0 es un *framework* que nace a partir de la recopilación de buenas prácticas en la aplicación del protocolo y que permite, a aplicaciones de terceros, obtener acceso limitado a un recurso protegido de un servicio HTTP (OAuth Working Group, 2012).

De la misma manera, la aplicación del protocolo de OAuth trae consigo retos de seguridad a enfrentar debido a debilidades presentes en la propia naturaleza del mismo. Sin embargo, gracias a un conjunto de consideraciones adicionales a tener presentes el momento de implementar el protocolo, podemos mitigar las dichas amenazas. Este conjunto de amenazas y consideraciones ha sido recopilado en el documento RFC 6819 (RFC-6819, 2013).

Por tal motivo, el presente trabajo se centra en el desarrollo e implementación de un entorno basado en una arquitectura de microservicios usando el protocolo de autenticación y autorización OAuth2.0 tomando en cuenta las consideraciones adicionales de seguridad expuestas en el documento RFC 6819.

Se analizarán los posibles ataques que pueden surgir contra el protocolo OAuth2.0 y se realizarán configuraciones de seguridad sobre el entorno siguiendo las recomendaciones del documento RFC 6819. Finalmente se exponen un listado de conclusiones del trabajo y se enumeran propuestas a ser consideradas para trabajos futuros.

1.2. Objetivos del Trabajo

El objetivo principal del presente trabajo es desarrollar un entorno basado en una arquitectura de microservicios que hagan uso del protocolo de autorización OAuth, teniendo en consideración las recomendaciones básicas de seguridad que se exponen en el documento RFC 6819. Este objetivo principal se apoya de los siguientes objetivos específicos:

- Identificar el concepto de microservicios, su utilidad e importancia en los entornos Cloud en la actualidad.
- Conocer el protocolo OAuth2.0 en tanto a su funcionamiento, operación e importancia como un sistema de autenticación y autorización de microservicios.
- Identificar las distintas debilidades y consideraciones de seguridad del protocolo OAuth2.0 expuestos en el documento RFC 6819.
- Desarrollar un entorno basado en una arquitectura de microservicios que hagan uso del protocolo OAuth2.0 para autenticación y acceso a recursos protegidos.
- Configurar el entorno para tener en consideración las recomendaciones de seguridad expuestas en el documento RFC 6819.
- Exponer las conclusiones del trabajo realizado y estudios futuros que sean necesarios para complementar el tema.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

El resultado del presente trabajo se define en las siguientes dimensiones:

Dimensión de sostenibilidad: El presente trabajo no involucra ningún impacto en los aspectos de sostenibilidad ya que no está alineado con ningún ODS (Objetivo de Desarrollo sostenible).

Dimensión de diversidad de género y derechos humanos: el presente trabajo no tiene ningún impacto referente a aspectos de diversidad de género y derechos humanos ya que no viola ni mejora ninguno de sus lineamientos al ser únicamente de carácter técnico.

Dimensión de comportamiento ético y de responsabilidad social: como objetivo central, el presente TFM es el desarrollo y la configuración adecuada de seguridad de un entorno basado en una arquitectura de microservicios. La violación de la seguridad de los microservicios implica un impacto negativo que pudiere suplantar la identidad de un usuario; así como capturar datos violentando la privacidad del mismo. Por tal motivo, el presente trabajo representa un impacto positivo en tanto se procede a implementar los lineamientos de seguridad en la autenticación y autorización de microservicios y dar un enfoque práctico de cómo usarlos para asegurar los datos y la identidad de los mismos.

Tecnologías usadas y costos asociados: El presente trabajo no representa un impacto económico adicional, dado que su implementación no genera valores adicionales a los normales del giro de negocio de las aplicaciones que hacen uso de arquitecturas de microservicios. De la misma manera, su aplicación involucra el uso de frameworks y tecnologías de código abierto y gratuitas tales como: OAuth, Keycloak, Java, Kotlin, SpringBoot, Insomnia Community Edition, Maven, Gradle, Tmux, etc. Para mayor descripción de tecnologías, vea el apartado 3.4.

1.4. Enfoque y método seguido

Para el presente trabajo, se toma en consideración el siguiente conjunto de fases que se seguirán en el trayecto del mismo:

Definición del plan de trabajo

En esta primera etapa se describe la problemática que se va a tratar, su contexto y justificación. Se definen los objetivos del mismo y las tareas necesarias a realizar. Se analiza el impacto ético-social y ambiental del mismo donde se exponen los impactos del presente proyecto. Se continúa elaborando el plan de trabajo con su cronograma, se define el enfoque y la metodología adecuada y se definen los entregables al finalizar el mismo.

Microservicios

En esta etapa se define el concepto de aplicación monolítica y de microservicios, sus características, funcionamiento e importancia en el escenario tecnológico actual. Se realiza una comparativa entre ambos enfoques con especial énfasis en la ventaja y desventaja de la implementación de cada enfoque.

Protocolo OAuth

En esta etapa se desarrolla un concepto de delegación de identidad, el protocolo OAuth1.0 y 2.0, sus actores y componentes, su funcionamiento, su terminología

base, sus ventajas y desventajas y en que nos aportan en cuanto a la autenticación y autorización de microservicios. Una parte importante para introducirnos en el mundo del protocolo.

Documento RFC 6819.

En esta etapa vamos a describir el conjunto de debilidades y amenazas del protocolo OAuth2.0; así como también definiremos cuales son las consideraciones generales de seguridad que especifica el documento para mitigar los ataques ya mencionados.

Desarrollo de una infraestructura de microservicios con OAuth

Esta etapa conlleva el desarrollo del servidor de autorización de OAuth 2.0. De la misma manera se desarrollarán 3 microservicios que expongan un recurso protegido mediante un API Gateway que servirá como punto de entrada de los mismos. Finalmente haremos que estos microservicios soliciten autorización al servidor de OAuth 2.0.

Configuración de seguridad del entorno acorde al documento RFC 6819

Esta etapa se conforma de la configuración necesaria que debe realizarse sobre el entorno de microservicios, con el fin de mitigar los posibles ataques al protocolo OAuth2.0 siguiendo las recomendaciones expuestas en el documento RFC 6819.

Conclusiones y trabajos futuros

En este apartado, se definirán cada una de las conclusiones que fueron emergiendo a lo largo del proceso de desarrollo y configuración del entorno de microservicios. De la misma manera, se expondrán las recomendaciones necesarias para extender el presente trabajo en lo posible.

1.5. Planificación del Trabajo

Empezaremos por describir las tareas a realizar acorde a las fases descritas en el apartado anterior:

- **Definición de la temática:**
  Análisis previo de la documentación: en esta primera parte se analiza brevemente el contexto del trabajo con el fin de focalizar el tema acorde a las necesidades.
- **Definición del plan de trabajo:**
  Introducción: se redacta una breve descripción inicial que introducirá al tema.

Contexto y justificación del trabajo: se define un contexto del trabajo procedente del estado del arte y se justifican los propósitos y razones del presente trabajo.
Objetivos del trabajo: se define el objetivo principal del trabajo y, a su vez, se enumeran los objetivos específicos que se deben satisfacer para cumplir con el objetivo general.
Impacto en sostenibilidad, ético-social y diversidad: se describen los recursos a utilizar, costes en ámbitos en el momento de despliegue y durante la vida del producto y su impacto ético-social y ambiental.
Enfoque y método seguido: se describen el enfoque que va a seguir el trabajo y el método a usar mediante las fases a seguir durante el trabajo.
Planificación del trabajo: se enumeran las tareas a realizar y se coloca un cronograma a seguir para la ejecución de cada tarea mediante un diagrama de Gantt.
Breve sumario de productos obtenidos: se enumeran los productos que serán el resultado al final de cada fase.
Breve descripción de otros capítulos de la memoria: se enumeran y definen los capítulos que contiene el trabajo de manera breve.

- **Microservicios:**
  Aplicaciones monolíticas: se define el concepto y características de las aplicaciones monolíticas.
  Arquitectura de Microservicios: se define el concepto y las características comunes de las arquitecturas de microservicios.
  Aplicación monolítica vs Microservicios: se presentan las principales diferencias entre ambos enfoques haciendo especial énfasis en las ventajas y desventajas de cada uno.
- **Protocolo OAuth:**
  OAuth 1.0: se define el protocolo OAuth 1.0, su funcionamiento y principales características.
  Flujo de Autorización: se define y explica el flujo de autorización que sigue el protocolo para la concesión de acceso.
  OAuth 2.0: se plantea una definición formal del *framework* OAuth 2.0 y sus características.
  Componentes: se identifica y enumera los actores y componentes necesarios para la aplicación de un flujo de autorización con el protocolo.
  Tipos de concesión de autorización: se identifica y enumera los distintos tipos de flujos que los componentes pueden seguir en OAuth 2.0 para interactuar entre sí.
  Tokens: se define el concepto de *token* y se identifica y enumera los distintos tipos de tokens que existen en OAuth 2.0.
  Flujo de autorización abstracto: definición de todas las partes y pasos que involucran un flujo de autorización con OAuth2.0.
  Ventajas del uso de OAuth 2.0: se describen las ventajas del uso de este protocolo de autorización en un ambiente de microservicios.
- **Documento RFC-6819:**
  Concepto: se plantea una descripción del documento y sus puntos principales.
  Flujo de autorización OAuth2.0 por códigos de autorización: se explica y describe un flujo completo de autorización OAuth 2.0 bajo el tipo de concesión por códigos de autorización con el objetivo de identificar los

puntos clave donde se enfocan las amenazas y consideraciones de seguridad.
Características de los despliegues: para describir los modelos de amenazas y consideraciones de seguridad de OAuth 2.0, es necesario definir las características que debe tener un despliegue OAuth 2.0 para seguir estos modelos.
Supuestos de los ataques: se definen los supuestos a considerar para que el despliegue del protocolo sea susceptible de sufrir dichas amenazas.
Modelos de amenazas y recomendaciones: se clasifica y describe las amenazas que pueden afectar al protocolo OAuth; así como un conjunto de consideraciones de seguridad para mitigarlas acorde a lo expuesto en el documento RFC 6819.

- **Desarrollo e implementación:**
  Patrones de arquitecturas de microservicios: se describe un conjunto de patrones que incluyen los componentes y arquitecturas que puede tener un entorno de microservicios.
  Definiendo una arquitectura de microservicios: se definen los requerimientos que debe tener la arquitectura de microservicios a desarrollar y como se integran con OAuth 2.0.
  Diseño de la arquitectura de microservicios: se describe el diseño y modelado de cada uno de los componentes que contendrá el entorno, así como su funcionamiento general.
  Desarrollo de la arquitectura de microservicios: se desarrolla cada componente del entorno de pruebas describiendo cada proceso realizado.
  Ejecución de la arquitectura de microservicios: se describe y aplica el proceso de implementación y montaje del entorno de microservicios desarrollado.
- **Seguridad en el entorno de microservicios con OAuth 2.0:**
  Conceptos básicos de Keycloak: se enumeran y definen un conjunto de conceptos para entender el funcionamiento del servidor de autorización implementado con Keycloak.
  Seguridad de los clientes: se identifican y realizan las configuraciones necesarias para mitigar las vulnerabilidades del protocolo acorde al documento RFC-6819 referente a los clientes OAuth.
  Seguridad de los usuarios: se identifican y realizan las configuraciones necesarias para mitigar las vulnerabilidades del protocolo acorde al documento RFC-6819 referente a los propietarios de los recursos (usuarios) OAuth.
- **Conclusiones y trabajos futuros:**
  Conclusiones: se enumeran las conclusiones resultado del trabajo y sus planteamientos.
  Trabajos futuros: se plantean los posibles complementos al trabajo que ayudarán a reforzar el tema para ser propuestos como futuros trabajos.
- **Video de presentación:**
  Planificación: se planifica el contenido, tiempos y estructura del video de presentación del trabajo.
  Producción: grabación y edición del video de presentación del trabajo.
  Entrega: Entrega del video de presentación.

- **Defensa del trabajo final de Máster**
  Defensa final: se defiende el trabajo final del Máster ante el jurado seleccionado.

Acorde con estas tareas, se plantea un cronograma de actividades en formato diagrama de Gantt de la siguiente manera:

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Trabajo final de Máster**	**88 días**	**28/9/2022**	**27/1/2023**
2	**Definición de la temática**	**2 días**	**28/9/2022**	**29/9/2022**
3	Análisis previo de la documentación	2 días	28/9/2022	29/9/2022
4	**Definición del plan de trabajo**	**6 días**	**30/9/2022**	**9/10/2022**
5	Introducción	1 día	30/9/2022	30/9/2022
6	Contexto y justificación del trabajo	1 día	3/10/2022	3/10/2022
7	Objetivos del trabajo	1 día	4/10/2022	4/10/2022
8	Impacto en sostenibilidad, ético-social y diversidad	1 día	5/10/2022	5/10/2022
9	Enfoque y método seguido	1 día	6/10/2022	6/10/2022
10	Planificación del trabajo	1 día	7/10/2022	7/10/2022
11	Breve sumario de productos obtenidos	1 día	8/10/2022	8/10/2022
12	Breve descripción de otros capítulos de la memoria	1 día	9/10/2022	9/10/2022
13	**Microservicios**	**3 días**	**12/10/2022**	**14/10/2022**
14	Aplicaciones monolíticas	1 día	12/10/2022	12/10/2022
15	Arquitectura de Microservicios	1 día	13/10/2022	13/10/2022
16	Aplicación monolítica vs Microservicios	1 día	14/10/2022	14/10/2022
17	**Protocolo OAuth**	**8 días**	**17/10/2022**	**26/10/2022**
18	OAuth 1.0	1 día	17/10/2022	17/10/2022
19	Flujo de Autorización	1 día	18/10/2022	18/10/2022
20	OAuth 2.0	1 día	19/10/2022	19/10/2022
21	Componentes	1 día	20/10/2022	20/10/2022
22	Tipos de concesión de autorización	1 día	21/10/2022	21/10/2022
23	Tokens	1 día	22/10/2022	22/10/2022
24	Flujo de autorización abstracto	1 día	25/10/2022	25/10/2022
25	Ventajas del uso de OAuth 2.0	1 día	26/10/2022	26/10/2022
26	**Documento RFC-6819**	**7 días**	**27/10/2022**	**4/11/2022**
27	Concepto	1 día	27/10/2022	27/10/2022
28	Flujo de autorización OAuth 2.0 por códigos de autorización	1 día	28/10/2022	28/10/2022
29	Características de los despliegues	1 día	29/10/2022	29/10/2022
30	Supuestos de los ataques	1 día	31/10/2022	31/10/2022
31	Modelo de Amenazas y recomendaciones	4 días	1/11/2022	4/11/2022
32	**Desarrollo e implementación**	**20 días**	**5/11/2022**	**3/12/2022**
33	Patrones de arquitecturas de microservicios	1 día	5/11/2022	5/11/2022
34	Definiendo una arquitectura de microservicios	2 días	6/11/2022	7/11/2022

35	Diseño de una arquitectura de microservicios	2 días	8/11/2022	9/11/2022
36	Desarrollo de la arquitectura de microservicios	17 días	10/11/2022	2/12/2022
37	Ejecución de la arquitectura de microservicios	1 día	3/12/2022	3/12/2022
38	**Seguridad en el entorno de microservicios con OAuth 2.0**	**15 días**	**5/12/2022**	**23/12/2022**
39	Conceptos básicos de Keycloak	1 día	5/12/2022	5/12/2022
40	Seguridad de los clientes	7 días	6/12/2022	14/12/2022
41	Seguridad de los usuarios	7 días	15/12/2022	23/12/2022
42	**Conclusiones y trabajos futuros**	**4 días**	**2/1/2023**	**5/1/2023**
43	Conclusiones	3 días	2/1/2023	4/1/2023
44	Trabajos futuros	1 día	5/1/2023	5/1/2023
45	**Video de presentación**	**5 días**	**11/1/2023**	**17/1/2023**
46	Planificación	1 día	11/1/2023	11/1/2023
47	Producción	3 días	12/1/2023	16/1/2023
48	Entrega	1 día	17/1/2023	17/1/2023
49	**Defensa del trabajo final de Máster**	**5 días**	**23/1/2023**	**27/1/2023**
50	Defensa final	5 días	23/1/2023	27/1/2023

Tabla 1: Cronograma de Actividades
Fuente: Elaboración propia

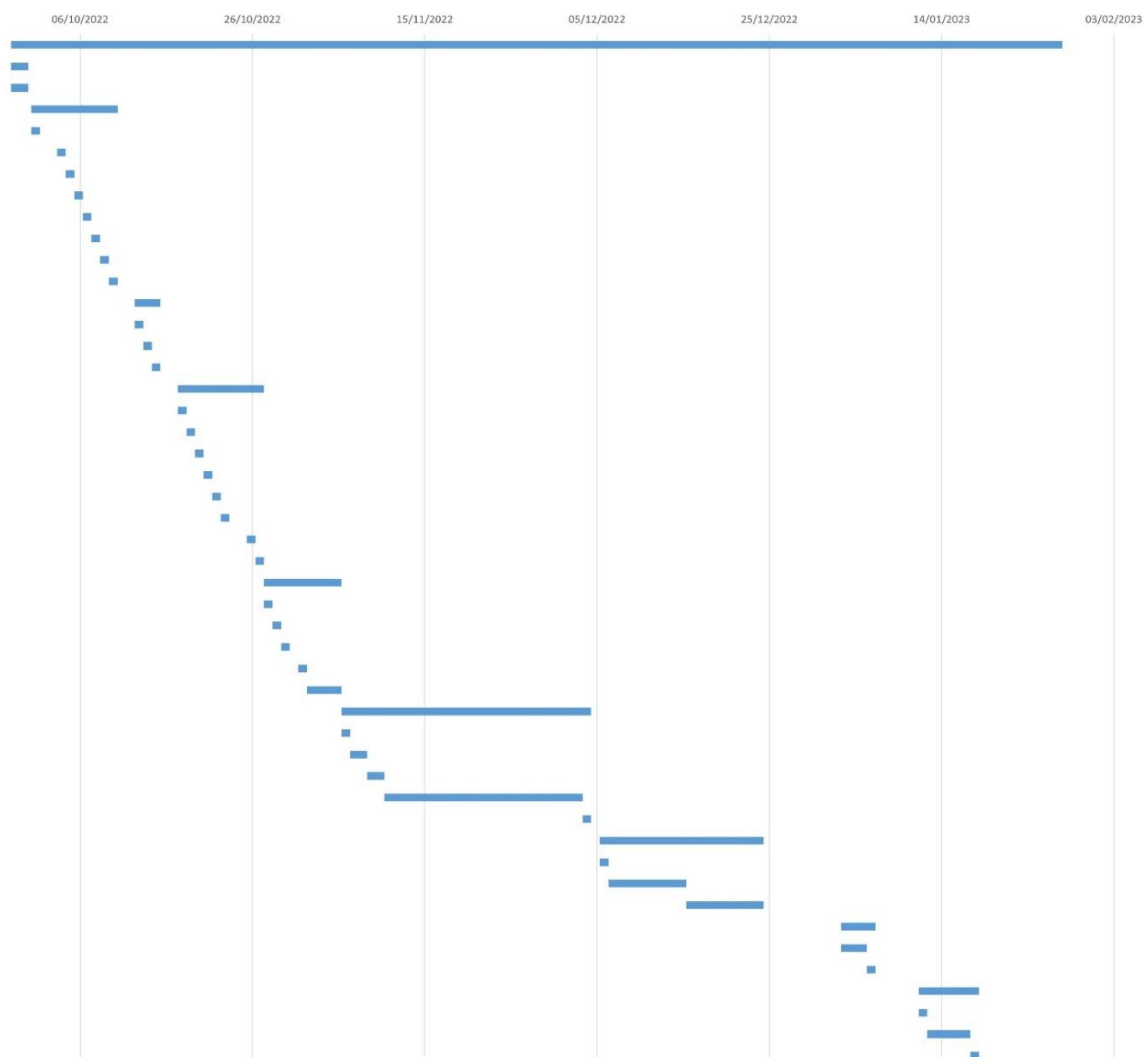


Figura 1: Cronograma de Actividades
Fuente: Elaboración propia

1.6. Breve sumario de productos obtenidos

El presente trabajo final de Máster se compone de un proyecto único e íntegro que se divide en los siguientes entregables parciales:

- PEC 1: Se trata de un entregable parcial que se compone del plan de trabajo, donde se define la problemática, el contexto y justificación, los objetivos, el impacto ético-social, la descripción de las tareas a realizar, un cronograma de trabajo y una descripción de los entregables a lo largo del trabajo.
- PEC 2: En esta entrega parcial, se entregarán un análisis teórico de las partes que componen el trabajo. Este marco será la base teórica de todo el conjunto práctico posterior.
- PEC 3: Esta entrega parcial contendrá el desarrollo e implementación de una infraestructura de microservicios usando el framework de autorización OAuth 2.0.
- PEC 4: Esta entrega parcial contiene la configuración de seguridad del entorno de microservicios acorde al documento RFC-6819. De la misma manera, se finaliza exponiendo las conclusiones y trabajos futuros del presente trabajo.
- Presentación del video: en este entregable parcial, se planificará y realizará un video donde se explique a detalle el trabajo realizado y sus conclusiones.

1.7. Breve descripción de los otros capítulos de la memoria

En los siguientes apartados y capítulos se detalla la siguiente información:

Capítulo 2 - Marco Teórico: en este apartado vamos a definir teóricamente y realizar un breve análisis de las partes necesarias para el desarrollo del presente trabajo, en tanto a Microservicios, el protocolo OAuth2.0, así como el documento RFC-6819.

Capítulo 3 – Desarrollo e implementación: en este apartado vamos a planificar, diseñar, desarrollar e implementar una infraestructura que se compone de un conjunto de 3 microservicios, un API Gateway y un Servidor de Autorización para aplicar un entorno acorde al framework OAuth2.0.

Capítulo 4 – Seguridad en un entorno de Microservicios: en este apartado vamos a analizar la seguridad en los entornos de microservicios que hacen uso del framework OAuth2.0 siguiendo los lineamientos del documento RFC-6819. Luego se procede a implementar las soluciones planteadas en el entorno desarrollado en el capítulo 3.

Capítulo 5 – Conclusiones y recomendaciones: en este capítulo se exponen las conclusiones obtenidas durante el presente trabajo, así como se enumeran una propuesta de trabajos futuros que complementarán al mismo.

2. Marco teórico

Para el desarrollo del presente trabajo, vamos a definir un conjunto de conceptos que sirven como una base teórica y que comprenden de la definición y características de microservicios, el protocolo OAuth 2.0, así como el documento RFC-6819 en sus distintas partes.

2.1 Microservicios

Continuamente las empresas se encuentran usando y mejorando arquitecturas para construir sus sistemas con el objetivo de conseguir mayor beneficio y menor costo. En este contexto, las arquitecturas de software se han guiado bajo la necesidad de separar las funcionalidades (Blinowski, Ojdowska, & Przybylek, 2022); y es donde las arquitecturas de microservicios han empezado a ganar terreno.

Un microservicio, acorde a varios autores, se define como:

- (Newman, 2015) establece que "los microservicios son servicios pequeños y autónomos que trabajan juntos".
- (Hunter II, 2017) define que "un microservicio es un tipo especial de servicio construido y realizado con menos funciones como sea posible".
- (Mayer & Weinreich, 2017) define que un microservicio es "un servicio autónomo que puede ser desplegado y operado independientemente de otro".

Es así que podemos definir un Microservicio como un sistema independiente y autónomo que es destinado a una función específica y puede ser desplegado y operado por sí mismo.

Un microservicio, cumple con las siguientes características (Newman, 2015):

- **Pequeño, enfocado en hacer una cosa bien:** un servicio limitado a una funcionalidad y manteniéndolo de esta manera para evitar que crezca demasiado, con todas las dificultades que esto puede introducir. El tamaño del microservicio dependerá del negocio.
- **Autónomo**: un microservicio es una entidad separada y debe ser desplegado como un servicio aislado en un contenedor o sistema operativo. La única comunicación que se realiza entre microservicios, se la realiza mediante llamadas de red de manera independiente.

2.1.1 Aplicaciones Monolíticas

En el contexto de las arquitecturas de sistemas, un sistema monolítico significa "todo compuesto en una pieza". En este sentido, un software monolítico es diseñado para ser autocontenido, donde las funciones y los componentes se encuentran acoplados y presentes en el mismo código para ser compilados y desplegados (Awati & Wigmore, 2022).

En una aplicación monolítica, muchos componentes se combinan en una sola aplicación larga donde usualmente un cambio en un componente, demanda rescribir otro componente y compilar todo el conjunto para ser ejecutado sin tomar en cuenta que componente realmente usa el usuario (Awati & Wigmore, 2022).

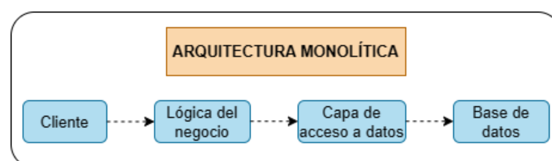


Figura 2: Arquitectura monolítica
Fuente: https://www.techtarget.com/whatis/definition/monolithic-architecture

2.1.2 Arquitectura de Microservicios

La idea de un microservicio es aislar una funcionalidad del resto para que sea tratada de manera independiente. Sin embargo, para abarcar con un negocio, que usualmente contiene varias funcionalidades, es necesario de un conjunto de microservicios operando entre sí.

De ahí que viene la necesidad de crear patrones de arquitecturas de sistemas que puedan implementar una lógica de negocio de una organización usando estos microservicios, o servicios de propósito único, en contraste con el método tradicional de construir un servicio monolítico que contenga todas las funcionalidades (Hunter II, 2017).

Formalmente, el término "Arquitectura de Microservicios" ha surgido en los últimos años para describir una forma particular de diseñar aplicaciones de software conformados por conjuntos de servicios que sean desplegables de manera independiente (Fowler & Lewis, 2014).

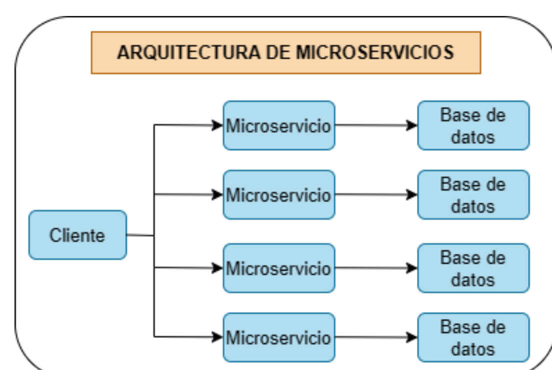


Figura 3: Arquitectura de Microservicios
Fuente: https://www.techtarget.com/whatis/definition/monolithic-architecture

Este conjunto de servicios, que conforman una aplicación de software, cumple con las siguientes características (Fowler & Lewis, 2014):

- **Componentización mediante servicios:** un componente es definido como una unidad de software que es independientemente reemplazable y actualizable. Esta característica nos menciona que se deben tratar a los

servicios como componentes, esto nos permitirá desplegarlos independientemente y mantenerlos como una interface explícita, es decir mantener el principio de encapsulamiento a nivel de servicio.

- **Organizado sobre capacidades del negocio:** una tradicional organización es la división de equipos de trabajo acorde a su tecnología como: UI, server-side, especialistas en bases de datos, etc. Sin embargo, esta organización tiene un problema descrito en estas simples palabras: "*lógica por todos lados*". En una arquitectura de microservicios, los servicios se organizan acorde a su lógica de negocio como: almacenamiento, usuarios, productos, etc.
- **Productos, no proyectos:** muchos esfuerzos en el desarrollo de aplicaciones siguen el modelo de proyecto: donde el objetivo es entregar una pieza de software que sea considerada completa. Sin embargo, en la arquitectura de microservicios se prefiere la noción de que un equipo debe ser el dueño de un producto durante su tiempo de vida.
- **Puntos finales (endpoints) inteligentes y tuberías tontas:** las aplicaciones construidas mediante microservicios pretenden estar lo más desacopladas y cohesionadas como sea posible, es decir son dueñas de su propia lógica de dominio y actúan como filtros: reciben una petición, aplican la lógica adecuada y producen una respuesta. Su comunicación se basa en protocolos simples de red como peticiones HTTP o en mensajería liviana y asíncrona como *RabbitMQ* o *ZeroMQ*.
- **Gobierno descentralizado:** uno de los clásicos problemas de las aplicaciones monolíticas es la tendencia de estandarizar la tecnología de las plataformas; es decir, usar un lenguaje de programación y una base de datos común entre todas las funcionalidades del sistema. En la arquitectura de microservicios, cada servicio puede construirse de manera independiente, escogiendo la tecnología adecuada para cada funcionalidad.
- **Manejo de datos descentralizado:** mientras que las aplicaciones monolíticas tienden a unificar las decisiones de almacenamiento de datos; es decir, persistir los datos en una única base de datos. En una arquitectura de microservicios se descentraliza las decisiones del almacenamiento de datos donde cada servicio tiene la capacidad de administrar su propia base de datos, tanto en tecnología como en esquema, pudiendo llegar a usar varias tecnologías de persistencia de datos en un enfoque denominado "Persistencia políglota".
- **Automatización de infraestructuras:** una de las características base en arquitecturas de microservicios es la entrega continua y su precursor la integración continua. Los equipos desarrollan varios servicios que hacen uso de técnicas extensas de automatización de infraestructura. Uno de los ejemplos es el uso de Amazon Web Services (AWS) que contienen un conjunto de herramientas para automatizar y desplegar microservicios con facilidad.
- **Diseño para el fallo:** una de las características más prominentes de la arquitectura de microservicios es la capacidad y tolerancia al fallo. Cualquier servicio puede fallar debido a una indisponibilidad; sin embargo, el cliente debe responder a ello con la mayor gracia posible. A su vez, esta característica introduce mayor complejidad de manejo debido a que los

equipos reflexionan constantemente sobre cómo los fallos de los servicios afectan a la experiencia del usuario.

- **Diseño evolutivo:** los profesionales de los microservicios, por lo general, provienen de un entorno de diseño evolutivo y ven la descomposición de servicios como una herramienta más para permitir a los desarrolladores de aplicaciones controlar los cambios en su aplicación, sin ralentizar el cambio. Si se realiza el control de los cambios, con la actitud y las herramientas adecuadas, éstos pueden ser frecuentes, rápidos y bien controlados en el software.

2.1.3 Aplicación Monolítica vs Arquitectura de Microservicios

Una de las tareas comunes, para los equipos de desarrollo, es la elección de la arquitectura adecuada. Los equipos deben decidir entre enfoques monolíticos o aquellos orientados a microservicios.

Para ello, las principales diferencias a tomar en consideración entre ambos enfoques son (Kharenko, 2015):

Aplicación Monolítica	Arquitectura de Microservicios
Simple de desarrollar	La complejidad se descompone en un conjunto de servicios manejables y fáciles de desplegar
Simple de probar	Cada servicio puede ser desarrollado de manera independiente
Simple de desplegar	Los servicios son libres de escoger la mejor tecnología independientemente de los otros servicios
Simple de escalar de manera horizontal	Cada servicio puede ser desplegado independientemente
	Cada servicio puede escalar de manera independiente

Tabla 2: Beneficios de ambas Arquitecturas
Fuente: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59

De la misma manera, cada arquitectura añade complejidades y retos resumidos en el siguiente conjunto de inconvenientes (Kharenko, 2015):

Aplicación Monolítica	Arquitectura de Microservicios
Este enfoque conlleva una limitación en tamaño y complejidad	Añade complejidad al proyecto únicamente por el hecho de que los microservicios suponen un sistema distribuido
Las aplicaciones se tornan demasiado largas y complejas de entender completamente o realizar cambios	Contienen una base de datos particionada. Cualquier transacción requiere actualizar múltiples bases de datos para mantener la integridad

El tamaño de la aplicación puede volver lento el arranque de la misma	Probar microservicios en conjunto es mucho más complejo
El impacto en un cambio no puede ser entendido en totalidad por lo que conlleva una intensa labor de prueba	Es más difícil implementar cambios que afecten muchos servicios
El despliegue continuo se torna difícil	El despliegue de microservicios es igual una tarea compleja ya que conlleva muchos sistemas que hay que desplegarlos independientemente
Difíciles de escalar cuando los distintos módulos entran en conflicto	
Baja fiabilidad; es decir un fallo en cualquier módulo puede comprometer todo el proceso	
Suponen una barrera cuando se pretende adoptar una nueva tecnología ya que un cambio afectaría a toda la aplicación	

Tabla 3: Inconvenientes de ambas Arquitecturas
Fuente: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59

En la figura 4, se presenta un diagrama de una breve descripción de cada arquitectura.

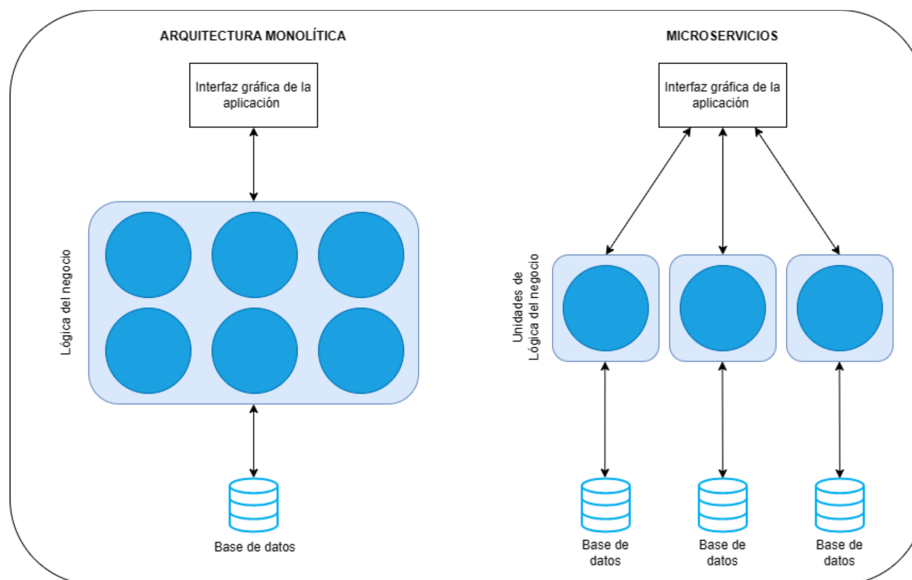


Figura 4: Diferencia entre Arquitectura Monolítica y Microservicios.
Fuente: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63

En este sentido, dependerá del equipo escoger el mejor enfoque para diseñar la arquitectura de software que más se adapte a sus necesidades.

2.2 Protocolo OAuth

Para entender la lógica detrás del protocolo OAuth y su consiguiente *framework* OAuth 2.0, es necesario abordar el concepto de delegación de identidad ya que toma un rol principal en la seguridad.

Imaginemos que eres el dueño de un API, pero no su directo consumidor; es decir, puede haber un tercero que quiera consumir dichos recursos. Compartir las credenciales no siempre es una buena práctica, ya que es otorgar un acceso completo sin restricciones ni límites que será muy difícil controlar. Por ello, se buscará usar un modelo de delegación de identidad; es decir, una forma de delegar la identidad y su autorización de un servicio a otro.

Una delegación de identidad tiene 3 roles principales (Siriwardena, 2014):

- **Delegador (delegator):** Es el dueño del recurso a acceder. Es usualmente llamado el dueño del recurso (*resource owner*).
- **Delegado (delegate):** el delegado es el tercero que quiere acceder al recurso en nombre del delegador.
- **Proveedor del servicio (service provider):** el proveedor del servicio alberga el recurso y valida que el delegado sea legítimo. Suele ser también denominado el servidor del recurso (*resource server*).

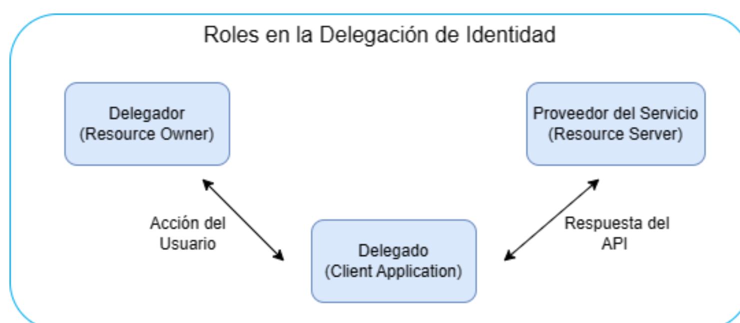


Figura 5: Roles de delegación de identidad.
Fuente: Elaboración propia

En este contexto, se desarrolló el protocolo OAuth 1.0 como una buena base para la delegación de identidad; sin embargo, dado que ha recibido múltiples críticas acorde a su usabilidad y extensibilidad, el *framework* OAuth 2.0 fue desarrollado en el año de 2012 (Siriwardena, 2014).

2.2.1 OAuth 1.0

OAuth 1.0 empezó alrededor de noviembre de 2006; sin embargo, para abril de 2007 se conformó un grupo de *Google* con un pequeño conjunto de implementadores con el propósito de escribir el protocolo abierto. Para Julio del mismo año escribió el primer borrador de una especificación inicial y dicho grupo fue abierto para que cualquiera pueda colaborar. Para el día 3 de octubre del 2007 el borrador final de OAuth Core 1.0 fue finalmente revelado (Hammer-Lahav, 2007).

OAuth es la estandarización y la sabiduría combinada de muchos protocolos bien establecidos en la industria. Es similar a otros protocolos (Google AuthSub, AOL OpenAuth, Yahoo BBAuth, Upcoming API, Flickr API, Amazon Web Services API, etc). Cada protocolo proporcionó un método propio para intercambiar credenciales de usuario por un token de acceso. OAuth se creó estudiando cuidadosamente cada uno de estos protocolos y extrayendo las mejores prácticas y los aspectos comunes que permitirán nuevas implementaciones, así

como una transición suave para que los servicios existentes soporten OAuth (Hammer-Lahav, 2007).

OAuth 1.0 utiliza tres partes en una transacción de delegación de identidad. El delegador, usualmente llamado usuario que asigna accesos a sus recursos a terceros. El delegado, también conocido como consumidor, que accede a un recurso en nombre del usuario. El proveedor de Servicio, más conocido como *resource provider*, que es la aplicación que alberga el recurso a acceder (Siriwardena, 2014).

El acceso se lo realiza mediante tokens en un flujo donde los mismos van siendo compartidos entre las partes. Estos tokens son generados por el proveedor del servicio y existen dos tipos distintos (OAuth Core, 2007):

- **Token de Solcitud (*request token*):** usado por el consumidor para solicitar al usuario la autorización al acceso del recurso protegido.
- **Token de acceso (*access token*):** usado por el consumidor para acceder al recurso protegido en nombre del usuario.

2.2.1.1 Flujo de autorización

El flujo de autorización básico de **OAuth 1.0** se describe en 3 pasos:

1. El consumidor obtiene un token de solicitud (*request token*) no autorizado.
2. El usuario autoriza el token de solicitud (*request token*).
3. El consumidor intercambia el token de solicitud (*request token*) por un token de acceso (*access token*).

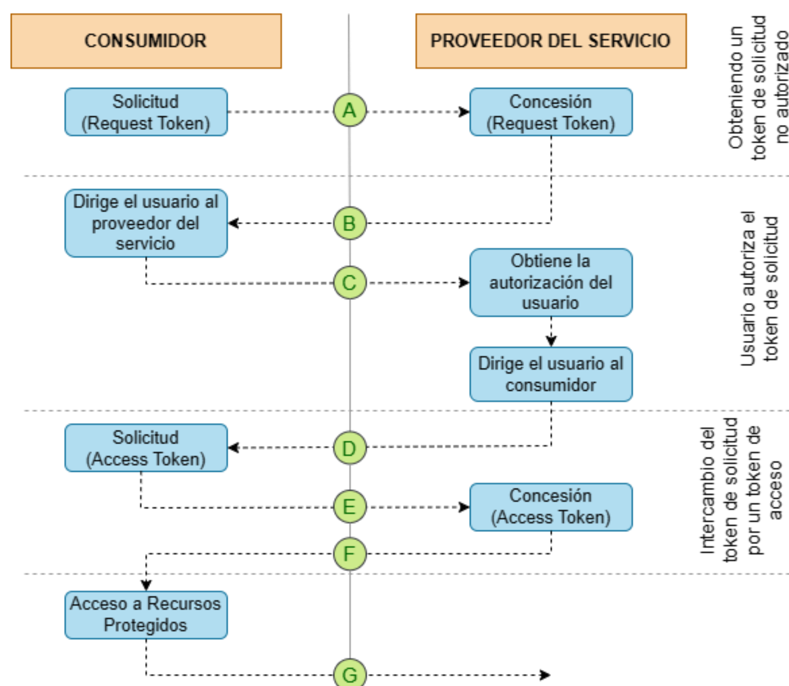


Figura 6: Flujo de Autorización de OAuth 1.0.
Fuente: https://oauth.net/core/1.0

2.2.2 OAuth 2.0

OAuth 1.0 representó un conjunto de problemas para los desarrolladores al momento de implementar el protocolo dado que contenía funciones criptográficas difíciles de implementar (Spasovski, 2013).

En este contexto, en noviembre de 2009, durante el *Internet Identity Workshop* (IIW), Dick Hardt de Microsoft, Brian Eaton de Google, y Allen Tom de Yahoo presentaron un nuevo borrador de una especificación para delegación de acceso. Éste fue llamado *Web Resources Authorization Profiles* (WRAP) el cual respondía a varias limitaciones encontradas en el protocolo OAuth 1.0. En el año de 2010, WRAP fue deprecado en favor del framework OAuth 2.0 que toma el mismo como base. (Siriwardena, 2014).

La mayor diferencia entre OAuth 1.0 y OAuth 2.0 es que OAuth 1.0 es un protocolo estándar para la delegación de identidad, mientras que OAuth 2.0 es un framework altamente extensible. OAuth es, de facto, un estándar para la seguridad de API y es ampliamente usado por muchas plataformas en la web (Siriwardena, 2014).

2.2.2.1 Componentes

OAuth 2.0 define 4 roles (RFC-6749, 2012):

- **Dueño del recurso (*resource owner*):** Es una entidad capaz de garantizar acceso a un recurso protegido. Cuando el dueño del recurso es una persona, es referido como un usuario final (*end-user*).
- **Servidor de recursos (*resource server*):** el servidor que aloja los recursos protegidos, capaz de aceptar y responder las llamadas al recurso protegido usando los tokens de acceso (*access tokens*).
- **Cliente (*client*):** una aplicación haciendo peticiones a los recursos protegidos en nombre del dueño del recurso (*resource owner*) y con su autorización.
- **Servidor de Autorización (*authorization server*):** el servidor que emite los tokens de acceso (*access tokens*) al cliente después de autenticar al dueño del recurso (*resource owner*) y obtener autorización.

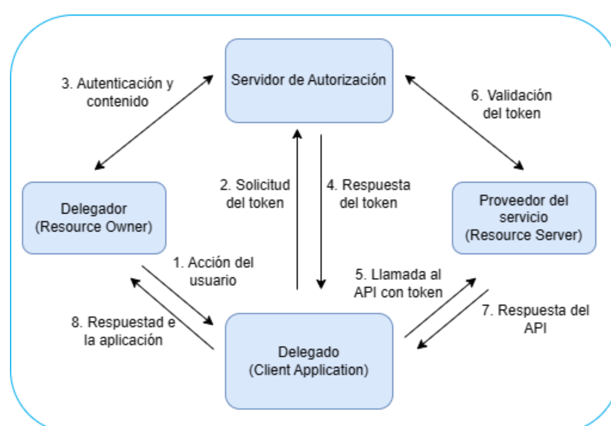


Figura 7: Componentes OAuth 2.0.
Fuente: https://www.zirous.com/2020/05/19/access-management-isnt-just-for-web-applications-its-for-apis-too/

2.2.2.2 Tipos de Concesión de Autorización

En resumen, es el cliente quién presenta una credencial al propietario de los recursos para acceder a los recursos protegidos. Para que este proceso se cumpla adecuadamente, es necesario pasar por un flujo de concesión de autorización. Este flujo puede ser realizado de cuatro formas distintas (RFC-6749, 2012):

Códigos de autorización (*authorization code*): El código de autorización se obtiene usando el Servidor de Autorización como intermediario entre el cliente y el propietario del recurso. En lugar de solicitar autorización directamente del propietario del recurso, el cliente dirige a un servidor de autorización para que el propietario del recurso obtenga la autorización y que retorne al cliente el código de autorización. El código de autorización nos dota de importantes beneficios de seguridad como la habilidad de autenticar el cliente, la transmisión del token de acceso (Access Token) directamente al cliente sin pasar a través del agente de usuario del propietario del recurso.

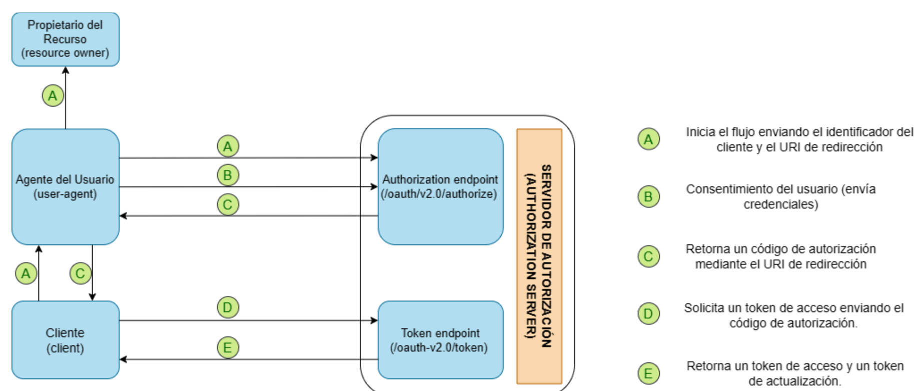


Figura 8: Flujo de código de Autorización OAuth 2.0
Fuente: RFC 6749

Implícito (*implicit*): es un flujo de código de autorización simplificado y optimizado para los clientes implementados en navegadores que usan el lenguaje JavaScript. En el flujo implícito, el cliente en lugar de obtener un código de autorización, obtiene un token de acceso directamente. En este proceso, el servidor de autorización no autentica al cliente.

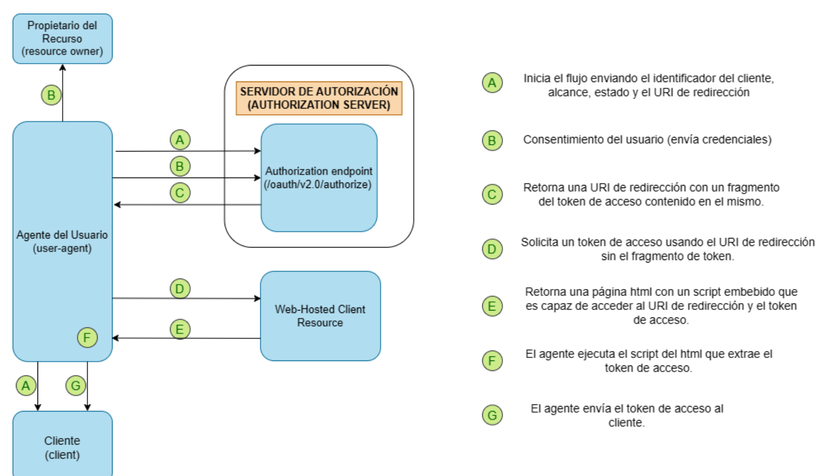


Figura 9: Flujo de Concesión Implícita.
Fuente: RFC 6749

Credenciales de contraseña del propietario del recurso (*resource owner password credentials*): las credenciales de contraseña del propietario de los recursos (por ejemplo: usuario/contraseña) pueden ser usados directamente como un método de autorización para obtener un token de acceso. Las credenciales solo deben ser usadas si hay un alto grado de confianza entre el propietario del recurso y el cliente. Exponer las credenciales sin un alto nivel de confianza puede ocasionar que la cuenta quede expuesta en su totalidad, ya que las credenciales otorgan acceso completo y carecen de límite o de control estricto.

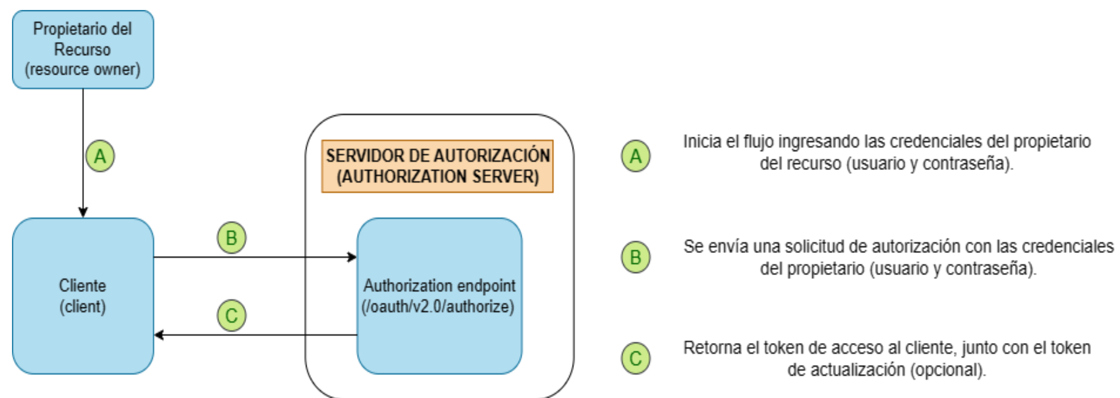


Figura 10: Flujo de concesión por credenciales de contraseña del propietario del recurso.
Fuente: RFC 6749

Credenciales del cliente (*client credentials*): Las credenciales del cliente pueden ser usadas como un método de autorización cuando el alcance de la autorización es limitado a los recursos protegidos bajo el control del cliente, o para recursos protegidos previamente negociados por el servidor de autorización.

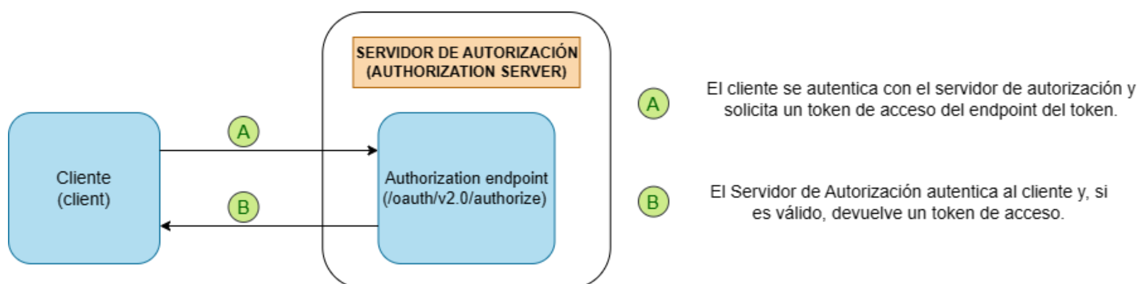


Figura 11: Flujo de concesión por credenciales del cliente.
Fuente: RFC 6749

2.2.2.3 Tokens

Un token es una cadena de texto usada por OAuth para que el cliente realice peticiones hacia otras partes. Sirve como una credencial que representa una autorización. Un token usualmente tiene un alcance y un tiempo de expiración. En OAuth 2.0 distinguimos dos tipos de tokens: de acceso (*Access Token*) y de actualización (*Refresh Token*) definidos como:

Tokens de Acceso (*Access Token*): los tokens de acceso son credenciales usadas para acceder a recursos protegidos. Es una cadena de texto que

representa una autorización concedida a un cliente. Un cliente sólo necesita un token de acceso válido y vigente para realizar peticiones al servidor de recursos.

Tokens de Actualización (*Refresh Token*): los tokens de actualización son credenciales usadas para obtener tokens de acceso. Cuando un token de acceso expira o queda inválido, los tokens de actualización son enviados por el cliente al servidor de autorización y son usados para obtener un nuevo token de acceso con un alcance similar al anterior. Un token de actualización es una cadena de texto que representa una autorización concedida al cliente por el propietario del recurso.

Así mismo, los tokens suelen tener uno o varios **ámbitos** (**scope**) que sirven para limitar el acceso a la cuenta de un usuario. El ámbito define el alcance y el límite que tiene un token sobre un nivel de acceso concedido (OAuth Working Group, 2012).

2.2.3 Flujo de autorización abstracto

Por otra parte, el flujo de autorización abstracto del protocolo OAuth 2.0 se compone de los siguientes pasos (RFC-6749, 2012):

1. El cliente solicita autorización al dueño del recurso (*Resource Owner*). La solicitud de autorización puede ser hecha directamente al dueño del recurso (*Resource Owner*) o mediante el servidor de autorización (*Authorization Server*) como intermediario.
2. El cliente recibe una concesión de autorización, que es una credencial que representa la autorización del propietario del recurso. El tipo de concesión de autorización depende del método utilizado por el cliente para solicitarla y de los tipos admitidos por el servidor de autorización.
3. El cliente solicita un token de acceso (*Access Token*) autenticándose con el servidor de autorización (*Authorization Server*) acorde a su tipo de concesión de autorización.
4. El servidor de autorización (*Authorization Server*) autentica al cliente y valida la concesión de la autorización; si es válido, emite un token de acceso (*Access Token*).
5. El cliente solicita el recurso protegido al servidor de recursos (*Resource Server*) y se autentica presentando su token de acceso (*Access Token*).
6. El servidor de recursos (*Resource Server*) valida el token de acceso (*Access Token*) y, si es válido, responde a la solicitud.

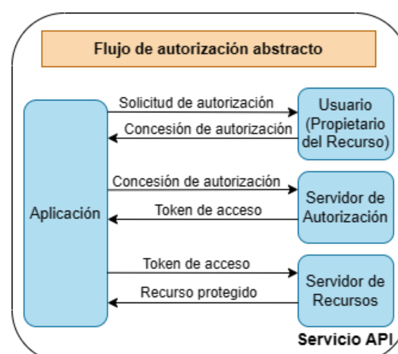


Figura 12: OAuth 2.0 - Flujo de Autorización básico
Fuente: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

2.2.4 Ventajas del uso de OAuth 2.0

Entre los principales beneficios de OAuth 2.0 se encuentra (Spasovski, 2013):

- **Seguridad para el API:** envío de tokens de acceso en lugar de credenciales codificadas.
- **Aplicaciones empresariales internas:** las aplicaciones guardan tokens de acceso en lugar de credenciales, lo cual es altamente recomendable en ambientes empresariales.
- **Integración del servicio y delegación de autorización:** El usuario puede conseguir acceso sin necesidad de tener las credenciales de la cuenta, en lugar de ello de le concede un token de acceso para que pueda acceder a un recurso protegido en nombre del propietario del recurso.
- **Identidad federada:** se genera una identidad digital de una persona y sus datos personales pueden ser compartidos entre varias aplicaciones.
- **Facilidad de monitorizar los servicios:** mediante el token de acceso se puede dar seguimiento y monitorización del usuario.

2.3 Documento RFC-6819

El documento RFC 6819 nos brinda un conjunto de consideraciones adicionales de seguridad para OAuth basado en una lista de amenazas comunes sobre OAuth 2.0 (RFC-6819, 2013).

2.3.1 Concepto

Acorde a (RFC-6819, 2013), este documento nos proporciona un modelo de amenazas completo para OAuth; así como algunas recomendaciones de seguridad para mitigarlas. Estas amenazas están divididas a lo largo de la estructura del protocolo, e incluyen cualquier ataque adicional en componentes y flujos OAuth.

2.3.2 Flujo de autorización OAuth 2.0 por códigos de autorización

El flujo de autorización del punto **2.2.3** es una abstracción de OAuth a varios tipos de flujos; sin embargo, nos centraremos ahora en un flujo completo de OAuth 2.0 utilizando el tipo de autorización por códigos de autorización. El objetivo de este flujo es mostrar el funcionamiento real de una implementación del protocolo OAuth 2.0. Esta figura servirá como punto de partida para entender las amenazas y consideraciones de seguridad presentados en el apartado **2.3.5**.

Para ello, se describen los siguientes pasos ilustrados en la **Figura 13**:

1. El usuario final ingresa una URL en el agente (navegador web).
2. El agente abre la URL en el servidor web.
3. El servidor web inicia el proceso de OAuth.
4. Envía al agente la URI para redirección hacia el servidor de autorización.

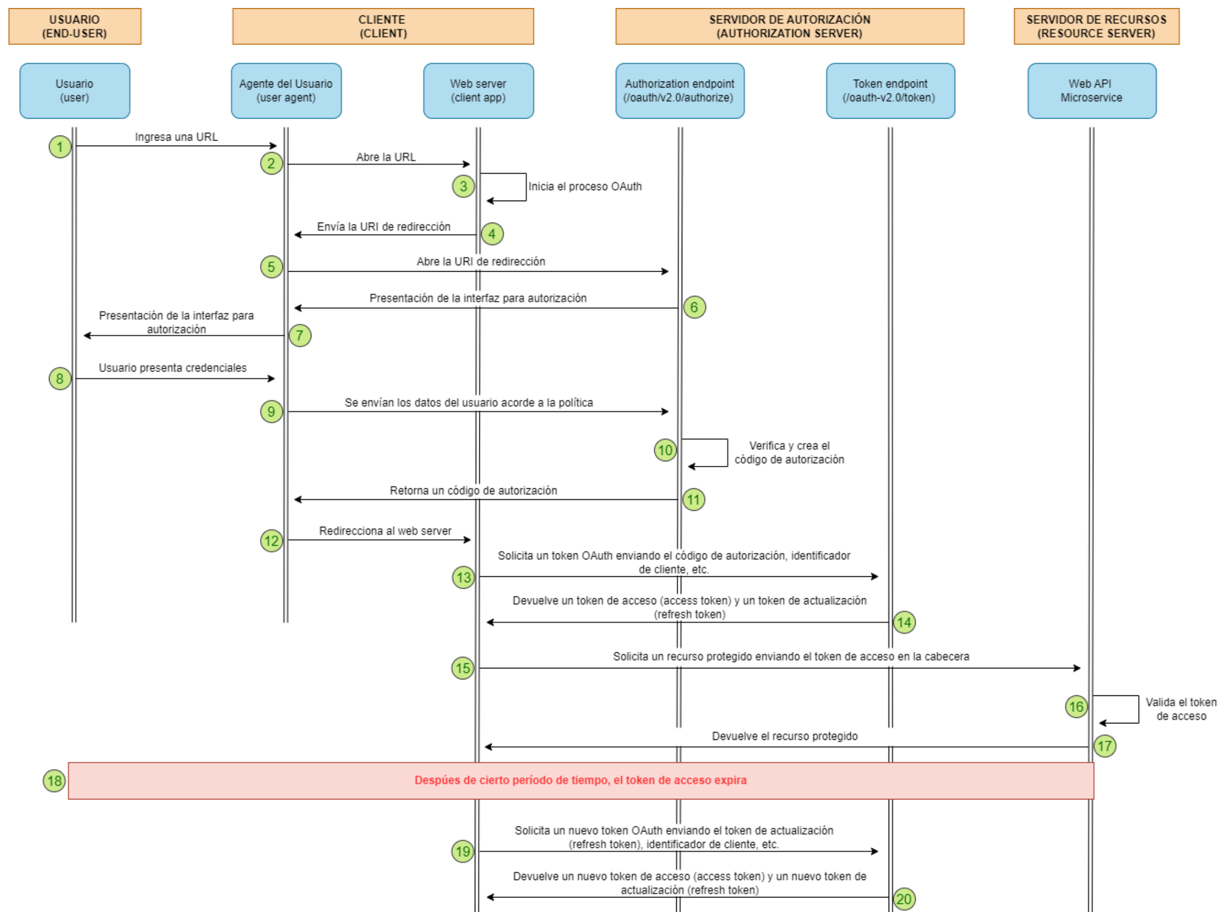


Figura 13: Flujo de Autorización Completo OAuth 2.0.
Fuentes: https://docs.oracle.com/cd/E50612_01/doc.11122/oauth_guide/content/oauth_flows.html
https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-auth-code-flow

5. El Agente abre la URI de redirección para iniciar el proceso de autorización en el servidor de autorización.
6. El Servidor de autorización solicita que se abra la URI para autorización al agente del cliente.
7. El agente presenta la interfaz al usuario final para que pueda ingresar sus credenciales.
8. El usuario ingresa sus credenciales en el agente del cliente.
9. El agente envía las credenciales del usuario acorde a las políticas al servidor de autorización mediante el endpoint de autorización.
10. El servidor de autorización verifica la validez del código de autorización.
11. El servidor de autorización, mediante el endpoint de autorización, retorna al agente un código de autorización.
12. El agente redirecciona este código al servidor web del cliente.
13. El cliente, mediante el servidor web solicita un token de acceso al servidor de autorización, mediante el endpoint de token, enviando el código de autorización e información opcional como el identificador del cliente, etc.
14. El servidor de autorización, mediante el endpoint de token, retorna un token de acceso y un token de actualización al servidor web del cliente.
15. El cliente solicita un recurso protegido al servidor de recursos, enviando una petición cuya cabecera contenga el token de acceso.
16. El servidor de recursos valida que el token de acceso sea válido y vigente.
17. El servidor de recursos retorna el recurso protegido al cliente.

18. Tiempo después de cierto periodo definido por la expiración del token, el cliente nuevamente solicita acceder al recurso; sin embargo, su token de acceso fue rechazado.
19. El cliente solicita un nuevo token de acceso al servidor de autorización, mediante el endpoint de acceso, enviando una petición cuya cabecera contenga el token de actualización.
20. El servidor de autorización, mediante el endpoint de token, devuelve un nuevo token de acceso y uno de actualización.

El proceso se repite nuevamente desde el punto 15.

2.3.3 Características de los despliegues

El documento de consideraciones de seguridad tiene en cuenta únicamente los casos que cumplen un despliegue particular soportado por el RFC 6749 (OAuth 2.0). Estos despliegues deben de tener estas características (RFC-6819, 2013):

- Las URL del servidor de recursos (*Resource Server*) son estáticas y conocidas en el momento del desarrollo; las URL del servidor de autorización (*Authorization Server*) pueden ser estáticas o descubiertas.
- Los valores de los ámbitos del token (por ejemplo, las URI y sus métodos) son bien conocidos en el momento del desarrollo. Como se abordó apartados anteriores, el ámbito controla el alcance de acceso que posee un token sobre la cuenta del usuario.
- El registro de clientes esta fuera del alcance de la especificación actual del núcleo. Por lo tanto, este documento asume una amplia variedad de opciones, desde el registro estático durante el tiempo de desarrollo hasta el registro dinámico en tiempo de ejecución.

Los siguientes supuestos son considerados fuera del rango (RFC-6819, 2013):

- La comunicación entre el servidor de autorización y el servidor de recursos (*Authorization Server – Resource Server*).
- Los formatos del token.
- A excepción del tipo de concesión de credenciales del propietario del recurso, el mecanismo utilizado por los servidores de autorización para autenticar el usuario.
- El mecanismo por el cual un usuario obtuvo una confirmación de acceso y cualquier ataque montado como resultado de una confirmación falsa.
- Clientes no vinculados con un despliegue específico. Estos clientes no pueden ser registrados de antemano con un despliegue concreto y deberían descubrir dinámicamente las URL relevantes para el protocolo.

2.3.4 Supuestos de los ataques

De la misma forma, el documento RFC 6819 establece un conjunto de conceptos y supuestos a tomar en cuenta para empezar a describir las debilidades del protocolo y los posibles ataques que buscan aprovecharlas.

Los siguientes supuestos son relativos al atacante y los recursos disponibles por el atacante (RFC-6819, 2013). Se asume que:

- Un atacante tiene acceso completo a la red entre el cliente-servidor de autorización, y el cliente-servidor de recursos respectivamente (Authorization Server – Client & Client – Resource Server).
- Un atacante tiene recursos ilimitados para montar un ataque.
- Dos de tres partes involucradas en el protocolo OAuth pueden colaborar para montar un ataque contra la tercera parte. Por ejemplo, el cliente y el servidor de autorización colaboran para atacar a un usuario para conseguir acceso a un recurso.

2.3.5 Modelo de Amenazas y Recomendaciones

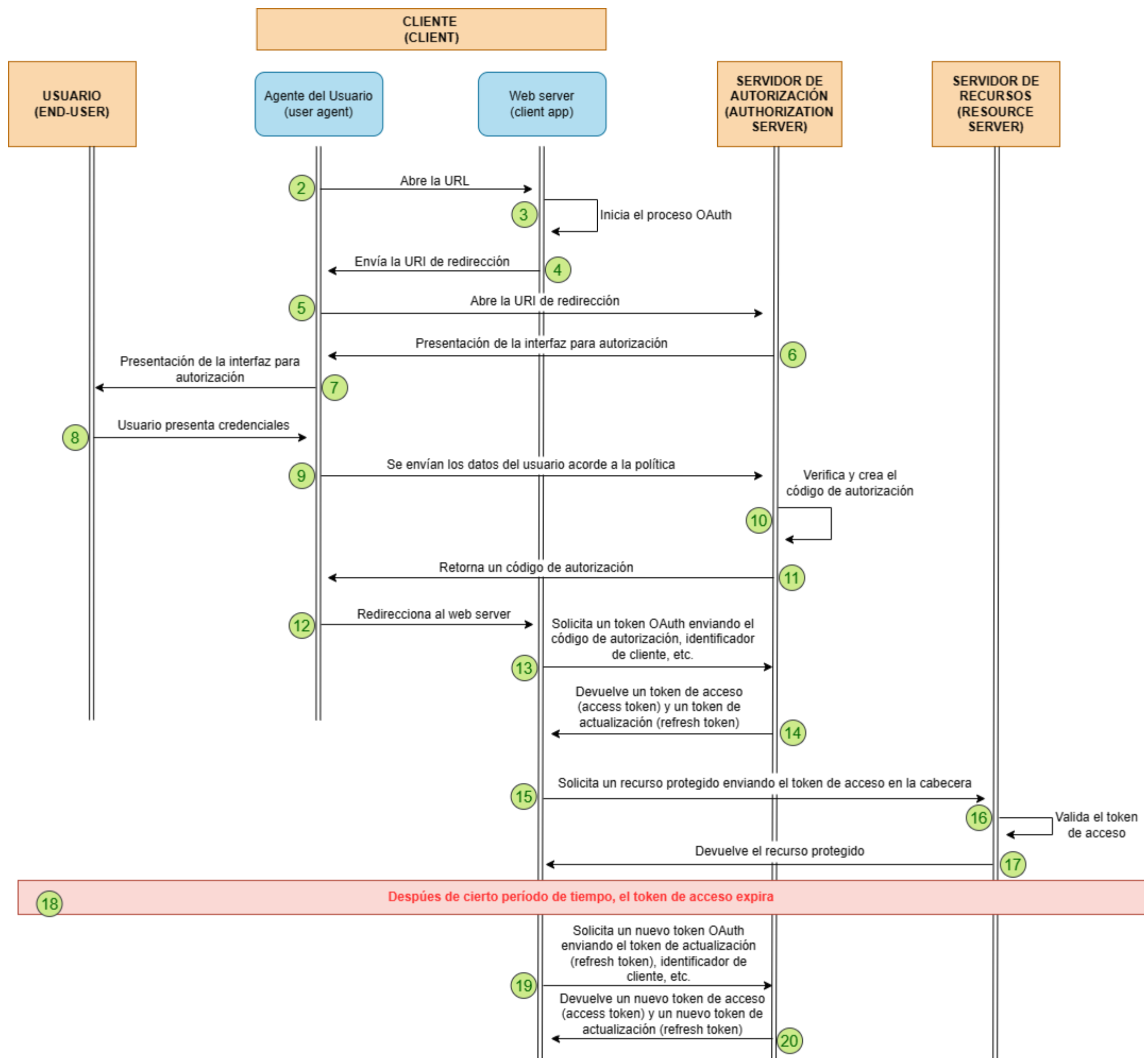
Las amenazas que enfrenta OAuth 2.0 están clasificadas por ataques dirigidos a un componente de OAuth (Client, Authorization Server y Resource Server) y agrupados por flujo (por ejemplo: obtener el token o el acceso a los recursos protegidos) acorde al diagrama de secuencia de la **Figura 13**.

2.3.5.1 Dirigidas al Cliente

Un cliente es una aplicación que hace peticiones a recursos protegidos por el propietario de recursos con su autorización. Existen varios tipos de clientes como aplicaciones nativas, clientes web, basados en agentes del usuario, etc. Sin embargo, los siguientes datos son guardados por un cliente de manera general (RFC-6819, 2013):

- Identificador del cliente (*client_id*) y el secreto o la credencial correspondiente (*client_secret*).
- Uno o más tokens de actualización (persistentes) y tokens de acceso (transitorios) por usuario final o contexto de seguridad o delegación.
- Certificados (HTTPS) de la Autoridad de certificación (CA) de confianza.
- Valores para el proceso de autorización: *redirect_uri*, *authorization code*.

En su flujo habitual, un cliente tiene acceso y guarda los datos antes descritos. Si nos fijamos en un flujo completo de OAuth como el presentado en la **Figura 13**, observaremos un conjunto de procesos que el cliente ejecuta, y en cada proceso va capturando estos datos, resumidos en la siguiente figura:



PROCESOS GENERADOS EN EL FLUJO	
CLIENTE (CLIENT)	
Agente del Usuario (user agent)	Web server (client app)
2	3
5	4
7	13
9	15
12	19

DATOS USADOS POR PROCESO	
PROCESO	DATO
4	URI del Servidor de autorización
9	Credenciales del usuario
13	Código de autorización, identificador del cliente, etc
15	Token de acceso, identificador del cliente, etc.
19	Token de actualización, identificador del cliente, etc.

Figura 14: Flujo y datos usados por el cliente en un entorno OAuth 2.0.
Fuente: Elaboración propia.

En este contexto, los ataques y recomendaciones enfocadas en el cliente de OAuth 2.0 se enfocan en los puntos y datos descritos anteriormente, y son (RFC-6819, 2013):

AMENAZA	ATAQUE	DESCRIPCIÓN	CONSECUENCIAS	RECOMENDACIONES
Obteniendo los secretos del cliente	Obtener el secreto del código fuente o el binario	Un atacante puede intentar obtener acceso al secreto de un cliente para reproducir su token de autorización o para actuar en una instancia en nombre del cliente. Este secreto puede ser obtenido del código fuente si es *open source* o del binario.	• Un atacante puede obtener la autenticación del cliente para acceder al servidor de autorización. • Los tokens de actualización o los códigos de autorización robados pueden ser reproducidos.	• No compartir secretos de clientes públicos o clientes con políticas inapropiadas. • Solicitar consentimiento del usuario para clientes públicos. • Usar secretos de cliente para despliegues específicos. • Revocar secretos de los clientes.
	Obtener el secreto de un despliegue específico	Un atacante puede intentar obtener acceso al secreto de un cliente para reproducir su token de autorización o para actuar en una instancia en nombre del cliente. Este token puede ser obtenido de una instalación del cliente, de un sitio web o de un dispositivo móvil.		• Aplicar medidas de protección de web server. • Guardar secretos en almacenamiento local seguro en aplicaciones nativas. • Revocar los secretos de los clientes
Obteniendo los tokens de actualización	Obtener el token de actualización de una aplicación web	Un atacante puede obtener el token de actualización de una aplicación web evadiendo los controles de seguridad del servidor web.	• Exposición de todos los tokens de actualización del sitio al atacante.	• El servidor de autorización debe validar el identificador del cliente asociado a un token de actualización en cada solicitud de actualización. • Limitar el alcance del token. • Revocar los secretos del cliente. • Los tokens de actualización pueden ser reemplazados automáticamente para detectar aquellos que no estén autorizados. • Medidas estándar de seguridad de servidores web. • Usar medidas de autenticación fuertes.
	Obtener el token de actualización de clientes nativos	Un atacante puede tratar de obtener el archivo del sistema de un dispositivo y leer su token de actualización.	• Exposición del token de actualización del dispositivo.	• Guardar secretos guardados en almacenamientos seguros. • Utilizar bloqueos de seguridad del dispositivo.
	Robo del dispositivo	El dispositivo puede ser robado y el atacante tener acceso a todas las aplicaciones usando la identidad del usuario.	• Acceso a todas las aplicaciones del dispositivo usando la identidad del usuario.	• Utilizar un bloqueo seguro del dispositivo. • Revocar los tokens de los dispositivos afectados.
	Clonación de dispositivo	Todos los datos y las aplicaciones son copiadas de un dispositivo a otro.		• Utilizar un bloqueo seguro del dispositivo. • Revocar los tokens de los dispositivos afectados. • Rotación de tokens de actualización. • Combinar las solicitudes de actualización de token con un identificador del dispositivo.
Obtener los tokens de acceso	Obtener los tokens de acceso	Los tokens de acceso pueden ser robados del dispositivo si la aplicación los guarda en un almacenamiento accesible por otras aplicaciones.	• El atacante puede acceder a todos los recursos asociados con el token y su alcance.	• Mantener los tokens de acceso en memorias temporales con accesos limitados. • Limitar el alcance del token. • Mantener los tokens de acceso en memorias privadas. • Mantener corto el tiempo de vida de un token de acceso.

Obtener las Credenciales de usuario	Credenciales de usuario final robadas mediante un navegador web comprometido	Una aplicación maliciosa puede robar las contraseñas de los usuarios finales usando un navegador embebido en el proceso de autorización del usuario final, o presentando su propia interfaz en lugar de permitir un navegador confiable.	• Si la aplicación del cliente o la comunicación está comprometida, las credenciales de acceso pueden ser capturadas.	• El flujo de OAuth permite que las aplicaciones del cliente nunca necesiten contraseñas. • Las aplicaciones cliente pueden ser validadas antes de ser publicadas en las tiendas para acceso de los usuarios. • Los desarrolladores de los clientes no deben escribir clientes que guarden información directa de los usuarios. En su lugar, debe delegarse esta tarea a un componente seguro.
Redirecciones abiertas en los clientes	Redirecciones abiertas en los clientes	Se expone un *endpoint* que usa un parámetro para redirigir automáticamente una petición hacia una dirección no validada.	• Un atacante puede obtener acceso a un token de acceso o a un código de autorización.	• Solicitar a los clientes el registro de una dirección URI completa que servirá para una autorización segura.

Tabla 4: Modelo de amenazas y recomendaciones para el Cliente OAuth2.0
Fuente: RFC 6819

2.3.5.2 Endpoint de Autorización

El endpoint de Autorización es usado para interactuar con el propietario del recurso y obtener una autorización. El servidor de autorización primero debe verificar la identidad del propietario del recurso. Las solicitudes a este endpoint conllevan la transmisión de las credenciales del usuario y dan como resultado la autenticación del mismo. Este endpoint debe soportar los métodos HTTP de GET y POST al mismo tiempo (RFC-6749, 2012).

De la misma manera, el servidor de autorización, guardará los siguientes datos (RFC-6819, 2013):

- Nombres de usuario y contraseñas
- Identificadores de cliente y secretos
- Tokens de actualización de clientes específicos
- Tokens de acceso de clientes específicos
- Certificado o llave HTTPS
- Valores para el proceso de autorización: *redirect_uri*, *authorization code*.

Acorde al flujo completo de OAuth 2.0 presentado en la **Figura 13**, observamos un conjunto de procesos que el servidor de autorización mediante su endpoint de autorización ejecuta, acorde a la siguiente figura:

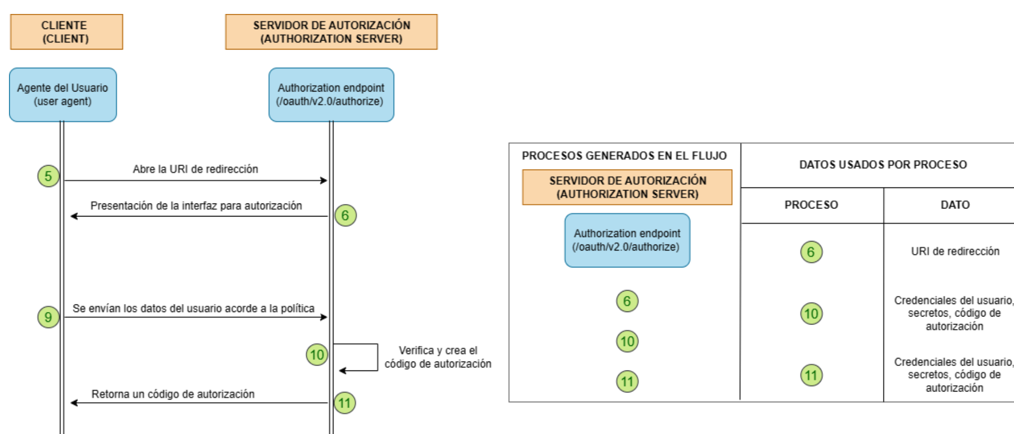


Figura 15: Datos usados por el endpoint de autorización en un flujo OAuth 2.0.
Fuente: Elaboración propia.

En este contexto, los ataques y recomendaciones enfocados en el endpoint de autorización del servidor de autorización de OAuth 2.0 se centran en los puntos y datos antes descritos y son (RFC-6819, 2013):

ATAQUE	DESCRIPCIÓN	CONSECUENCIAS	RECOMENDACIONES
Falsificación del Servidor de Autorización	Un atacante puede interceptar las solicitudes del cliente y retornar respuestas incorrectas. Estas respuestas incorrectas pueden llevar a los usuarios a colocar sus contraseñas en sitios web inseguros.	• Los atacantes pueden robar las credenciales de los usuarios.	• Encriptar la comunicación con el Servidor de Autorización mediante certificados SSL/TLS. • Capacitar a los usuarios sobre la naturaleza y los riesgos de los ataques de phishing.
El usuario concede involuntariamente demasiados privilegios en el acceso.	Tras obtener una autorización de parte del usuario final, el mismo no es consciente del nivel de acceso otorgado a un cliente, por lo que podría acceder a recursos para los cuales no debería tener permiso.	• Un cliente puede acceder a recursos a los cuales no debería tener permiso.	• El servidor de autorización debe explicar claramente al usuario final acerca del proceso de autorización, el nivel de acceso y el tiempo de duración del mismo. • Definición de políticas del cliente y configuraciones específicas, con menos acceso para clientes públicos. • El servidor de autorización debe permitir varios niveles de acceso basado en el tipo de autorización.
Clientes maliciosos que consiguen autorización mediante el fraude.	Los servidores de autenticación podrían procesar automáticamente las solicitudes de los clientes ya autorizados. Es decir, un cliente intenta tener autorización; sin embargo, el servidor detecta que ya existe una autorización previa, por lo que redirige automáticamente al cliente sin informar al usuario final.	• Un cliente malicioso puede obtener el código de autorización en lugar de un cliente legítimo.	• Los Servidores de Autorización no deben procesar automáticamente las peticiones de clientes públicos, salvo que esté pre registrado y validado su URI de redirección. • Los Servidores de Autorización pueden mitigar estos riesgos limitando el alcance de un token de acceso.
Redirección Abierta	Un atacante puede usar el endpoint de autorización y el parámetro que contiene la URI de redirección para usar el servidor de autorización como un punto de redirección.	• Un atacante puede usar la confianza del usuario en el Servidor de Autorización, para lanzar un ataque de phishing.	• Requerir a los clientes que se registre una dirección URI completa para redirección. • No redirigir a ninguna URI si la misma no ha sido verificada acorde al identificador del cliente.

Tabla 5: Modelo de amenazas y recomendaciones para el endpoint de autorización OAuth2.0
Fuente: RFC 6819

2.3.5.3 Endpoint de Token

El endpoint de token es usado por el cliente para obtener un token de acceso al presentar su credencial acorde al tipo de concesión de autorización o su token de actualización. El endpoint del token se usa con cada tipo de concesión de autorización; con excepción del tipo de concesión implícito. Las solicitudes a este endpoint conllevan la transmisión de las credenciales del usuario. Este endpoint debe soportar el método HTTP de POST (RFC-6749, 2012).

De la misma manera, el servidor de autorización, guardará los siguientes datos (RFC-6819, 2013):

- Nombres de usuario y contraseñas
- Identificadores de cliente y secretos
- Tokens de actualización de clientes específicos
- Tokens de acceso de clientes específicos
- Certificado o llave HTTPS

- Valores para el proceso de autorización: *redirect_uri*, *authorization code*.

Acorde al flujo completo de OAuth 2.0 presentado en la **Figura 13**, observamos un conjunto de procesos que el servidor de autorización mediante su endpoint de token ejecuta, acorde a la siguiente figura:

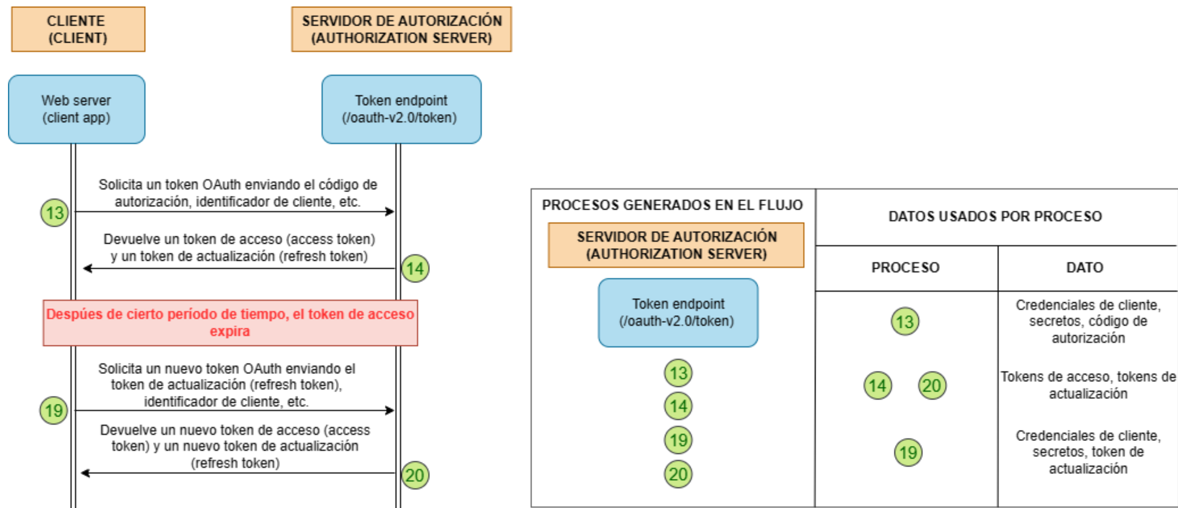


Figura 16: Flujo y datos usados por el endpoint de token en un flujo OAuth 2.0.
Fuente: Elaboración propia.

En este contexto, los ataques y recomendaciones enfocadas en el endpoint de token del servidor de autorización de OAuth 2.0 se centran en los puntos y datos antes descritos y son (RFC-6819, 2013):

ATAQUE	DESCRIPCIÓN	CONSECUENCIAS	RECOMENDACIONES
Captura de tokens de acceso.	Los atacantes pueden estar a la escucha de las comunicaciones en el tránsito del servidor de autorización hacia el cliente para capturar los tokens de acceso.	• Un atacante puede ser capaz de acceder a todos los recursos y con los alcances que tenga el token de acceso.	• Los Servidores de Autorización deben asegurar que todas las transmisiones sean encriptadas usando TLS. • Si la confidencialidad no puede ser garantizada, se puede reducir el alcance y el tiempo de expiración de los tokens de acceso.
Obteniendo los tokens de acceso de la base de datos del Servidor de Autorización	Si el Servidor de Autorización guarda todos los tokens de acceso en una base de datos, un atacante podría obtener acceso a la base de daos ejecutando un ataque de inyección SQL.	• Divulgación de todos los tokens de acceso.	• Asegurarse que el servidor esté usando el mínimo de privilegios de la base de datos posible. • Evitar *inputs* que se concatenen con código SQL de manera dinámica. • Si se usa SQL dinámico, parametrizar las consultas a la base usando *bind arguments*. • Sanear los datos en el momento del ingreso.
Revelación de las credenciales del Cliente durante la transmisión	Un atacante puede intervenir en la transmisión de las credenciales del cliente durante la comunicación server - cliente en el proceso de autenticación o durante el proceso de solicitud de tokens OAuth.	• Revelación de una credencial de cliente que permita la suplantación de identidad.	• El proceso de transmisión de las credenciales del cliente debe ser protegidos por encriptación tales como TLS. • Usar una alternativa de transmisión que no requiera enviar credenciales en texto plano, tales como hash, etc.

Obteniendo los secretos del cliente de la base de datos del Servidor de Autorización	Un atacante puede obtener un secreto de cliente válido de una base de datos del Servidor de Autorización ejecutando ataques de inyección SQL.	• Revelación de todos los secretos del cliente que le permite a un atacante ejecutar consultas de clientes legítimos.	• Asegurarse que el servidor esté usando el mínimo de privilegios de la base de datos posible. • Evitar *inputs* que se concatenen con código SQL de manera dinámica. • Si se usa SQL dinámico, parametrizar las consultas a la base usando *bind arguments*. • Sanear los datos en el momento del ingreso. • Asegurar un manejo apropiado del acceso a la base de datos.
Adivinando de los secretos del cliente	Un atacante puede tratar de adivinar los secretos del cliente.	• Revelación del secreto de un cliente. Permite a un atacante ejecutar consultas como un cliente legítimo.	• Usar una alta entropía en el secreto del cliente. Por lo general el valor de un token debe ser mayor o igual a 128 bits construido con una fuerte generación aleatoria. • Bloquear la cuenta después de unos intentos de acceso. • Usar una forma alternativa de autorización que eliminen el uso de secretos del cliente.

Tabla 6: Modelo de amenazas y recomendaciones para el endpoint de token OAuth2.0
Fuente: RFC 6819

Una vez analizados estos componentes, se describen unos ataques dirigidos contra los flujos usados en el proceso completo de OAuth descrito en la **Figura 13**.

2.3.5.4 Flujo: Obteniendo Autorización

El flujo para obtener una autorización se caracteriza por ser una parte del flujo completo de OAuth 2.0 que es utilizado para obtener los tokens de acceso. Cada flujo es caracterizado por los tipos de respuesta, o los tipos de concesión de autorización del usuario final y el endpoint del token, respectivamente (RFC-6819, 2013).

Dado que existen varios tipos de concesión de autorización (ver **2.2.2.2**), vamos a dividir en los posibles ataques acorde a cada tipo.

2.3.5.4.1 Concesión: Código de Autorización

El código de autorización es un tipo de concesión de OAuth 2.0 usado para obtener el token de acceso y actualización y es optimizado para clientes confidenciales. Debido a que este tipo de concesión es basado en redirección, el cliente debe ser capaz de interactuar con el agente de usuario del propietario del recurso (típicamente un navegador web) y capaz de recibir peticiones provenientes del servidor de autorización (RFC-6749, 2012).

El flujo de autorización de este tipo de concesión puede verse reflejado en la **Figura 8** y en el diagrama de secuencia de la siguiente figura:

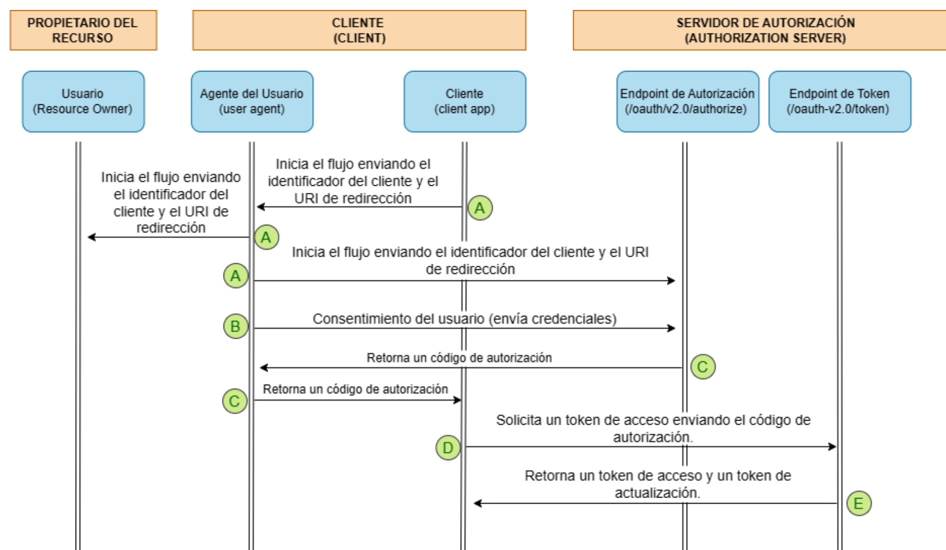


Figura 17: Diagrama de concesión por código de Autorización.
Fuente: RFC-6749

Los ataques y recomendaciones enfocadas en el flujo planteado por el tipo de concesión de autorización denominado código de autorización de OAuth 2.0 son (RFC-6819, 2013):

ATAQUE	DESCRIPCIÓN	CONSECUENCIAS	RECOMENDACIONES
Filtración de códigos de autorización	Los códigos de autorización son enviados al navegador web que pueden filtrarlos inintencionalmente a sitios web no seguros usando varias técnicas como: las cabeceras "referrer", mediante una solicitud de envío de logs, redirección a sitios web maliciosos, captura del historial de navegación.	• Captura de código de autorización para ejecutar solicitudes como un cliente legítimo.	• La comunicación entre Cliente - Servidor de Autorización debe ser seguro usando mecanismos de seguridad en la capa de transporte como TLS. • Un Servidor de Autorización debe incluir a cada código de autorización, el identificador del cliente que inició con el proceso de autorización. • Tiempo de expiración del código de autorización más cortos. • Un servidor de Autorización debe restringir el número de solicitudes para un código de autorización. Es decir, si recibe más de una solicitud de autorización para un cierto token, debe revocar el acceso. • Reducir el alcance y el tiempo de expiración de tokens de acceso. • El cliente debe recargar la página web que se abrió por redirección para limpiar el cache del navegador.
Obteniendo los códigos de autorización de la base de datos del servidor de Autorización	Un atacante puede obtener los códigos de autorización de la base de datos del servidor de autorización obteniendo acceso a la base de datos o realizando un ataque de inyección SQL.	• Divulgación de todos los códigos de autorización para realizar peticiones como clientes legítimos.	• Aplicando buenas prácticas para el almacenamiento de credenciales. • Usar el mínimo de privilegios a la base de datos. • Usar buenas prácticas para evitar los ataques de inyección de SQL. • Almacenamiento de credenciales usando códigos hash.

Adivinando el código de autorización	Un atacante puede adivinar el código de autorización de un cliente.	• Divulgación de un token de acceso y probablemente también su token de actualización. El atacante lo usará para realizar peticiones de un cliente legítimo.	• Usar una alta entropía en el secreto del cliente. Por lo general el valor de un token debe ser mayor o igual a 128 bits construido con una fuerte generación aleatoria. • Tokens deben ser firmados para detectar cualquier intento de modificación o falsificación. • Incluir el identificador del cliente con el código de autorización. • Incrustar el código de autorización en la URI de redirección. • Usar tiempos de expiración más cortos para los tokens de acceso.
Clientes maliciosos obteniendo autorización.	Un cliente malicioso puede pretender ser un cliente legítimo para obtener una autorización de acceso. El cliente malicioso puede intentar simular un consentimiento real del usuario.	• Un cliente malicioso que consigue una autorización y ejecuta acciones como un cliente legítimo.	• Un servidor de autorización puede emitir por separado los identificadores del cliente y los secretos de cada instalación del cliente. • El servidor de autorización debe validar el URI de redirección de cada cliente, comparándolo con el URI de redirección pre registrado. • Después de la autenticación del usuario final, el servidor de autorización debe solicitar consentimiento. • El servidor de autorización no debería autorizar o volver a autorizar automáticamente clientes que no puede validar. • Uso de captchas o similares. • El servidor de autorización debe limitar el alcance de los tokens.
Phishing del código de autorización	Una parte hostil puede hacerse pasar por cliente y obtener acceso al código de autorización. Esto se puede conseguir al suplantar el DNS o ARP.	• Afecta a las aplicaciones web y puede causar la divulgación de códigos de autorización, y potencialmente, divulgar los tokens de acceso y actualización.	• El URI de redirección del cliente debe apuntar a un endpoint protegido con el protocolo HTTPS, y el navegador web debe ser usado para validar esta URI de redirección mediante el dominio registrado en el cliente. • El código de autorización debe ser incrustado junto con el identificador del cliente en un token para ser validado.
Suplantación de sesiones del usuario.	Una parte hostil puede hacerse pasar por el sitio el cliente y hacerse pasar por una sesión de este cliente. Esto se puede conseguir al suplantar el DNS o ARP.	• Un atacante puede obtener acceso a recursos protegidos y/o modificarlos.	• El URI de redirección del cliente debe apuntar a un endpoint protegido con el protocolo HTTPS, y el navegador web debe ser usado para validar esta URI de redirección mediante el dominio registrado en el cliente.
Fuga del código de autorización mediante un cliente falso	El atacante abusa de una aplicación cliente existente y la combina con su propio sitio web cliente falso. El atacante depende de que la víctima espere que la aplicación cliente solicite acceso a un determinado servidor de recursos. La víctima, al ver sólo una solicitud normal de una aplicación esperada, aprueba la solicitud. A continuación, el atacante utiliza la autorización de la víctima para acceder a información autorizada por ésta sin saberlo.	• El atacante consigue acceso a los recursos de la víctima asociados con su cuenta en su lado de cliente. .	• El servidor de autorización debe incrustar cada código de autorización con la URI de redirección actual usada en el proceso de autorización. Esta dirección debe ser validada en cada petición junto con la pre registrada. • Desplegar identificadores de cliente y secretos específicos para aplicaciones nativas. Estos deben ser validados para encontrar ataques. • El cliente debe usar otros flujos de autorización que no sean vulnerables a este ataque.
Ataque *Cross-site request forgery* (CSRF) contra la URI de redirección	Un CSRF es un ataque de web donde las solicitudes HTTP son transmitidas desde un usuario que es de confianza para el sitio web. El atacante puede obtener el código de autorización.		• El cliente debe usar un parámetro "*state*" para enviar al servidor de autorización un valor que vincule la solicitud con el estado de autenticación del agente. Este valor permite al cliente la validez de la solicitud comparando el valor de ese parámetro. • Los desarrolladores del cliente y los usuarios finales deben ser capacitados para no abrir URLs no confiables.

Ataque de ClickJacking contra la autorización	Con el ClickJacking, un sitio malicioso puede cargar un sitio objetivo dentro de un iFrame transparente y superpuesto a un conjunto de botones falsos que se construyen cuidadosamente para colocarlos justo debajo de los botones importantes del sitio web destino. Cuando un usuario da click en un botón visible, en realidad puede estar autorizando a la página oculta.	• Un atacante puede robar las credenciales de autenticación del usuario y acceder a sus recursos.	• Para navegadores web más recientes, se puede evitar el uso de *iFrames* durante la autorización en el lado del servidor utilizando la cabecera *X-FRAME-OPTIONS*. Esta cabecera tiene dos parámetros "*DENY*" y "*SAME ORIGIN*" que pueden bloquear el uso de *iFrames*. • Para navegadores antiguos, existe una técnica llamada *JavaScript frame-busting* que puede ser usada para evitar el uso de *iFrames*, pero puede no ser compatible con todos los navegadores.
Suplantación del Propietario del Recurso	Un cliente malicioso podría incrustar HTML oculto para interpretar los formularios HTML enviados al servidor de autorización y enviar automáticamente las peticiones del formulario correspondiente. El atacante debe poder ejecutar el proceso de autorización en una sesión ya autorizada.	• Un atacante puede obtener acceso a los recursos de la víctima sin su aprobación.	• Para prevenir este ataque, el servidor de autorización debe forzar una interacción basado en ingreso de valores no predecibles como parte de la aprobación del usuario. Para ello podría usar: ○ Combinar la autenticación por contraseña y el consentimiento del usuario en un mismo formulario. ○ Usar CAPTCHAs. ○ Usar secretos de un solo uso fuera de la banda del propietario del recurso. Por ejemplo: mensajes de texto.
Ataques DoS que agoten los recursos	Si un servidor de autorización no limita el número de códigos de autorización o tokens de acceso por usuario, un atacante podría agotar la pila de códigos de acceso redirigiendo repetidamente peticiones de autorización.	• Un atacante podría comprometer la accesibilidad del servidor de autenticación y acceder a recursos protegidos.	• El servidor de autorización debe limitar el número de tokens de acceso concedidos por usuario. • El servidor de autorización debe incluir una cantidad no trivial de entropía en los códigos de autorización.
Ataques DoS usando códigos de autorización fabricados.	Un atacante que posea una red de *bots* puede localizar las *URIs* de redirección de los clientes que escuchan en HTTP, acceder a ellos con "códigos" de autorización aleatorios y hace que un gran número de conexiones HTTPS se concentren en el servidor de autorización. Esto puede dar lugar a un ataque de denegación de servicio en el servidor de autorización.	• Problemas en la velocidad de las conexiones al servidor. • Utilización asimétrica de los recursos del servidor.	• Utilización del parámetro "*state*" en los tokens para verificar el estado de las solicitudes. • El cliente debe suspender el acceso de una cuenta de usuario si la cantidad de códigos de autorización no válidos enviados por este usuario supera un umbral. • El servidor de autorización debe enviar una repuesta de error al cliente informando de un código de autorización no válido y limitar la velocidad o prohibir las conexiones de clientes cuyo número de solicitudes no válidas supere un umbral.
Sustitución de código (inicio de sesión de OAuth)	Un atacante puede usar la identidad de una víctima utilizando el API de identidad que provee OAuth, para iniciar sesión en otra aplicación.	• El Atacante obtiene acceso a una aplicación y a datos específicos del usuario dentro de la misma.	• Todos los clientes deben indicar sus identificadores de clientes con cada solicitud para intercambiar un código de autorización para un token de acceso. El servidor de autorización validará que el código de autorización sea concedido a un cliente en particular. • Los clientes deben usar un protocolo para inicio de sesión apropiado como *OPENID* o *SAML*.

Tabla 7: Modelo de amenazas y recomendaciones para la concesión por código de autorización OAuth2.0.
Fuente: RFC 6819

2.3.5.4.2 Concesión: Implícita

La concesión implícita es un tipo de concesión de OAuth 2.0 usado para obtener el token de acceso (no soporta tokens de actualización) y optimizado para clientes públicos que se conoce que operan con una URI de redirección en particular. Estos clientes normalmente se implementan en un navegador que

utiliza un lenguaje de script como JavaScript. Dado que es un flujo basado en redirección, el cliente debe ser capaz de interactuar con el agente de usuario del propietario del recurso (usualmente un navegador web) y recibir solicitudes (por redirección) del servidor de autorización (RFC-6749, 2012). Diferenciando del tipo de concesión por código de autorización, el método implícito realiza una sola llamada para recibir el token de acceso, en lugar de recibir el código de autorización.

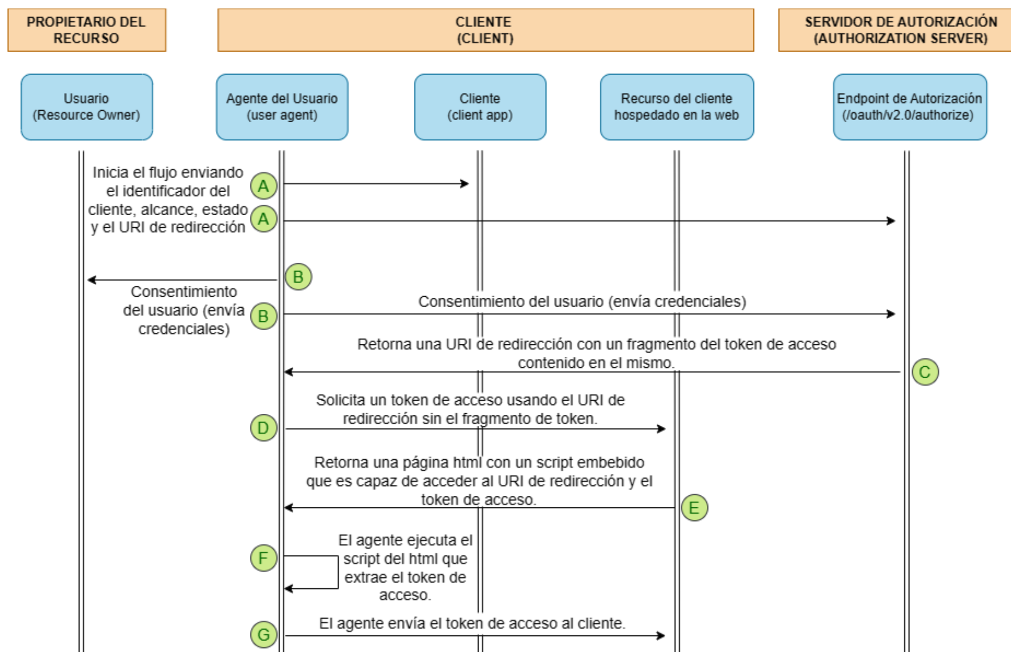


Figura 18: Diagrama de concesión Implícita.
Fuente: RFC-6749

El flujo de concesión de autorización implícita es susceptible de sufrir los siguientes ataques con sus respectivas recomendaciones de seguridad (RFC-6819, 2013):

ATAQUE	DESCRIPCIÓN	CONSECUENCIAS	RECOMENDACIONES
Filtración de tokens de acceso en los puntos de transporte/endpoints	El token se envía desde el servidor al cliente a través de un fragmento de URI en el URI de redirección. Si la comunicación no está protegida extremo a extremo, el token podría filtrarse al analizar el URI derivado.	• El atacante podría obtener el mismo derecho que los otorgados al token.	• Los ataques pueden ser mitigados usando mecanismos TLS. • VPN pueden ser consideradas.
Filtración de los tokens de acceso en el historial de navegación	Un atacante puede obtener un token del historial de navegación del navegador web. Para ello el atacante debe tener acceso a un dispositivo en particular.		• Usar periodos cortos de expiración de tokens. • Reducir el alcance del token de acceso. • Realizar respuestas sin el uso de memoria caché.

Clientes maliciosos obteniendo autorización.	Un cliente malicioso puede pretender ser un cliente legítimo para obtener una autorización de acceso. El cliente malicioso puede intentar simular un consentimiento real del usuario.	• Un cliente malicioso que consigue una autorización y ejecuta acciones como un cliente legítimo.	• Un servidor de autorización puede emitir por separado los identificadores del cliente y los secretos de cada instalación del cliente. • El servidor de autorización debe validar el URI de redirección de cada cliente, comparándolo con el URI de redirección pre registrado. • Después de la autenticación del usuario final, el servidor de autorización debe solicitar consentimiento. • El servidor de autorización no debería autorizar o re autorizar automáticamente clientes que no puede validar. • Uso de captchas o similares. • El servidor de autorización debe limitar el alcance de los tokens.
Manipulación de scripts.	Una parte hostil puede actuar como un servidor web y reemplazar o modificar la implementación actual de un cliente. Esto se puede lograr usando suplantación de DNS o ARP.	• Un atacante podría obtener las credenciales de un usuario y asumir la identidad completa del usuario.	• El cliente debe validar la incrustación de su nombre de dominio. Si el servidor falla al probar esa incrustación, se considera un ataque *man-in-the-middle*. • Se debe asegurar la confidencialidad de las solicitudes enviadas del cliente al servidor de autorización o el servidor de recursos. Pueden usarse técnicas de TLS o VPN o similares. • Introducir secretos de un solo uso que pueden ser usados en scripts por lapsos cortos de tiempo cargados desde el servidor.
Ataque *Cross-site request forgery* (CSRF) contra la URI de redirección	Un CSRF es un ataque de web donde las solicitudes HTTP son transmitidas desde un usuario que es de confianza para el sitio web. El atacante puede obtener el código de autorización.	• El atacante consigue el token de acceso y accede a los recursos de la víctima asociados con su cuenta en su lado de cliente.	• El cliente debe usar un parámetro "*state*" para enviar al servidor de autorización un valor que vincule la solicitud con el estado de autenticación del agente. Este valor permite al cliente la validez de la solicitud comparando el valor de ese parámetro. • Los desarrolladores del cliente y los usuarios finales deben ser capacitados para no abrir URL no confiables.
Sustitución de Token (Inicio de sesión OAuth)	El atacante necesita capturar un token de acceso válido de la víctima del mismo proveedor de identidad utilizado por la aplicación cliente de destino. El atacante engaña a la víctima para que inicie sesión en una aplicación maliciosa utilizando el mismo proveedor de identidad que la aplicación destino.	• El Atacante obtiene acceso a una aplicación y a datos específicos del usuario dentro de la misma.	• Los clientes deben usar un protocolo apropiado para implementar un inicio de sesión seguro como OPENID o SAML.

Tabla 8: Modelo de amenazas y recomendaciones para la concesión Implícita OAuth2.0.
Fuente: RFC 6819

2.3.5.4.3 Concesión: Credenciales de contraseña del propietario del recurso

Es recomendable el uso del tipo de concesión de credenciales de contraseña del propietario del recurso en los casos en que el propietario del recurso confía en el cliente. En este flujo, el propietario otorga sus credenciales (generalmente usuario y contraseña) al cliente y este envía una petición al servidor de

autorización para que retorne un token de acceso y (opcional) un token de actualización (RFC-6749, 2012).

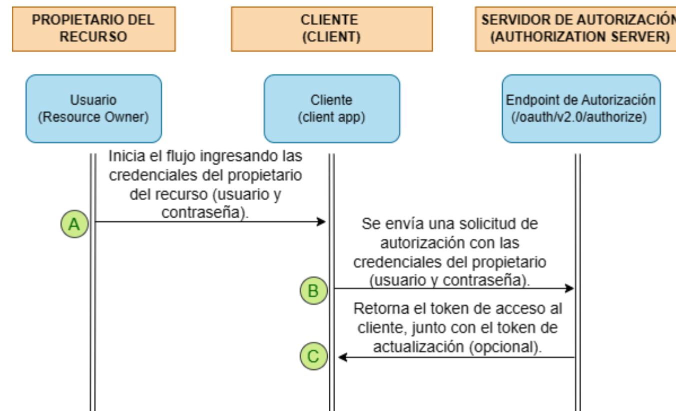


Figura 19: Diagrama de Concesión por Credenciales de contraseña del Propietario del Recurso.
Fuente: RFC-6749

El flujo de concesión de autorización por credenciales de contraseña del propietario del recurso es susceptible de sufrir los siguientes ataques con sus respectivas recomendaciones (RFC-6819, 2013):

ATAQUE	DESCRIPCIÓN	CONSECUENCIAS	RECOMENDACIONES
Uso del anti patrón UID/password.	El propietario del recurso permite al cliente el solicitar un token de acceso mediante el identificador y la contraseña del usuario final.	•Poner en riesgo la credencial de diversas maneras. •Evitar la revocación del token. •Evitar el alcance de la sesión.	•Minimizar el uso del anti patrón. •Proteger las transmisiones con técnicas TLS. •Solicitar al usuario el uso de la misma contraseña para varios servicios. •Limitar el uso de solicitudes de acceso con identificador-contraseña para escenarios dentro de la misma organización.
Exposición accidental de contraseñas en el sitio del cliente.	El cliente puede proveer baja protección y un atacante o un empleado descontento puede recuperar las contraseñas de un usuario.	•Un atacante puede obtener las credenciales de acceso de un usuario.	•Utilizar otros flujos que no dependan de la cooperación del cliente para el manejo seguro de las credenciales del propietario del recurso. •Utilizar la autenticación *digest* (hash sobre las credenciales) en lugar del procesamiento de credenciales en texto plano.
El Cliente obtiene un alcance mayor sin la autorización del usuario final.	Toda la interacción del propietario del recurso es realizada por el cliente. Podría pasar que un cliente obtiene, intencional o inintencionalmente, un token con un alcance desconocido por el propietario de los recursos.	•Un atacante podría realizar operaciones con privilegios más altos de los esperados.	•Utilizar otros flujos que no dependan de la cooperación del cliente para el manejo seguro de las credenciales del propietario del recurso. •El servidor de autorización debe restringir generalmente el alcance del token de acceso. •El servidor de autorización debería notificar al propietario del recurso el acceso concedido. Para ello debe usar un medio apropiado.

El cliente obtiene el token de autorización a través de la autorización automática.	Toda la interacción del propietario del recurso es realizada por el cliente. Podría pasar que un cliente obtiene, intencional o inintencionalmente, un token de actualización de larga duración.	•Un atacante puede obtener un token de actualización que le permitirá obtener tokens de acceso para realizar operaciones como un usuario legítimo.	•Utilizar otros flujos que no dependan de la cooperación del cliente para la interacción con el propietario de los recursos. •El servidor de autorización puede generalmente negarse a emitir tokens de actualización en este flujo. Si el cliente es confiable, puede autenticarse de manera confiable. •El servidor de autorización debería notificar al propietario del recurso el token de actualización concedido. Para ello debe usar un medio apropiado.
Obtención de contraseñas de usuario en el transporte	Un atacante puede hacerse de las credenciales en la transmisión entre el cliente y el servidor.	•Divulgación de una contraseña de un usuario.	•Asegurar la confidencialidad de las solicitudes usando mecanismos TLS, VPN, etc. •Usar una autenticación alterna que no requiera el envío de contraseñas en texto plano, como por ejemplo los basados en códigos hash.
Obteniendo las contraseñas de los usuarios de la base de datos del Servidor de autorización.	Un atacante puede obtener una combinación válida de usuario/contraseña de la base de datos del servidor de autorización obteniendo acceso a la base de datos o ejecutando ataques de inyección SQL.	•Divulgación de una combinación válida de nombre de usuario/contraseña. Un atacante puede usar estas combinaciones para acceder a varios servicios.	•Reducir los privilegios de acceso de la base de datos al mínimo. •Evitar los inputs concatenados con SQL dinámico. •Parametrizar las consultas para inyectar los argumentos en lugar de concatenarlos. •Filtrar y limpiar todas las entradas de datos. •No guardar credenciales en texto plano en la base de datos. Usar encriptación. •Usar criptografía asimétrica.
Adivinando las credenciales del usuario (nombre/contraseña).	Un atacante puede intentar adivinar combinaciones válidas de usuario/contraseña.	•Revelación de una combinación usuario/contraseña de un usuario.	•Utilizar políticas para asegurar contraseñas. •Bloquear cuentas que cumplen un límite de intentos de acceso fallidos. •Usar CAPTCHAs en el proceso de envío. •Considerar no usar el tipo de autorización por envío de credenciales. •Usar un segundo factor de autenticación del lado del cliente.

Tabla 9: Modelo de amenazas y recomendaciones para la concesión por contraseña del propietario del recurso OAuth2.0.
Fuente: RFC 6819

2.3.5.4.4 Concesión: Credenciales del cliente

La concesión por credenciales del cliente comprende de la confianza del propietario del recurso sobre el cliente. Un cliente contiene un identificador, así como un secreto que sirve para conseguir autorización. Su flujo, así como los ataque a los que es susceptible, son los mismos descritos en el paso anterior.

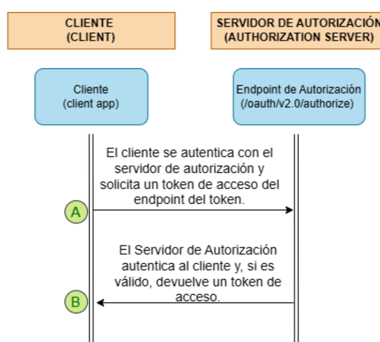


Figura 20: Diagrama de concesión por credenciales del cliente.
Fuente: RFC-6749

2.3.5.5 Flujo: Actualizando un token de acceso

Un token de acceso tiene un tiempo de vida específico. Cuando su tiempo de vida haya expirado, o se haya revocado los permisos del mismo, el cliente vuelve a solicitar un token de acceso para lo cual usa un token de actualización. Este token de actualización evitará realizar todo el flujo de autorización de nuevo y le permitirá al cliente hacer una sola llamada para conseguir nuevamente un token de acceso válido y vigente.

El flujo que se sigue para obtener el token de acceso, a partir de un token de actualización, está representado por los pasos 19 y 20 de la **Figura 13**, específicamente son:

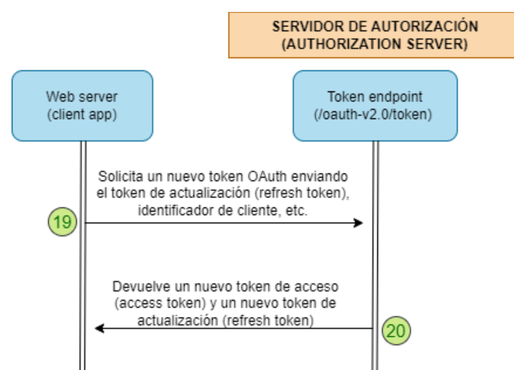


Figura 21: Flujo de actualización de un token de acceso.
Fuente: https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-auth-code-flow

Este flujo es susceptible de sufrir los siguientes ataques con sus respectivas recomendaciones (RFC-6819, 2013):

ATAQUE	DESCRIPCIÓN	CONSECUENCIAS	RECOMENDACIONES
Filtración de tokens de actualización del servidor de autorización.	Un atacante puede filtrar tokens de actualización cuando son transmitidos de un servidor de autorización al cliente.	• Divulgación de tokens de actualización que concedan accesos a largos plazos de un atacante a los recursos protegidos.	• Los servidores de autorización deben asegurar la transmisión mediante técnicas TLS. • Si no se puede garantizar la confidencialidad (extremo a extremo), se puede reducir el alcance y el tiempo de expiración de los tokens de actualización.
Obtención de tokens de actualización de la base de datos del servidor de autorización.	Un atacante puede obtener un token de actualización de la base de datos del servidor de autorización obteniendo acceso a la base de datos o ejecutando un ataque de inyección SQL.		• Aplicando buenas prácticas para el almacenamiento de credenciales. • Usar el mínimo de privilegios a la base de datos. • Usar buenas prácticas para evitar los ataques de inyección de SQL. • Incrustar el identificador del cliente al token.
Adivinando el token de actualización.	Un atacante puede adivinar un token de actualización y enviarlo para conseguir un token de acceso.	• Exposición de un token de actualización que deriva en un token de acceso.	• Usar secretos de alta entropía para causar mayor dificultad de adivinanza. Por lo general un token debe ser mayor o igual a 128 bits. • Tokens deben ser firmados con el objetivo de verificar su validez. • Incrustar el token al identificador del cliente también de modo que se añade otro elemento a adivinar. • Autenticar el cliente, esto añade otro elemento más a adivinar.

Phishing del token de actualización por un Servidor de Autorización falsificado.	Un atacante podría obtener los tokens de actualización válidos enviando solicitudes al servidor de autorización. Dada la circunstancia que el atacante conozca una dirección de un servidor de autorización, éste intentará suplantar la identidad para tener éxito.	• Exposición de tokens de actualización válidos que deriva en los tokens de acceso.	• El URI de redirección del cliente debe apuntar a un endpoint protegido con el protocolo HTTPS, y el navegador web debe ser usado para validar esta URI de redirección mediante el dominio registrado en el cliente.

Tabla 10: Modelo de amenazas y recomendaciones para el flujo de actualización de tokens de acceso OAuth2.0.
Fuente: RFC 6819

2.3.5.6 Flujo: Accediendo a un recurso protegido

Una vez que un cliente consiga un token de acceso, lo utilizará para acceder a un recurso protegido. Los pasos 15, 16 y 17 de la **Figura 13** reflejan claramente este flujo.

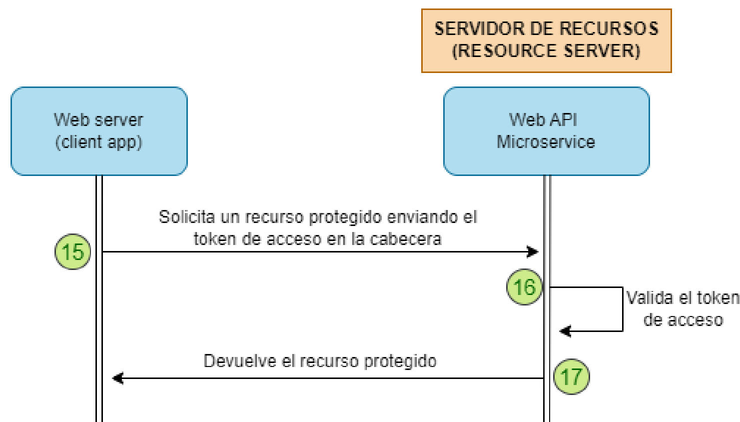


Figura 22: Flujo de acceso a un recurso protegido.
Fuente: https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-auth-code-flow

Este flujo es susceptible de sufrir los siguientes ataques con sus respectivas recomendaciones (RFC-6819, 2013):

ATAQUE	DESCRIPCIÓN	CONSECUENCIAS	RECOMENDACIONES
Filtrado de tokens de acceso en el transporte	Un atacante puede intentar obtener un token de acceso válido durante la comunicación entre el cliente y el servidor de recursos.	• Exposición de los tokens de acceso.	• Los tokens de acceso no pueden ser enviados por canales inseguros. Para ellos podemos usar mecanismos de TLS. • Tokens con tiempo de vida corto, para reducir el impacto. • El token de acceso puede ser incrustado al identificador del cliente para proveer de legitimidad en el servidor de recursos.
Reproducción de solicitudes del servidor de recursos autorizado.	Un atacante puede intentar reproducir solicitudes válidas para obtener o modificar/eliminar datos del usuario.	• Los datos del usuario podrían sufrir modificaciones o pérdida.	• El servidor de autorización debe usar canales de comunicación seguros con mecanismos de TLS. • Las solicitudes deben ser firmadas y el servidor de recursos debería aceptar sólo peticiones firmadas.

Adivinando tokens de acceso.	Donde el token es manejado, un atacante puede intentar adivinar el token de acceso basado en el conocimiento que tengan de los tokens de acceso.	• Acceso a los datos de un usuario.	• Usar secretos de alta entropía para causar mayor dificultad de adivinanza. Por lo general un token debe ser mayor o igual a 128 bits. • Tokens deben ser firmados con el objetivo de verificar su validez. • Incrustar el token al identificador del cliente también de modo que se añade otro elemento a adivinar. • Emitir tokens de corta duración.
Phishing del token de acceso por un Servidor de Autorización falsificado.	Un atacante puede suplantar un servidor de recursos en particular y aceptar tokens de un servidor de autorización.	• Si el cliente envía un token de acceso válido a este servidor de recursos falso, el servidor puede usar dicho token para acceder a otros servicios en nombre del propietario del recurso.	• Los clientes no se deberían enviar tokens de acceso a servidores de recursos no familiares, incluso en presencia de canales seguros. • Asociar la URL al servidor de recursos y validar la asociación a un servidor de recursos legítimos. • Asociar un token de acceso a un cliente y autenticar el cliente con las peticiones al servidor de recursos. • Restringir el alcance del token de acceso.
Abuso de un token por un Servidor de recursos o un cliente legítimos.	Un servidor de recursos legítimo podría intentar usar un token de acceso para acceder a otro servidor de recursos. De manera similar un cliente podría usar un token de acceso para otro servidor de recursos.	• Exposición de los tokens de acceso.	• Los tokens deben ser restringidos para servidores de recursos particulares. • Se pueden generar varios tokens con diferentes contenidos para diferentes servidores de recursos.
Fuga de datos confidenciales en proxys HTTP.	Un esquema de autenticación HTTP OAuth es opcional. Sin embargo, los proxys y los caches pueden fallar por no proteger adecuadamente las solicitudes que no contengan encabezados de seguridad como *WWW-Authenticate*. Esta falla suele almacenar contenidos privados.	• Exposición de datos confidenciales.	• Los clientes y los servidores de recursos que no utilizan un esquema HTTP OAuth deben tener cuidado de usar los encabezados de control de caché para minimizar el riesgo de que el contenido privado quede expuesto. • Reducir el alcance y el tiempo de expiración para los tokens de acceso disminuyan el riesgo.
Fuga de tokens mediante los archivos de registro (logs) y referencias HTTP.	Si el token de acceso es enviado mediante parámetros de consulta en las URI, dichos tokens pueden filtrarse a los archivos de registro y al parámetro HTTP "*referrer*".	• Exposición de tokens de acceso.	• Usar la cabecera de autorización o los parámetros del POST en lugar de los parámetros de consulta. • Configurar apropiadamente los registros (logs). • Configurar apropiadamente los sistemas para evitar el acceso a archivos de configuración o a bases de datos. • Un servidor de autorización puede incrustar tokens a un cierto identificador de un cliente y habilitar un servidor de recursos para validar esa asociación al acceder a un recurso. • Puede mitigarse al limitar el alcance o reducir el tiempo de vida de los tokens.

Tabla 11: Modelo de amenazas y recomendaciones para el flujo de acceso a recursos protegidos OAuth2.0.
Fuente: RFC 6819

3. Desarrollo e Implementación

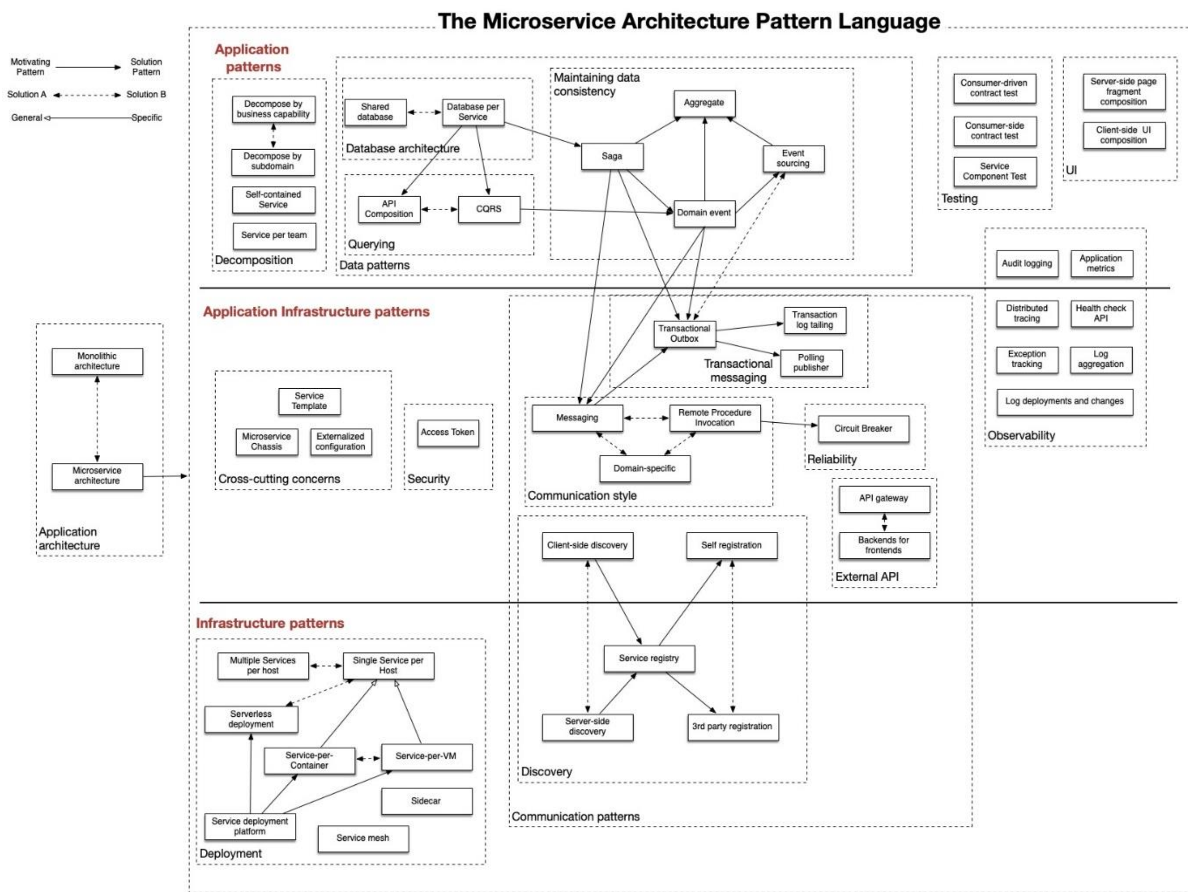
Cuando se habla de arquitecturas de microservicios, existen distintos enfoques donde cada uno de ellos integra varios componentes que varían acorde a los múltiples retos a enfrentar cuando se implementa este tipo de arquitecturas. En este sentido, existen varias guías a tomar en cuenta cuando de arquitecturas de microservicios se implementa; sin embargo, tomamos como referencia el lenguaje de patrones de arquitecturas de microservicios (Richardson, 2022).

3.1 Patrones en Arquitecturas de Microservicios

Este lenguaje de patrones de arquitecturas de microservicios es una colección de patrones aplicados a arquitecturas de microservicios que tiene como objetivos (Richardson, 2022):

- El lenguaje de patrones ayuda a decidir que arquitectura es adecuada para tu proyecto.
- El lenguaje de patrones ayuda a usar una arquitectura de microservicios de manera exitosa.

Este lenguaje está resumido en la siguiente figura:



Copyright © 2022. Chris Richardson Consulting, Inc. All rights reserved.

Figura 23: Lenguaje de patrones de arquitecturas de microservicios.
Fuente: https://microservices.io/patterns/index.html

3.2 Definiendo una arquitectura de microservicios

Por la naturaleza del presente trabajo, se ha decidido desarrollar una arquitectura sencilla con el patrón de API externa (*External API*) haciendo uso de un componente denominado API Gateway.

Una puerta de enlace API o API Gateway es una herramienta de gestión de APIs que se encuentra entre el cliente y un conjunto de servicios de *backend* encargados de las tareas habituales que se utilizan en los sistemas de servicios de API como la autenticación de los usuarios, la limitación de la frecuencia, estadísticas, etc. En general, una puerta de enlace API varía de una implementación a otra, pero su objetivo principal es interceptar todas las solicitudes que ingresan, y las envían a través de un sistema de gestión de API para obtener los recursos de varios servicios (RedHat, 2019).

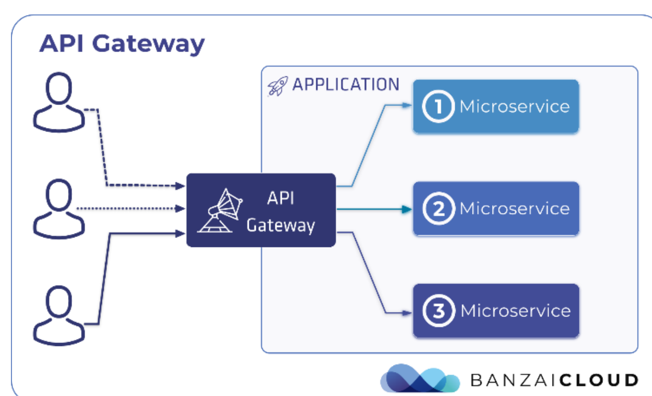


Figura 24: Puerta de enlace API (API Gateway).
Fuente: https://banzaicloud.com/blog/backyards-api-gateway/

Ahora bien, existen principalmente dos formas de integrar un patrón de microservicios con puerta de enlace (API Gateway) al protocolo OAuth 2.0 (Sevestre, 2022):

- API Gateway como un cliente OAuth 2.0
- API Gateway como un Servidor de Recursos

En el **API Gateway como cliente OAuth 2.0**, el flujo queda determinado por los siguientes pasos:

1. El usuario inicia una petición de recursos al API Gateway.
2. El API Gateway verifica si posee un token de acceso válido, caso contrario inicia una redirección hacia el Servidor de Autorización para solicitar acceso al usuario.
3. El usuario ingresa sus credenciales.
4. El Servidor de Autorización envía al API Gateway un token de acceso y un token de actualización.
5. El API Gateway solicita a un microservicio (Servidor de recursos) un recurso protegido enviando el token de acceso.
6. El microservicio verifica la validez del token con el Servidor de Autorización.
7. En caso de ser un token de acceso válido, retorna el recurso protegido.

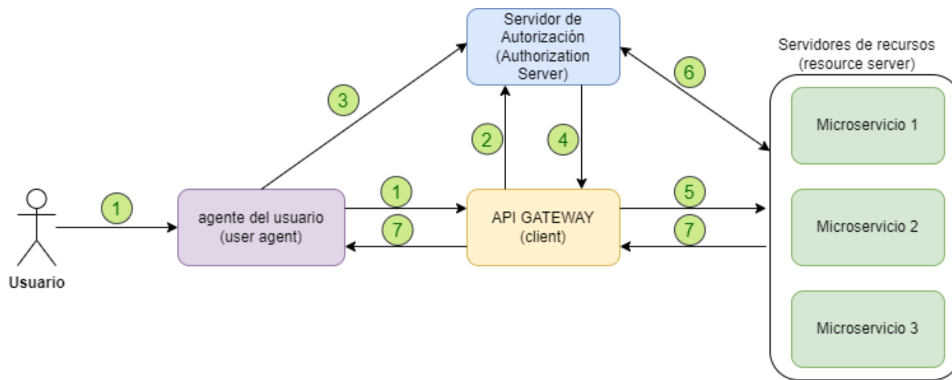


Figura 25: API Gateway como un cliente OAuth 2.0.
Fuente: Elaboración propia

En el enfoque de un **API Gateway como un servidor de recursos**, el flujo queda determinado por los siguientes pasos:

1. El usuario inicia una petición de autorización al Servidor de Autorización.
2. El usuario envía las credenciales al Servidor de Autorización.
3. El Servidor de Autorización envía el token de acceso y el token de actualización al cliente.
4. El Cliente solicita un recurso protegido al API Gateway.
5. El API Gateway verifica en el Servidor de Autorización que el token de acceso es válido.
6. En el caso de serlo, el API Gateway solicita el recurso protegido a un microservicio.
7. El microservicio verifica en el Servidor de Autorización que el token de acceso es válido.
8. En caso de serlo, se devuelve el recurso protegido.

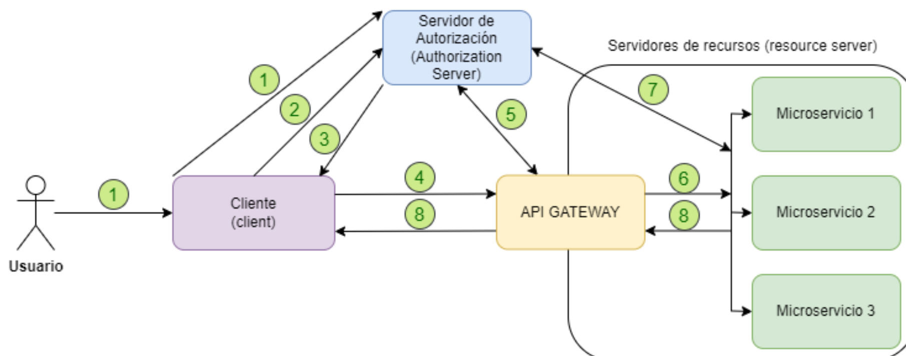


Figura 26: API Gateway como un servidor de recursos OAuth 2.0.
Fuente: Elaboración propia

Cabe recalcar que, en estas arquitecturas mencionadas, no se ha incluido ningún patrón de integridad de datos como Sagas, ni patrones de descubrimiento como Service Discovery, ni tampoco comunicación asíncrona como RabbitMQ o protocolos AMQP, ni otros; el único objetivo es mostrar una integración del protocolo OAuth 2.0 en el mundo de la autenticación de microservicios.

3.3 Diseño de la arquitectura de microservicios

De las arquitecturas de microservicios que integran un flujo OAuth2.0 mencionadas en el apartado anterior, ambas varían únicamente en la forma de actuar del usuario y el API Gateway con el Servidor de Autenticación. De tal forma que, para una mayor comprensión y simplicidad, se ha decidido incorporar aquella arquitectura cuyo API Gateway actúa como un cliente OAuth 2.0, mencionando que toda la implementación posterior, puede ser fácilmente adaptable a la otra arquitectura mencionada.

En este sentido, el sistema se regirá a cumplir el flujo representado en la figura 20, así como también en el diagrama de secuencia de la figura 22 (Sevestre, 2022).

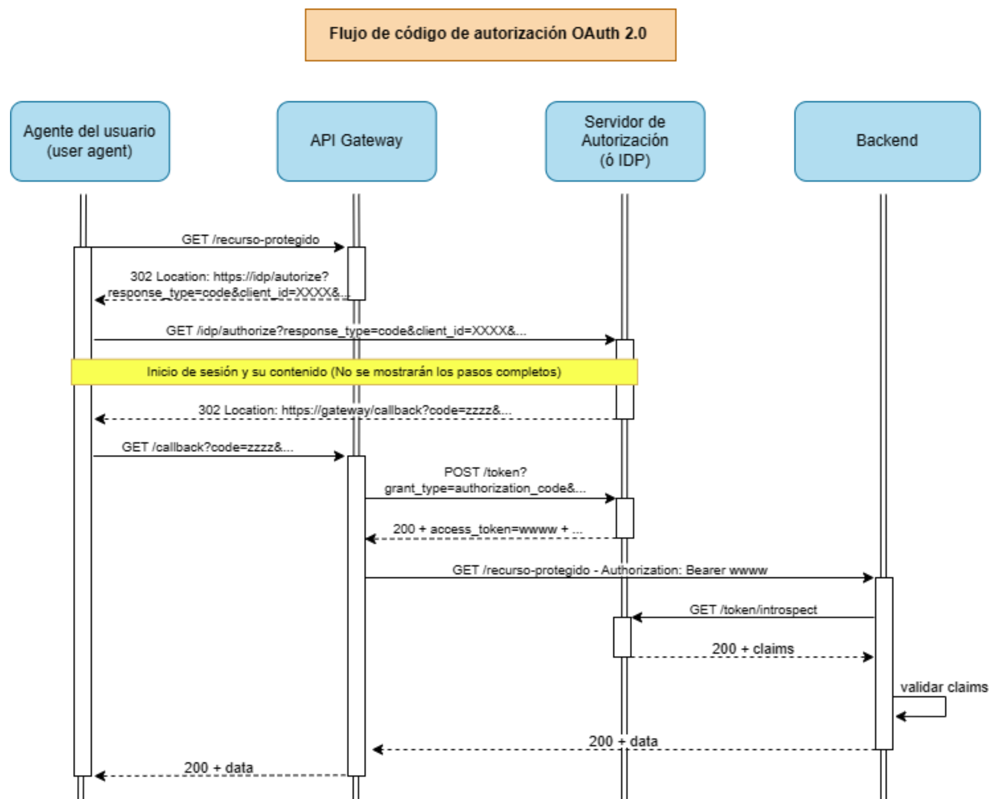


Figura 27: Diagrama de secuencia de una arquitectura de microservicios con un API Gateway como un cliente OAuth.
Fuente: https://www.baeldung.com/spring-cloud-gateway-oauth2

Con estos antecedentes, se definen los componentes a desarrollar y su descripción, en la siguiente tabla:

COMPONENTE	DESCRIPCIÓN	ROLE EN OAUTH 2.0
Agente del Usuario (*user-agent*)	Es el aplicativo o tecnología usada por el cliente para ejecutar llamadas al API Gateway y al Servidor de Autorización.	Cliente (*client*)
API Gateway	Es un gestor de API destinado a servir de proxy reverso para obtener los recursos de los microservicios.	Cliente (*client*)

Servidor de Autorización	Un Servidor de Autorización, acorde a OAuth 2.0.	Servidor de Autorización (*Authorization Server*)
Microservicio 1	Un microservicio que expone un endpoint protegido que nos devolverá los valores protegidos.	Servidor de Recursos (*Resource Server*)
Microservicio 2		
Microservicio 3		

Tabla 12: Definición de componentes a desarrollar.
Fuente: Elaboración propia.

Estos componentes, interactuarán entre sí de la siguiente manera:

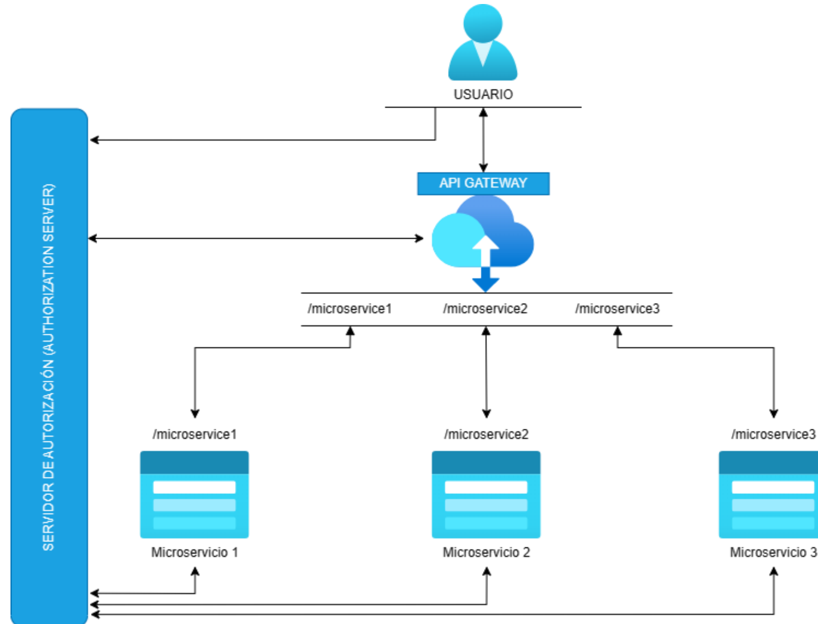


Figura 28: Funcionamiento de los componentes a desarrollar.
Fuente: Elaboración propia.

De donde podemos destacar:

- El API Gateway expondrá 3 endpoints: */microservice1*, */microservice2* y */microservice3.*
- Cada endpoint del API Gateway redireccionará la petición a cada microservicio que contiene un endpoint con un URI similar.
- El API Gateway transmitirá el mismo token de acceso que le fue concedido por el servidor de autorización para que los microservicios hagan uso del mismo y puedan devolver el recurso protegido.
- En caso de que el token de acceso sea válido, el microservicio enviará el recurso protegido al Gateway para ser enviado al agente del usuario.
- Todos los endpoints, tanto del API Gateway como de cada microservicio, se encuentran protegidos por el protocolo OAuth 2.0.
- El Servidor de Autorización estará presente durante todo el flujo de datos del microservicio.

3.4 Desarrollo de la arquitectura de microservicios

Acorde al diseño planteado en el apartado anterior, procedemos a desarrollar cada uno de los componentes de la arquitectura de microservicios.

3.4.1 Desarrollo del API Gateway

Para el desarrollo de este componente, se tuvo presente el siguiente conjunto de tecnologías:

Característica	Valor
Lenguaje de programación	Kotlin 1.6.21
Framework	Spring Boot 2.7.7. RELEASE
Gestor de construcción	Gradle 7.5.1
Versión de lenguaje	java-1.8.0-openjdk
Librerías usadas	• spring-boot-starter-oauth2-client V. 2.7.7 • spring-cloud-starter-gateway V. 2021.0.5

Tabla 13: Tecnologías usadas para el desarrollo del API Gateway
Fuente: Elaboración propia.

Endpoints expuestos:

- /microservice1 -> enrutará al microservicio 1
- /microservice2 -> enrutará al microservicio 2
- /microservice3 -> enrutará al microservicio 3

Configuraciones principales:

Existe tres configuraciones importantes que se encuentran dentro del archivo *application.yml* ubicado dentro del directorio *src/main/resources/*. La primera se refiere al valor del puerto donde será expuesto el servicio:

```
1.  server:
2.    port: 8090
```

La segunda de esas configuraciones responde a la capacidad del API Gateway de ser un cliente de OAuth2.0, para ellos definimos:

```
1.  spring:
2.    security:
3.      oauth2:
4.        client:
5.          provider:
6.            keycloak:
7.              issuer-uri: http://localhost:9000/auth/realms/security
8.          registration:
9.            gateway:
10.             provider: keycloak
11.             client-id: gateway
12.             client-secret: ePEArOxZNZN0AvFig8BN0wSHHCs7eBen
13.             scope:
14.               - email
15.               - profile
16.               - roles
```

Como podemos observar, el Gateway posee un identificador de cliente y un secreto. Este parde valores deberán ser pre registrados en el Servidor de Autorización OAuth 2.0 con el fin de verificar la confianza en el cliente. En el siguiente capítulo se analiza a detalle estos valores.

La tercera parte de la configuración, es la capacidad de enrutar adecuadamente a través de los microservicios. Para ello, definimos:

```
1.  main:
2.    allow-bean-definition-overriding: true
3.    web-application-type: reactive
4.  cloud:
5.    gateway:
6.      redis:
7.        enabled: false
8.      routes:
9.        - id: microservice1
10.         uri: http://localhost:8081
11.         predicates:
12.           - Path=/microservice1
13.         filters:
14.           - TokenRelay=
15.       - id: microservice2
16.         uri: http://localhost:8082
17.         predicates:
18.           - Path=/microservice2
19.         filters:
20.           - TokenRelay=
21.       - id: microservice3
22.         uri: http://localhost:8083
23.         predicates:
24.           - Path=/microservice3
25.         filters:
26.           - TokenRelay=
```

En la anterior configuración, se declaran las rutas para que el API Gateway pueda enrutar hacia los endpoints de los microservicios. Un valor importante es el filtro *TokenRelay* ya que es el necesario para que el Gateway comparta el token de acceso obtenido del servidor de autorización hacia los microservicios.

Código fuente: https://github.com/paulro1575/microservices-authentication-oauth2.0/tree/main/gateway.

NOTA: los secretos compartidos en los archivos de configuración YAML han sido incluidos por motivos académicos. En ambientes de producción, estos secretos, no deberían ser compartidos en los repositorios del software ni ampliamente difundidos; sino más bien, deben ser manejados con mucha confidencialidad por el equipo encargado de los despliegues.

3.4.2 Desarrollo del Servidor de Autorización

Para el servidor de autenticación, se usó una solución *open-source* denominada keycloak (https://www.keycloak.org/). Keycloak es una solución de gestión de acceso e identidad de código abierto que permite la implementación sencilla de

múltiples protocolos; entre ellos OAuth 2.0 junto a OpenID-Connect, SAML, entre otros.

Con este antecedente, para el desarrollo de este componente, se tuvo presente el siguiente conjunto de tecnologías:

Característica	Valor
Código fuente original	https://github.com/thomasdarimont/embedded-spring-boot-keycloak-server
Lenguaje de programación	Java 13 (11 compatible)
Framework	Spring Boot 2.6.7
Gestor de construcción	Apache Maven 3.6.3
Versión de lenguaje	java-13-openjdk
Librerías usadas	• h2database Versión latest • Keycloak 18.0.0

Tabla 14: Tecnologías usadas para el desarrollo del Servidor de Autorización
Fuente: Elaboración propia.

Configuraciones principales:

La primera configuración hace referencia al puerto por el cual se lo despliega:

```
1.  server:
2.    port: 9000
```

La siguiente configuración es importante y se refiere a valores necesarios para que *keycloak* opere normalmente, definidos por:

```
1.  keycloak:
2.    server:
3.      customRealm: security
4.      contextPath: /auth
5.      adminUser:
6.        username: admin
7.        password: password
8.      customUser:
9.        username: security-admin
10.       password: password
11.     realmImportFile: realm-export.json
```

Donde se destaca el usuario y contraseña que tendrá por defecto la consola de administración del servidor de autorización de *keycloak* para el *Realm* máster; así como un *Realm* propio denominado *Security* con un usuario y contraseña. De la misma manera, la configuración se la está almacenando en el archivo JSON *realm-export.json* dentro el directorio */src/main/resources*.

En el siguiente capítulo se aborda a mayor detalle las opciones de keycloak referentes a la seguridad del protocolo de autenticación.

Dado que no se configura una base de datos, el servidor necesita ser configurado cada vez que se inicie (ver **Anexo 1**) para lo cual se exporta un archivo de

configuración y éste será importado cada vez que se inicie el servidor de autorización.

Código fuente: https://github.com/paulro1575/microservices-authentication-oauth2.0/tree/main/oauth-authorization-server

3.4.3 Desarrollo de los Microservicios

Para desarrollar los microservicios, se desarrolló una base similar para todos, y se cambió el archivo de configuración para distinguirlos entre sí. A continuación, se describen los detalles del módulo en común.

Característica	Valor
Lenguaje de programación	Kotlin 1.6.21
Framework	Spring Boot 2.7.7. RELEASE
Gestor de construcción	Gradle 7.5.1
Versión de lenguaje	java-1.8.0-openjdk
Librerías usadas	• spring-boot-starter-oauth2-resource-server V. 2.7.7 • spring-boot-starter-web V. 2021.0.5 • io.projectreactor:reactor-core V. 3.4.24

Tabla 15: Tecnologías usadas para el desarrollo de los Microservicios
Fuente: Elaboración propia.

Endpoints expuestos:

- /microservice1 -> microservicio 1
- /microservice2 -> microservicio 2
- /microservice3 -> microservicio 3

Configuraciones principales:

Dentro del archivo *application.yml* localizado dentro del directorio *src/main/resources/,* cada microservicio se registra como un servidor de recursos en el protocolo OAuth2.0 de la siguiente manera:

Microservicio 1, puerto 8081, configuración para OAuth2.0:

```
1.  security:
2.    oauth2:
3.      resourceserver:
4.        opaquetoken:
5.          introspection-uri: http://localhost:9000/auth/realms/security/protocol/openid-connect/token/introspect
6.          client-id: microservice1
7.          client-secret: 5FlyzwdNmF68kQFfYzifqAeMmqXOnfoJ
```

Código fuente Microservicio 1: https://github.com/paulro1575/microservices-authentication-oauth2.0/tree/main/microservice1

Microservicio 2, puerto 8082, configuración para OAuth2.0:

```
1. security:
2.   oauth2:
3.     resourceserver:
4.       opaquetoken:
5.         introspection-uri: http://localhost:9000/auth/realms/security/protocol/openid-connect/token/introspect
6.         client-id: microservice2
7.         client-secret: rISR8jYh6xagFI7eBAXDpwUPNgyj9Ljx
```

Código fuente Microservicio 2: https://github.com/paulro1575/microservices-authentication-oauth2.0/tree/main/microservice2

Microservicio 3, puerto 8083, configuración para OAuth2.0:

```
1. security:
2.   oauth2:
3.     resourceserver:
4.       opaquetoken:
5.         introspection-uri: http://localhost:9000/auth/realms/security/protocol/openid-connect/token/introspect
6.         client-id: microservice3
7.         client-secret: hSMTHZllY5PGeIHSRcIIICteG7imTVJ1
```

Código fuente Microservicio 3: https://github.com/paulro1575/microservices-authentication-oauth2.0/tree/main/microservice3

NOTA: los secretos compartidos en los archivos de configuración YAML han sido incluidos por motivos académicos. En ambientes de producción, estos secretos, no deberían ser compartidos en los repositorios del software ni ampliamente difundidos; sino más bien, deben ser manejados con mucha confidencialidad por el equipo encargado de los despliegues.

3.5 Ejecución de la arquitectura de microservicios

En este apartado, se compila y ejecuta los componentes de la arquitectura de microservicios descrita en el apartado anterior. Para ello, vamos a necesitar de comandos de consola dentro del directorio principal de cada componente, acorde a la siguiente tabla:

COMPONENTE	PREREQUISITOS	COMPILACIÓN	EJECUCIÓN
Authorization server	* Java JDK 13 * Maven 3.6.x	*mvn install*	*mvn spring-boot:run*
API Gateway	* JAVA 1.8 o superior	*./gradlew build*	*./gradlew bootRun*
Microservice1			
Microservice2			
Microservice3			

Tabla 16: Compilación y ejecución de componentes.
Fuente: Elaboración propia

Al ejecutar el Servidor de Autorización usando *keycloak*, obtenemos un servicio levantado en la dirección ***http://localhost:9000.*** Keycloak, además, expone una interfaz de administración que permite configurar todas las tareas relativas al servidor de autorización (ver **Anexo 1**). Así mismo, en el capítulo siguiente se analizan las opciones de keycloak que mejoran la seguridad de la implementación del protocolo.

Una vez levantado el servicio, esta interfaz es accesible mediante el siguiente enlace ***http://localhost:9000/auth/admin/master/console/#/realms/master*** donde podemos observar la siguiente interfaz:

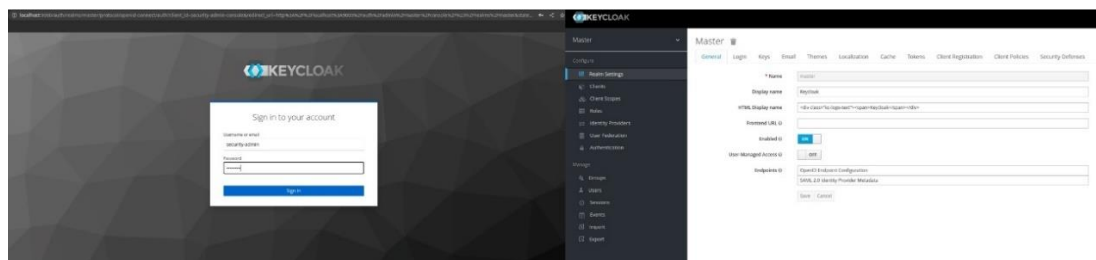


Figura 29: Servidor de Autorización con Keycloak.
Fuente: Elaboración propia.

Una vez configurado el servidor de autorización, se ejecutan el resto de componentes, obteniendo el siguiente resultado:

Componente	URL	Puntos de entrada
Servidor de Autorización	http://localhost:9000	Expuestos por Keycloak
API Gateway	http://localhost:8090	/microservice1 /microservice2 /microservice3
Microservicio 1	http://localhost:8081	/microservice1
Microservicio 2	http://localhost:8082	/microservice2
Microservicio 3	http://localhost:8083	/microservice3

Tabla 17: Características del despliegue de los componentes
Fuente: Elaboración propia.

Con este entorno, se procede a ejecutar los microservicios que forman parte de esta arquitectura y verlos operando en conjunto con el protocolo OAuth2.0. En este sentido, se ejecuta la siguiente lista de pasos:

1. Una llamada al API Gateway para traer el recurso protegido proveniente del microservicio 1.
2. El API Gateway verifica que no existe autorización del usuario aún y redirige al Servidor de Autorización (de fondo envía una URI de redirección).
3. Una vez otorgado el consentimiento, el servidor de autorización usa la URI de redirección, comprueba su veracidad, y redirige al usuario al endpoint originalmente llamado.
4. De fondo, cada microservicio recibió el token de acceso, verificó su validez, y devolvió su recurso protegido hacia el API Gateway.

Este comportamiento se puede ver resumido en la siguiente figura:

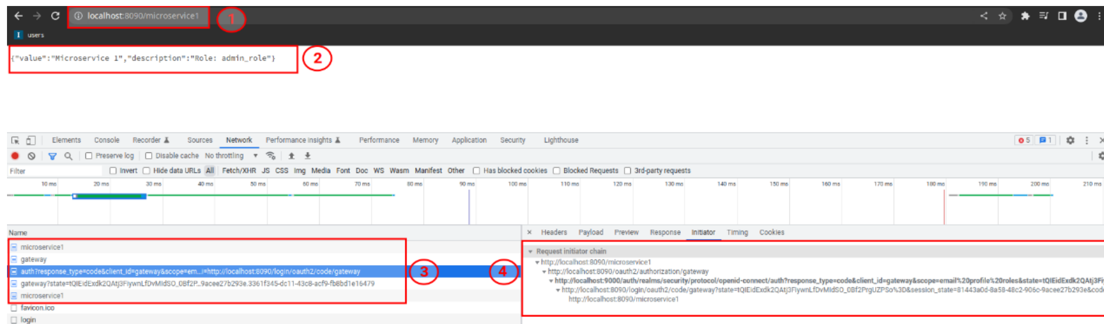


Figura 30: Ejecución de la arquitectura de microservicios hacia el microservicio 1
Fuente: Elaboración propia.

En ella se resalta:

1. La URI del API Gateway en la barra de direcciones.
2. El valor del recurso protegido siendo mostrado.
3. Actividad de la red ejecutada al llamar al endpoint del API Gateway.
4. Conjunto de endpoints y llamadas ejecutadas desde el navegador web: la llamada fue iniciada por el API Gateway (*localhost:8090*), esta redirigió al servidor de autorización (*localhost:9000*) y nuevamente fue redirigido hacia el API Gateway con la misma URL llamada al inicio. De fondo, el Gateway se comunicó con el microservicio 1, donde se envió el token de acceso para que el microservicio se comunique con el Servidor de Autorización, valide el token de acceso y devuelva el recurso protegido.

Si llamamos al microservicio 2 y 3, podemos ver una actividad similar en el navegador, con la única diferencia que es el microservicio 2 y 3 respectivamente el que devuelve su recurso protegido:

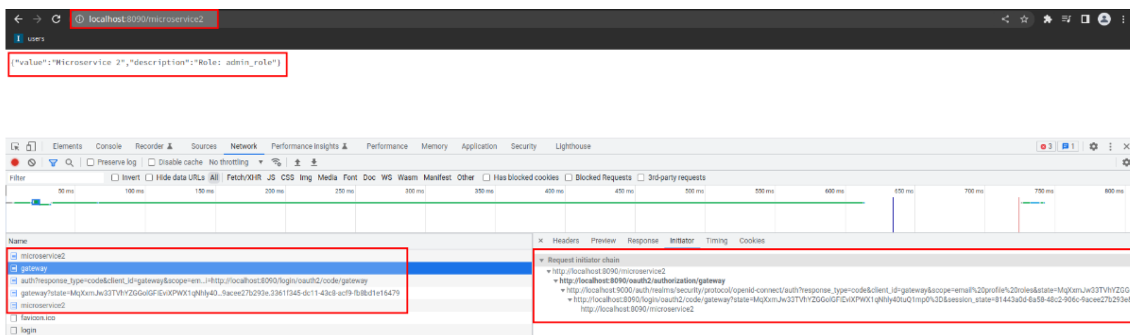


Figura 31: Ejecución de la arquitectura de microservicios hacia el microservicio 2
Fuente: Elaboración propia.

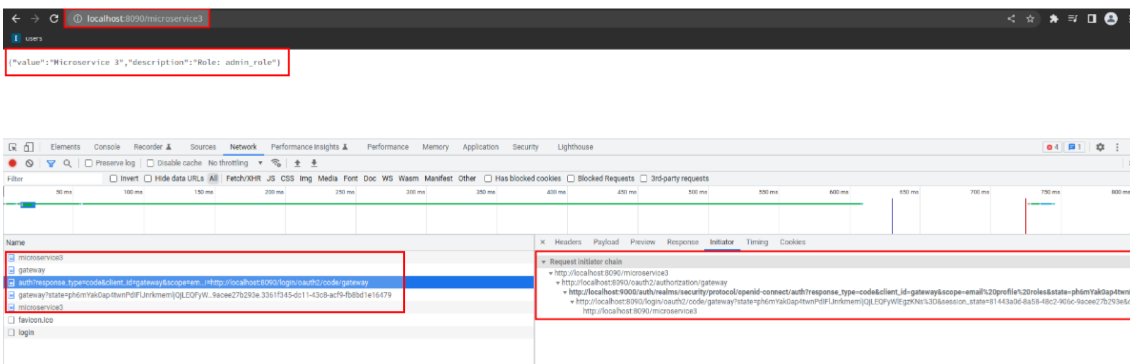


Figura 32: Ejecución de la arquitectura de microservicios hacia el microservicio 3
Fuente: Elaboración propia.

De forma conjunta, en la **Figura 33** se muestran los registros de ejecución de todos los componentes al mismo tiempo.

Figura 33: Ejecución conjunta de toda la infraestructura de OAuth2.0 y microservicios
Fuente: Elaboración propia.

En este último, gracias a la ayuda de un multiplexor de terminal, en la misma imagen destacamos:

1. Servidor de Autorización con Keycloak
2. Ejecución y registro del API Gateway
3. Microservicio 1 y la llamada efectuada sobre su endpoint.
4. Microservicio 2 y la llamada efectuada sobre su endpoint.
5. Microservicio 3 y la llamada efectuada sobre su endpoint.

4. Seguridad en el entorno de microservicios usando el protocolo OAuth

En este apartado se analiza la configuración del servidor de autorización construido con Keycloak para verificar que las recomendaciones de seguridad expuestas en el documento RFC-6819 y en el **apartado 2.3.5** del presente trabajo sean tomadas en cuenta.

4.1 Conceptos básicos de Keycloak

Keycloak permite configurar e implementar la seguridad necesaria en torno al buen uso del protocolo OAuth en el entorno de microservicios, haciendo uso de los siguientes conceptos iniciales (Keycloak, 2023):

Realms: un *realm* es un dominio, en keycloak hace referencia a un conjunto de usuarios, roles y grupos. Un usuario pertenece a un dominio (realm) y se registra en él. Estos dominios están aislados entre sí y sólo se puede gestionar y autenticar a los usuarios que controlan.

Users: son entidades capaces de iniciar sesión en el sistema. Cada usuario tiene atributos y alcances, pueden ser asignados a grupos y roles.

Groups: los grupos manejan conjuntos de usuarios y para cada grupo se pueden definir atributos y límites. Se pueden mapear un conjunto de roles para un grupo de la misma manera.

Clients: un cliente en keycloak son entidades que pueden solicitar la autorización del usuario.

Client scope: cuando un cliente se registra, se debe definir los ámbitos (scope). Éstos son útiles para definir accesos y límites que tendrá el cliente.

Sessions: cuando un usuario o cliente inicia sesión en keycloak, éste conserva la sesión de usuario y recuerda cada cliente que fue visitado usando dicha sesión. Keycloak implementa un administrador de sesiones, donde se puede remover una sesión concedida.

Identity providers: un proveedor de identidad IDP es un servicio capaz de autenticar un usuario, Keycloak por sí mismo es un IDP que nos permite configurar algún otro IDP como Facebook, Google, Microsoft, GitHub, LinkedIn, PayPal, entre muchos otros. De la misma manera, nos permite configurar algún otro IDP propio del usuario como un directorio activo (Active Directory), así como un LDAP o Kerberos haciendo uso de las federaciones del usuario.

En el capítulo anterior, los usuarios, roles, API Gateway y los microservicios fueron registrados en el dominio (*realm*) denominado "*security*", sobre el cuál vamos a trabajar. De la misma manera, los clientes fueron registrados y el entorno fue pre configurado para cumplir con el desarrollo como se observa en el **Anexo 1**.

En este sentido, se utilizará el entorno desarrollado que se compone de los clientes, servidor de Autorización Keycloak y los proveedores de recursos para verificar algunas amenazas y consideraciones de seguridad expuestas en el documento RFC 6819.

4.2 Seguridad de los clientes

Un aspecto esencial de la configuración del entorno usando OAuth 2.0 es la configuración de seguridad referente al cliente. En este apartado se analizan distintos aspectos a tener en cuenta para configurar los clientes en Keycloak y OAuth2.0.

Como ya fue descrito, un cliente de Keycloak es cualquier entidad que desee conseguir autorización. La entidad deberá ser registrada como cliente para poder comunicarse con el servidor de autorización.

En nuestro entorno, hemos registrado el Gateway y los microservicios como clientes de keycloak. Es decir, nuestros clientes y nuestros servidores de recursos de OAuth son tratados como clientes Keycloak.

Figura 34: Registro de clientes en servidor Keycloak
Fuente: Elaboración propia.

Los clientes pueden ser públicos o confidenciales. La principal diferencia entre ellos es que un cliente público podrá solicitar autorización de manera directa en el servidor de autorización únicamente con su identificador; mientras que un cliente confidencial necesitará de un secreto, que es un código que actúa como llave; así como de su identificador.

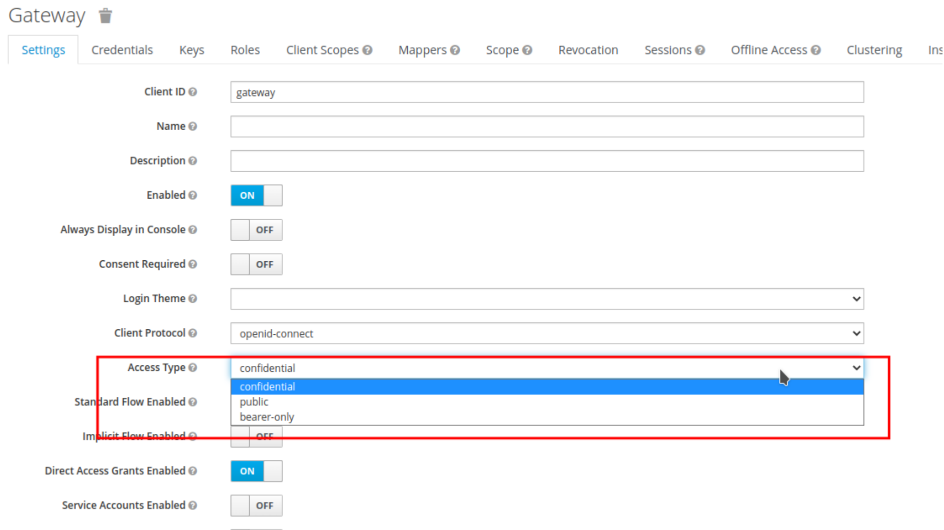


Figura 35: Tipos de clientes en servidor Keycloak
Fuente: Elaboración propia.

4.2.1 Entropía en los secretos del cliente

Cuando se registra un cliente confidencial, Keycloak crea automáticamente un identificador y un secreto para el mismo. El objetivo de estos campos es que los clientes tengan una identidad y una seguridad el momento de solicitar autorización ante el servidor de autorización.

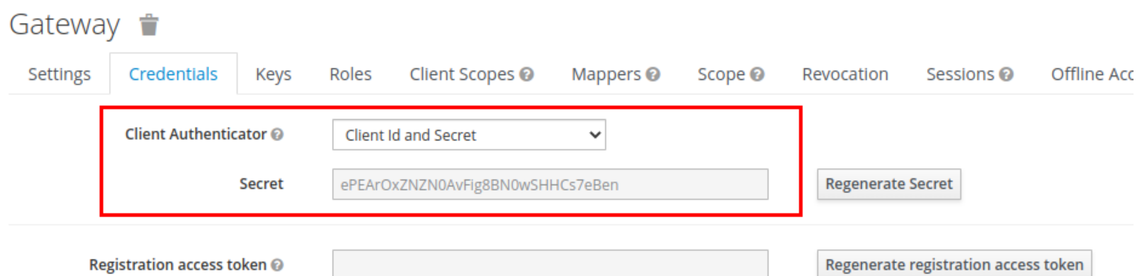


Figura 36: Secretos del cliente de Keycloak
Fuente: Elaboración propia.

El documento RFC 6819 expone algunas amenazas cuyo objetivo es este secreto, en este punto se analizará la amenaza: “Adivinando los secretos del cliente” expuestos en la tabla **Tabla 6**. Entre las consideraciones expuestas en este RFC, una de ellas nos indica que cuando se crean secretos cuya intención es no ser usados por humanos, el servidor de autorización debe incluir un nivel razonable de entropía para mitigar esta amenaza. Nos indica, además, que el secreto debe ser de un tamaño >=128 bits generados de manera segura (RFC-6819, 2013), apartado 5.1.4.2.2 de dicho documento.

En este sentido, se toma el secreto generado en la **Figura 36**, y se usa la siguiente herramienta online de medición de entropía, como es *timecutting.co.uk/tools/password-entropy*, con el objetivo de medir la entropía de secreto anterior.

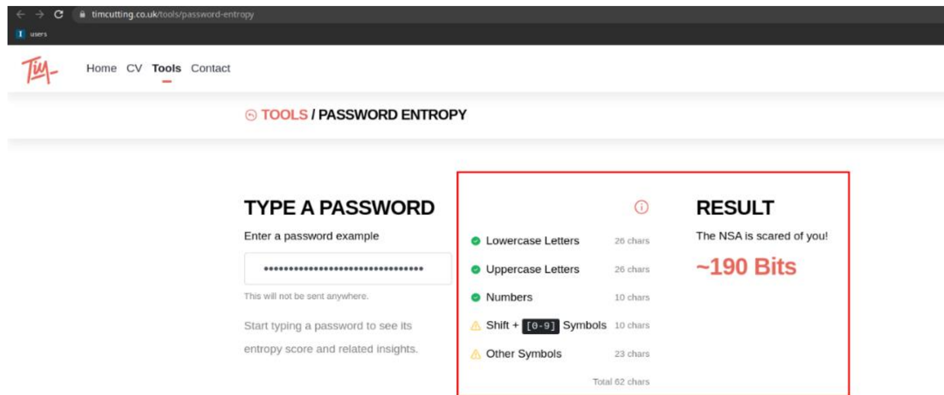


Figura 37: Nivel de entropía del secreto del cliente
Fuente: https://timecutting.co.uk/tools/password-entropy,

Como se observa en la figura anterior, el nivel de entropía de los secretos generador por Keycloak se encuentran dentro del rango considerado seguro por el documento RFC 6819.

4.2.2 Clientes públicos y de baja confianza

Como se mencionó en apartados anteriores, existen clientes confidenciales y públicos. Cuando un cliente es confidencial, Keycloak asigna automáticamente un secreto y un identificador; mientras que, para clientes públicos, únicamente se le asigna un identificador del cliente.

El documento RFC 6819 nos indica que no se puede compartir secretos con clientes que no los puedan proteger; es decir, clientes públicos cuyo código pertenezca a fuentes desconocidas o sea públicamente distribuido. A estos clientes se los considera de políticas inapropiadas o públicas.

La idea principal de no compartir secretos con dichos clientes, radica en evitar que un cliente pueda ser tratado, por el servidor de autenticación, como un cliente fuertemente autenticado (RFC-6819, 2013).

En este contexto, una de las amenazas expuestas por el documento RFC 6819 nos indica que, si un cliente es público, hay que tratarlo como tal. En Keycloak se lo debe configurar como un cliente público de la siguiente manera.

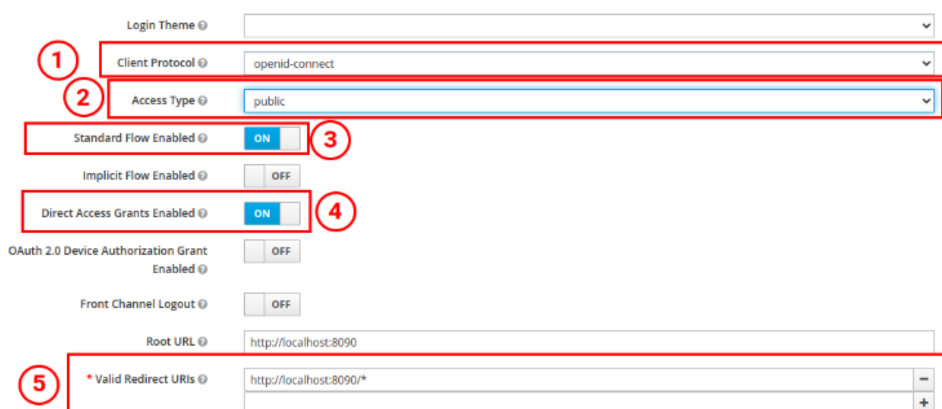


Figura 38: API Gateway como un cliente público
Fuente: Elaboración propia.

Donde se destaca:

1. El Protocolo de identidad del cliente, en este caso OpenID Connect, es una capa adicional que se monta sobre OAuth con el fin de dotar de identidad al protocolo (su análisis y prueba no se incluyen en este apartado).
2. Tipo de acceso público que define al cliente Keycloak como cliente público. Al definirlo de esta manera, automáticamente Keycloak evita crear secretos para este tipo de cliente.
3. El flujo estándar habilitado hace referencia a permitir que, dentro del flujo de autorización, se ocupe el tipo de concesión por código de autorización del **apartado 2.2.2.2** (este parámetro es necesario para el funcionamiento del entorno definido en el **apartado 3.3**).
4. El flujo de acceso directo sirve para que el servidor de autorización permita el tipo de concesión por credenciales de contraseñas del usuario definido en el **apartado 2.2.2.2.** En este caso, se lo habilita para las pruebas realizadas directamente mediante el cliente API de la **Figura 39** y similares.
5. Registro de las URI de redirección permitidas para este cliente. Este paso es obligatorio en Keycloak y cumple con las consideraciones expuestas en el documento, específicamente la amenaza denominada **Redirección Abierta** en la **Tabla 5** (su análisis y prueba no se incluyen en este apartado).

Una vez configurado el API Gateway como cliente público y, una vez permitido el punto 4 de la **Figura 38**, vamos a usar un cliente API (en nuestro caso Insomnia) para probar el flujo actual. Para lo cual se definió la siguiente configuración de variables para de entorno:

```
1.  {
2.      "auth_server": "http://localhost:9000",
3.      "server": "http://locahost:8090",
4.      "resource_server_1": "http://localhost:8081",
5.      "resource_server_2": "http://localhost:8082",
6.      "resource_server_3": "http://localhost:8083",
7.      "realm": "security",
8.      "client_id": "gateway",
9.      "client-secret": "bD8ABWRzWn2MpGxtfrNSzIp7rlwdEfQC",
10.     "user": "security-admin"
11.     "password": "password"
12. }
```

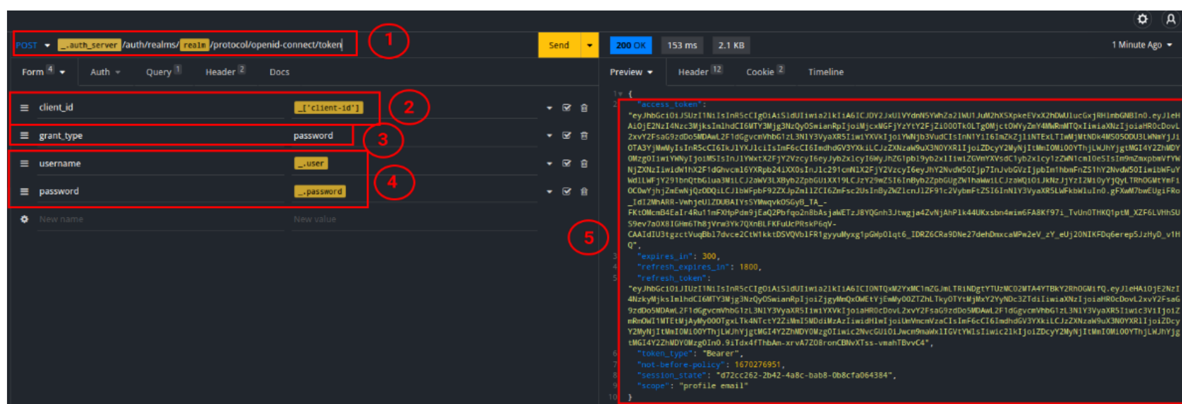


Figura 39: Solicitud de acceso para el API Gateway como cliente público
Fuente: Elaboración propia.

En la solicitud de acceso que realiza un cliente público al servidor de autorización de la **Figura 39**, se destaca:

1. La URI de autorización expuesta por el Servidor de Autorización mediante Keycloak.
2. Parámetro solicitado por el endpoint destinado para el Identificador del cliente.
3. Parámetro solicitado por el endpoint para describir el flujo acorde al tipo de concesión se desea seguir. Estos flujos van acorde al **apartado 2.2.2.2**. El flujo elegido en nuestro caso de prueba es acorde al tipo de concesión por credenciales de contraseña del propietario del recurso.
4. Credenciales del propietario del recurso (Usuario).
5. Observamos que la respuesta del endpoint fue exitosa, con código HTTP 200 y entre los valores de respuesta se encuentra el token de acceso y el token de actualización acorde al **apartado 2.2.2.3**.

Si se toma el valor del token de acceso, podemos realizar una petición de acceso a los recursos protegidos por uno de los proveedores de recursos (microservicios) enviándolo en el *HEADER* de la petición anteponiendo el término ***Bearer*** antes del token de la siguiente manera:

Figura 40: Acceso a recursos protegidos por el microservicio 2 usando un token de acceso
Fuente: Elaboración propia.

Por otro lado, si se selecciona a un cliente como confidencial, automáticamente Keycloak generará un secreto para el cliente similar al de la **Figura 36**. Si repetimos la solicitud de autorización sin enviar el secreto del cliente para un cliente confidencial, el servidor de autorización nos denegará el acceso de la siguiente manera:

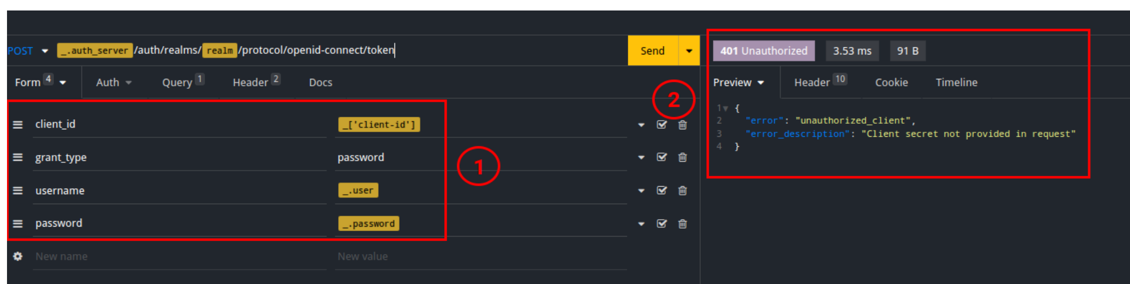


Figura 41: Solicitud de acceso denegado por falta del secreto del cliente
Fuente: Elaboración propia.

De esta forma, Keycloak, como servidor de autorización, cumple con la consideración de seguridad expuesta por el RFC 6819 referente a clientes públicos.

4.2.3 Secretos específicos para despliegues específicos

Continuando con las amenazas de los clientes, una pregunta surge inherente a la arquitectura de OAuth y presentada en el protocolo: ¿Qué pasa si un cliente engaña al Servidor de Autorización?

Por ejemplo, se tiene el siguiente escenario: se ha desplegado un Servidor de Autorización que va a controlar el acceso a un conjunto de recursos protegidos, como los microservicios; sin embargo, se despliegan clientes para múltiples plataformas, entre ellas acceso mediante la web, acceso móvil, y acceso a aplicaciones de escritorio. Al ser clientes oficiales desplegados, cada instancia se encuentra con el secreto del cliente embebido dentro de su código fuente.

Sin embargo, un atacante consigue, mediante métodos de decompilación, acceder el código de una de las aplicaciones donde obtiene el secreto de ese cliente y lo usa en un cliente propio para hacerse de las credenciales de un usuario y/o acceder a los recursos protegidos.

Esta amenaza se describe como “Clientes maliciosos obteniendo autorización” descritas en el **apartado 2.3.5.4.2** y otros. Una de las consideraciones de seguridad del RFC 6819, concretamente la del apartado 5.2.3.4 de dicho documento, nos indica que una manera de mitigar esta amenaza se consigue mediante la creación de secretos del cliente para despliegues específicos (RFC-6819, 2013). Es decir, si se compromete el secreto del cliente que fue desplegado en la web, se deshabilita este secreto sin comprometer los clientes desplegados para otras plataformas.

Para ello, hemos encontrado una limitante en keycloak: la posibilidad de crear múltiples secretos para un mismo cliente ya que Keycloak genera un secreto único para cada cliente registrado. Sin embargo, esta limitante pudo ser superada creando varios registros para el cliente.

Por ejemplo, se crea un cliente que sea destinado para el acceso del Gateway mediante el cliente de Insomnia, de la siguiente manera:

Clients › Add Client

Add Client

Import | Select file

Client ID * | gateway_for_insomnia

Client Protocol | openid-connect

Root URL | http://localhost:8090

Save | Cancel

Figura 42: Registro de un cliente para el API Gateway para Insomnia
Fuente: Elaboración propia.

Ahora, suponemos que este cliente venía operando con normalidad. Tiempo después su integridad ha sido expuesta debido a un problema de seguridad, donde se determinó que su acceso ya no es confiable. El siguiente paso será

deshabilitar el cliente específico desde nuestro servidor de Keycloak de la siguiente manera:

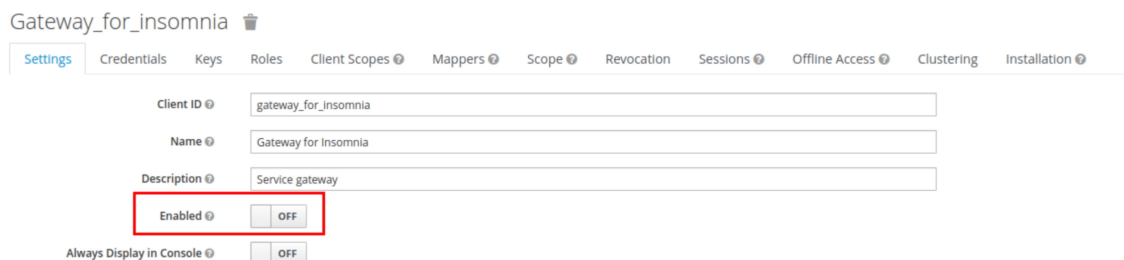


Figura 43: Deshabilitado de un cliente específico
Fuente: Elaboración propia.

De esta manera, cualquier petición realizada por el cliente deshabilitado, dará como resultado:

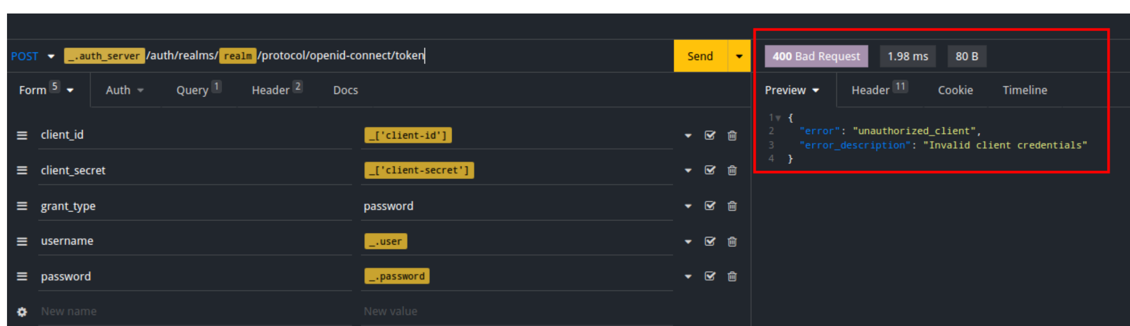


Figura 44: Solicitud de autorización rechaza por un cliente deshabilitado
Fuente: Elaboración propia.

Cabe recalcar que esta acción también dará como resultado que cualquier token de acceso o de autorización previamente generados, queden automáticamente inválidos.

4.3 Seguridad de los usuarios

Los usuarios son los dueños de los recursos dentro de la arquitectura de OAuth. El documento RFC 6819 también aborda amenazas y consideraciones de seguridad relativas a estos usuarios.

4.3.1 Número de intentos de acceso permitidos

Una amenaza presente en los usuarios son los ataques de fuerza bruta. En el documento la amenaza es denominada como “Adivinando las credenciales de los usuarios” que se expone en la **Tabla 9**. Esta amenaza se mitiga mediante la limitante del número de intentos permitidos de acceso fallido, cuyo exceso genera el bloqueo de la cuenta por cierto número de intentos. Esta consideración de seguridad es expuesta bajo el apartado 5.1.4.2.3 del documento RFC 6819.

Para ello, Keycloak nos otorga una pantalla de administración donde podemos habilitar este comportamiento usando los parámetros:

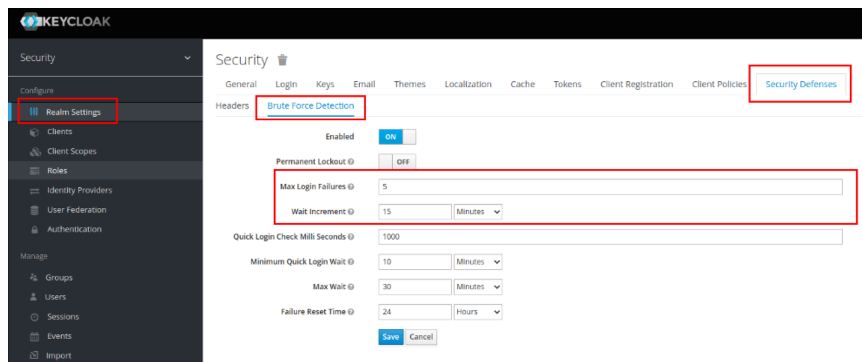


Figura 45: Configuración de seguridad contra ataques de fuerza bruta
Fuente: Elaboración propia.

- **Max Login Failures:** número de intentos fallidos de acceso permitidos antes de un bloqueo.
- **Wait increment:** si el parámetro de número de intentos fallidos ha sido excedido, el tiempo que el usuario debe esperar para ser desbloqueado.
- Los **parámetros posteriores** son usados para intentos rápidos, es decir cuanto el intento es demasiado rápido, podemos sospechar de un bot intentado ataques de fuerza bruta, por lo que keycloak nos permite tratarlos de manera especial.

Si este parámetro es alcanzado, cualquier intento de acceso posterior desencadenará en una acción similar a la de la **Figura 44**.

5. Conclusiones y trabajos futuros

Durante el desarrollo del presente trabajo, se han planteado sus objetivos y se han abordado un conjunto de capítulos acorde a los mismos. De igual forma, se ha desarrollado un conjunto de microservicios que hacen uso del protocolo OAuth2.0 para autenticarse y conseguir autorización. Finalmente se procedió a configurarlos acorde a las consideraciones de seguridad del documento RFC 6819.

Durante todo este proceso, podemos concluir con los siguientes enunciados:

- Los microservicios son, dado el auge actual del cloud computing, de amplio uso de las empresas en que buscan unidades lógicas de negocio en lugar de sistemas únicos que cumplen múltiples funciones.

- Las infraestructuras de microservicios traen consigo un conjunto de retos que los equipos deben solventar; entre ellos podemos destacar la seguridad; en especial, la autenticación y autorización de los mismos.

- El framework OAuth 2.0 se generó a partir de un conjunto de retos y buenas prácticas obtenidas en el uso e implementación OAuth1.0 en entornos empresariales diversos.

- El framework OAuth 2.0 trajo consigo un conjunto de debilidades y amenazas que son analizadas en el documento RFC 6819. En este mismo documento se exponen un conjunto de consideraciones de seguridad enfocadas en fortalecer el protocolo mitigando dichas amenazas.

- El protocolo OAuth, en los entornos de microservicios, puede ser implementado de diversas maneras acorde a los intereses del negocio. Los equipos de desarrollo decidirán el mejor enfoque.

- El diseño y desarrollo del entorno de microservicios debe tener en cuenta la arquitectura del protocolo OAuth para decidir el mejor enfoque de aplicación del mismo.

- Existen múltiples amenazas de seguridad expuestas en el documento RFC 6819 que aprovechan la naturaleza del protocolo OAuth para comprometer la seguridad del entorno.

- Existe un amplio conjunto de consideraciones de seguridad expuestas en el documento RFC 6819 a tener en cuenta para asegurar el uso del protocolo OAuth.

- Keycloak es un servidor de entorno que brinda una interfaz de administración con capacidad de implementar las consideraciones de seguridad expuestas en el documento RFC 6819.

- La sola implementación de Keycloak no brinda un entorno seguro de autenticación. Se requiere configuración adicional y políticas que permitan asegurar el mismo.

- Si bien Keycloak no implementa de manera textual todas las consideraciones del documento RFC 6819, nos permite adaptar su funcionamiento para cumplir con las mismas.

- No se ha conseguido configurar el 100% de consideraciones de seguridad del documento RFC 6819, en este presente trabajo, debido a su extensión; sin embargo, es un buen inicio para trabajos posteriores. Las configuraciones implementadas se resumen en la siguiente tabla:

APARTADO DEL TFM	AMENAZA	IDENTIFICADOR EN RFC	DESCRIPCION
2.3.5.1	Obteniendo los secretos del cliente	5.2.3.1	No compartir secretos con clientes públicos o clientes con políticas inapropiadas
		5.2.3.4	Secretos de clientes para despliegues específicos.
2.3.5.3	Adivinando los secretos del cliente	5.1.4.2.2	Usar alta entropía en los secretos
		5.1.4.2.3	Bloqueo de la cuenta después de cierto número de intentos
2.3.5.4.1	Clientes malicioso obteniendo autorización	5.2.3.4	Secretos de clientes para despliegues específicos.
	Fuga del Código de Autorización mediante un cliente falso		
	Sustitución de código	4.4.1.13	Clientes deben indicar sus identificadores de cliente en cada solicitud de token
2.3.5.4.2	Clientes malicioso obteniendo autorización	5.2.3.4	Secretos de clientes para despliegues específicos.

Tabla 18: Consideraciones de seguridad implementadas acorde al RFC 6819
Fuente: Elaboración propia.

De la misma forma, se proponen los siguientes trabajos futuros:

- En el presente TFM, se ha desarrollado una arquitectura de microservicios bajo el tipo de concesión de OAuth denominado código de autorización. Como trabajo futuro, se propone el diseño e implementación de arquitecturas de microservicios que hagan uso de los otros tipos de concesión detallados en el **punto 2.2.2.2** haciendo énfasis en la diferencia entre dichas implementaciones.

- En el presente TFM se ha desarrollado una arquitectura de microservicios donde el API GATEWAY es tratado como un cliente de OAuth2.0; mientras que los microservicios son tratados bajo el rol de Proveedores de Recursos del protocolo, de acuerdo a la **Figura 25**. Se propone, a

futuro, la implementación del protocolo OAuth en arquitecturas de microservicios donde el API GATEWAY sea tratado como un proveedor de recursos al igual que los microservicios, siguiendo el flujo de la **Figura 26** y haciendo énfasis en las diferencias entre las mismas.

- Acorde a los objetivos específicos planteados en el presente trabajo, se expone el siguiente mapa de cobertura de objetivos donde se utiliza 3 valores de cobertura: cubierto, parcialmente cubierto, no cubierto. La medida de cobertura de objetivos se muestra acorde a la siguiente tabla:

OBJETIVO	COBERTURA
Identificar el concepto de microservicios, su utilidad e importancia en los entornos Cloud en la actualidad.	CUBIERTO
Conocer el protocolo OAuth2.0 en tanto a su funcionamiento, operación e importancia como un sistema de autenticación y autorización de microservicios.	CUBIERTO
Identificar las distintas debilidades y consideraciones de seguridad del protocolo OAuth2.0 expuestos en el documento RFC 6819.	CUBIERTO
Desarrollar un entorno basado en una arquitectura de microservicios que hagan uso del protocolo OAuth2.0 para autenticación y acceso a recursos protegidos.	CUBIERTO
Configurar el entorno para tener en consideración las recomendaciones de seguridad expuestas en el documento RFC 6819.	PARCIALMENTE CUBIERTO
Exponer las conclusiones del trabajo realizado y estudios futuros que sean necesarios para complementar el tema.	CUBIERTO

Tabla 19: Tabla de cobertura de objetivos
Fuente: Elaboración propia.

Por tal motivo, como trabajo futuro, se propone la configuración del resto de consideraciones de seguridad expuestas en el documento RFC 6819 que no han sido parte del presente TFM por temas de extensibilidad.

6. Glosario

API (Application Programming Interfaces): es un conjunto de definiciones y protocolos que se utilizan para desarrollar e integrar el software de aplicaciones, permitiendo las comunicaciones entre dos aplicaciones mediante un conjunto de reglas.

Backend: es la parte de las aplicaciones que no se puede ver cuya función es acceder a la información que se solicita para devolverla a un usuario final.

Cloud Computing: es un término que hace referencia al uso de servicios a través de la conectividad y a gran escala en internet.

Endpoint: en el contexto del API, es el extremo de una conexión de API, donde se recibe una llamada API.

Framework: es un marco o esquema de trabajo utilizado para base de un desarrollo.

Header: es la cabecera que forma parte de una petición y está destinada a añadir información necesaria para el procesamiento de una petición.

HTTP (Hypertext Transfer Protocol): es el nombre de un protocolo que nos permite realizar una petición de datos y recursos como puede ser el HTML.

HTTPS (Hypertext Transfer Protocol Secure): es la versión segura del protocolo HTTP mediante el envío de datos encriptados.

Protocolo: es un conjunto formal de estándares y normas.

RabbitMQ: es un software de negociación de mensajes de código abierto que funciona como un middleware de mensajería. Implementa el estándar AMQP.

URI (Unified Resourse Identifier): es un identificador de un recurso y lo diferencia mediante un nombre, una ubicación o ambos.

URL (Unified Resource Location): es la dirección única e identifica la dirección web o la ubicación de un recurso único. La URL es un subconjunto de un URI.

ZeroMQ: es una biblioteca de comunicaciones de alto rendimiento orientada a mensajes, destinada a la construcción de aplicaciones distribuidas.

7. Bibliografía

Awati, R., & Wigmore, I. (May de 2022). *WhatIs.com*. Obtenido de WhatIs.com: https://www.techtarget.com/whatis/definition/monolithic-architecture

Blinowski, G., Ojdowska, A., & Przybylek, A. (2022). Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. *IEEEAcess*, 20357 - 20374.

Fowler, M., & Lewis, J. (25 de March de 2014). *martinfowler.com*. Obtenido de martinfowler.com: https://martinfowler.com/articles/microservices.html

Hammer-Lahav, E. (05 de September de 2007). *OAuth*. Obtenido de oauth.net: https://oauth.net/about/introduction/

Hannousse, A., & Yahiouche, S. (2020). Securing microservices and microservice architectures: A systematic mapping study.

Hunter II, T. (2017). *Advanced Microservices: A Hands-on Approach to Microservice Infrastructure and Tooling.* San Francisco: Springer Science+Business Media.

Keycloak. (01 de 01 de 2023). *Keycloak*. Obtenido de https://www.keycloak.org/docs/latest/server_admin/#:~:text=that%20group%20defines.-,realms,the%20users%20that%20they%20control

Kharenko, A. (9 de Octubre de 2015). *Monolithic vs. Microservices Architecture*. Obtenido de articles.microservices.com: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59

Mayer, B., & Weinreich, R. (2017). A Dashboard for Microservice Monitoring and Management. *2017 IEEE International Conference on Software Architecture* (págs. 66-69). Gothenburg, Sweden: CPS.

Newman, S. (2015). *Building Microservices.* Sebastopol: O´Really Media, Inc.

OAuth Core, 1. (4 de December de 2007). *OAuth.* Obtenido de OAuth: https://oauth.net/core/1.0/#anchor3

OAuth Working Group, I. (Octubre de 2012). *OAuth*. Obtenido de OAuth: https://oauth.net/2/

RedHat. (08 de Enero de 2019). *https://www.redhat.com/*. Obtenido de https://www.redhat.com/: https://www.redhat.com/es/topics/api/what-does-an-api-gateway-do

RFC-6749. (October de 2012). *rfc-editor.* Obtenido de rfc-editor: https://www.rfc-editor.org/rfc/rfc6749

RFC-6819. (Enero de 2013). *rfc-editor.org.* Obtenido de rfc-editor: https://www.rfc-editor.org/rfc/rfc6819

Richardson, C. (2022). *https://microservices.io/*. Obtenido de https://microservices.io/: https://microservices.io/patterns/index.html

Sevestre, P. (13 de Julio de 2022). *Baeldung*. Obtenido de Baeldung: https://www.baeldung.com/spring-cloud-gateway-oauth2

Siriwardena, P. (2014). *Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWT.* New York: Springer Science+Business Media.

Spasovski, M. (2013). *OAuth 2.0 Identity and Access Management Patterns: A practical hands-on guide to implementing secure API authorization flow scenarios with OAuth 2.0.* Birmingham: Packt Publishing Ltd.

TheExpressWire. (26 de Julio de 2022). *Digital Journal*. Obtenido de Digital Journa: https://www.digitaljournal.com/pr/cloud-microservices-market-growth-statistics-2022-size-global-share-regional-developments-demand-status-and-cagr-value-forecast-by-top-key-players-till-2028

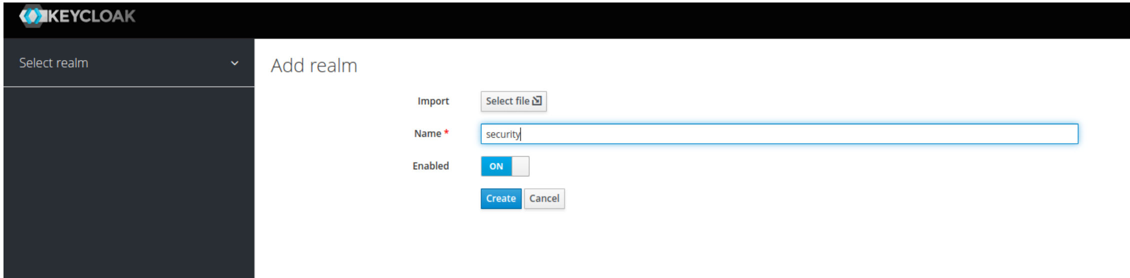
Triartono, Z., Muldina Negara, R., & Sussi. (2019). Implementation of Role-Based Access Control on OAuth 2.0 as Authentication and Authorization System. *Proc. EECSI*, 259 - 263.

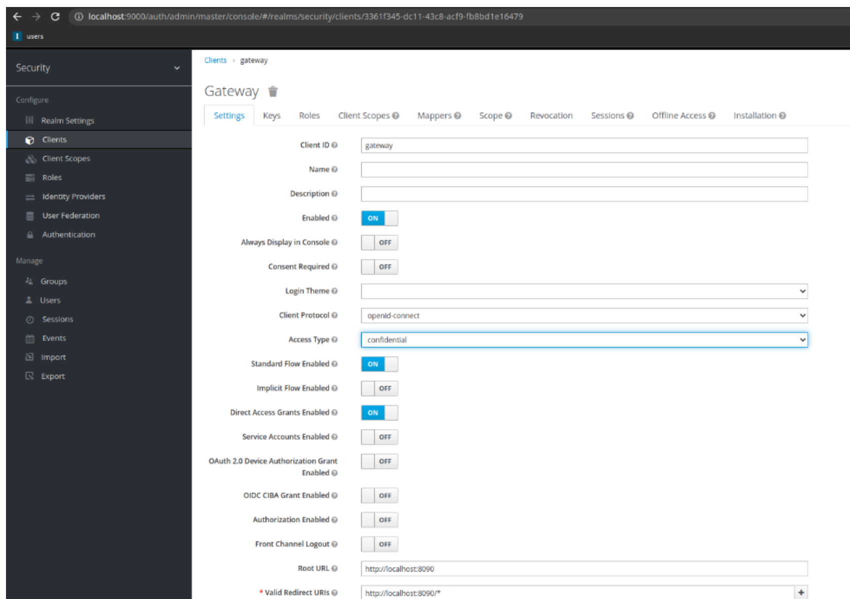
Yang, J., Huo, H., Li, H., & Zhu, Q. (2021). User Fast Authentication Method Based on Microservices. *2021 IEEE International Conference on Power Electronics, Computer Applications (ICPECA)*, 93-98.

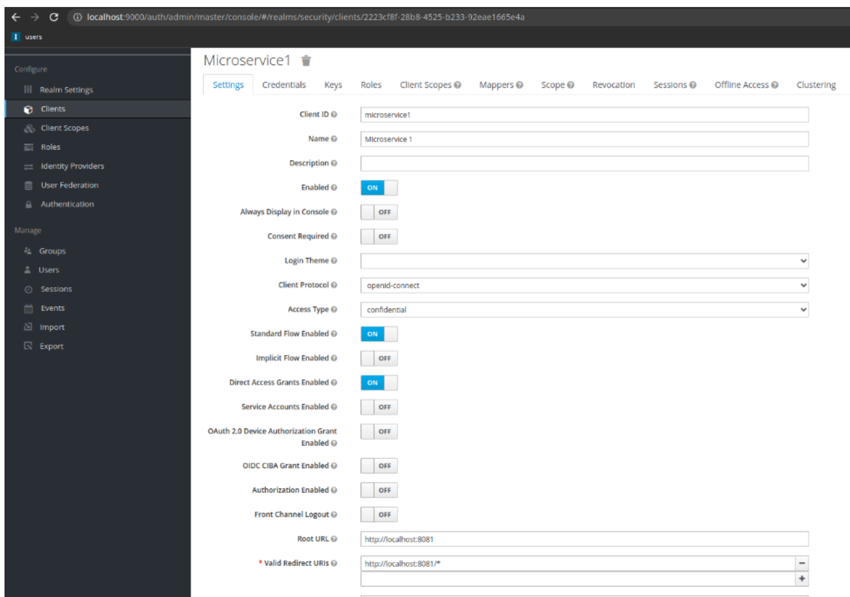
8. Anexos

Anexo 1: Configuración básica del Servidor de Autorización usando Keycloak

El conjunto de configuraciones básicas de Keycloak se conforma por las siguientes capturas:







localhost:9000/auth/admin/master/console/#/realms/security/clients/d0c084c4-e005-475f-82e4-5ee2c61491a1

Security

Configure: Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication

Manage: Groups, Users, Sessions, Events, Import, Export

Clients > microservice2

Microservice2

Settings | Keys | Roles | Client Scopes | Mappers | Scope | Revocation | Sessions | Offline Access | Installation

Field	Value
Client ID	microservice2
Name	
Description	
Enabled	ON
Always Display in Console	OFF
Consent Required	OFF
Login Theme	
Client Protocol	openid-connect
Access Type	confidential
Standard Flow Enabled	ON
Implicit Flow Enabled	OFF
Direct Access Grants Enabled	ON
Service Accounts Enabled	OFF
OAuth 2.0 Device Authorization Grant Enabled	OFF
OIDC CIBA Grant Enabled	OFF
Authorization Enabled	OFF
Front Channel Logout	OFF
Root URL	http://localhost:8082
* Valid Redirect URIs	http://localhost:8082/*

localhost:9000/auth/admin/master/console/#/realms/security/clients/b2425fd9-f8a7-4598-abf8-47568991fd87

Security

Configure: Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication

Manage: Groups, Users, Sessions, Events, Import, Export

Clients > microservice3

Microservice3

Settings | Keys | Roles | Client Scopes | Mappers | Scope | Revocation | Sessions | Offline Access | Installation

Field	Value
Client ID	microservice3
Name	Microservice 3
Description	
Enabled	ON
Always Display in Console	OFF
Consent Required	OFF
Login Theme	
Client Protocol	openid-connect
Access Type	confidential
Standard Flow Enabled	ON
Implicit Flow Enabled	OFF
Direct Access Grants Enabled	ON
Service Accounts Enabled	OFF
OAuth 2.0 Device Authorization Grant Enabled	OFF
OIDC CIBA Grant Enabled	OFF
Authorization Enabled	OFF
Front Channel Logout	OFF
Root URL	http://localhost:8083
* Valid Redirect URIs	http://localhost:8083/*

localhost:9000/auth/admin/master/console/#/realms/security/clients

KEYCLOAK

Success! Your changes have been saved to the client.

Security-admin

Security

Configure: Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication

Manage: Groups, Users, Sessions

Clients

Lookup

Search... | Create

Client ID	Enabled	Base URL	Actions		
account	True	http://localhost:9000/auth/realms/security/account/	Edit	Export	Delete
account-console	True	http://localhost:9000/auth/realms/security/account/	Edit	Export	Delete
admin-cli	True	Not defined	Edit	Export	Delete
broker	True	Not defined	Edit	Export	Delete
gateway	True	http://localhost:8090http://localhost:8090	Edit	Export	Delete
microservice1	True	http://localhost:8081http://localhost:8081	Edit	Export	Delete
microservice2	True	http://localhost:8082http://localhost:8082	Edit	Export	Delete
microservice3	True	http://localhost:8083http://localhost:8083	Edit	Export	Delete
realm-management	True	Not defined	Edit	Export	Delete
security-admin-console	True	http://localhost:9000/auth/admin/security/console/	Edit	Export	Delete