UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Identity Management in Cloud with Entra ID and OAuth 2.0 Flows

Bachelor's Thesis

Bachelor's Degree in Telecommunication Technologies and Services Engineering

AUTHOR: Navarro Rueda, Pablo

Tutor: López Patiño, José Enrique

ACADEMIC COURSE: 2023/2024

Resumen

En el contexto actual del mundo empresarial, marcado por una creciente migración hacia entornos cloud y el abandono gradual de las infraestructuras tradicionales de centro de datos, la gestión efectiva de identidades y accesos se presenta como un desafío crucial para las organizaciones y los profesionales de seguridad.

Este trabajo de fin de grado pretende abordar el reto comenzando por una introducción al entorno cloud mostrando la evolución de las organizaciones y su transición de una gestión de las identidades en directorio activo a realizarla a través de Entra ID. Se llevará a cabo un análisis detallado de las diversas herramientas que Entra ID proporciona, explorando sus funcionalidades y su aplicación para lograr una gestión eficaz de identidades, abordando aspectos claves como la seguridad.

Finalmente, estudiaremos los diferentes flujos de autenticación OAuth 2.0 que existen clasificándolos en base a su seguridad. Concluiremos examinando exhaustivamente un caso de uso: una aplicación que emplea un flujo considerado no seguro para acceder a datos de riesgo o de alta seguridad de la organización. Con base en este análisis, se propondrán diversos métodos y estrategias que aprovechen las herramientas ofrecidas por Entra ID para mitigar adecuadamente estos riesgos y fortalecer la seguridad de las identidades en el entorno cloud.

Palabras clave: ciberseguridad; Entra ID ; Azure ; computación en nube ; gestión de identidades ; OAuth ; Acceso seguro.

Resum

En l'actual escenari empresarial, marcat per una creixent migració cap a entorns cloud i l'abandonament gradual de les infraestructures tradicionals de centres de dades, la gestió efectiva d'identitats i accessos es presenta com un repte crucial per a les organitzacions i els professionals de la seguretat.

Aquest treball de fi de grau pretén abordar aquest repte començant per una introducció a l'entorn cloud, mostrant l'evolució de les organitzacions i la seua transició d'una gestió d'identitats en Directori Actiu a realitzar-la a través d'Entra ID. Es durà a terme una anàlisi detallada de les diverses eines que proporciona Entra ID, explorant les seues funcionalitats i la seua aplicació per a aconseguir una gestió eficaç d'identitats, abordant aspectes clau com la seguretat.

Finalment, estudiarem els diferents fluxos d'autenticació OAuth 2.0 que existeixen, classificant-los en funció de la seua seguretat. Conclourem examinant de manera exhaustiva un cas d'ús: una aplicació que utilitza un flux considerat no segur per a accedir a dades de risc o de seguretat alta de l'organització. A partir d'aquesta anàlisi, es proposaran diversos mètodes i estratègies que aprofiten les eines oferides per Entra ID per a mitigar adequadament aquests riscos i enfortir la seguretat de les identitats en l'entorn cloud.

Paraules clau: ciberseguretat; Entra ID; Azure; computació en núvol; gestió d'identitats; OAuth; accés segur.

Abstract

In the current business landscape, marked by an increasing migration to cloud environments and the gradual abandonment of traditional data center infrastructures, effective identity and access management is emerging as a crucial challenge for organizations and security professionals.

This final degree project aims to address this challenge by starting with an introduction to the cloud environment, showcasing the evolution of organizations and their transition from managing identities in Active Directory to doing so through Entra ID. A detailed analysis of the various tools provided by Entra ID will be conducted, exploring their functionalities and application for achieving effective identity management, addressing key aspects such as security.

Finally, we will study the different OAuth 2.0 authentication flows, classifying them based on their security. We will conclude by thoroughly examining a use case: an application that employs a flow considered insecure to access high-risk or high-security data within the organization. Based on this analysis, various methods and strategies will be proposed that leverage the tools offered by Entra ID to adequately mitigate these risks and strengthen identity security in the cloud environment.

Keywords: cybersecurity; Entra ID; Azure; cloud computing; identity management; OAuth; secure access.

.

EXECUTIVE SUMMARY

The Bachelor's thesis of the Degree in Telecommunications Technology and Services Engineering must develop the following concepts in the text, duly justified and discussed, focused on the field of telecommunications engineering:

CONCEPT (ABET)	**CONCEPTO (traducción)**	**Complies? (Y/N)**	**¿Where? (pages)**
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	Y	1
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	Y	11-31
1.3. Setting of goals	1.3. Establecimiento de objetivos	Y	1
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	Y	32 - 45
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	Y	46 - 49
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	Y	51
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	Y	51

Index

Illustrations index

Chapter 1. Introduction

1.1 Motivation

This project arises from the growing interest in cybersecurity and as a way to test what I have learned in the field of identity management during my internship at NTT Data. During this period, I used the Entra ID tool alongside professionals in the field who shared their enthusiasm and encouraged me to keep learning.

This Final Degree Project is the best opportunity to test my knowledge of the tools used, their practical application, and to demonstrate the importance of securing digital identity in a web application.

1.2 Objectives

The main objective of this work is to delve into the field of digital identity management in the cloud, focusing on the importance of securing identity and using a secure OAuth 2.0 flow. The specific goals are:

- Explore in detail the tools provided by Entra ID, understanding their functionality and utility.
- Analyze the features of different OAuth 2.0 flows and the potential security vulnerabilities that could arise from choosing a non-recommended flow.
- Demonstrate through a practical case how to address these vulnerabilities and use Entra ID tools to secure access.
- Contribute to the knowledge and understanding of the importance of identity management, as well as the implementation of strong security practices to create a secure environment.

By achieving these objectives, the aim is not only to expand theoretical understanding of identity management but also to provide practical recommendations and concrete solutions to address security challenges associated with authentication and authorization in cloud environments.

1.3 Report structure

The project consists of 7 chapters:

- **Introduction**. General presentation of the project, providing initial context.
- **Cloud Environment**. Introduction to the fundamentals of the cloud environment.
- **Evolution of Identity Management**. Exploration of identity management before Entra ID and the transition to its current state.
- **Entra ID**. Overview of the various management capabilities offered by Entra through its different tools.
- **OAuth 2.0 Flows**. Analysis of what an OAuth 2.0 flow is, how it works, and its different types.
- **Use Case**. Detailed description of a practical case in which a web application is integrated with Entra ID, ensuring a secure process.
- **Conclusions**. Analysis of the results obtained.

Additionally, there are appendices with relevant information:

- **Glossary of Terms**. Technical terminology used throughout the report.

- **Sustainable Development Goals**. Information on how the project contributes to the SDGs.

1.4 NTT DATA

NTT Data is a telecommunications and information technology services company based in Japan, employing approximately 300,000 people worldwide. It specializes in information systems and IT consulting. In 2006, NTT Data acquired Everis, a Spanish consulting firm, to expand its presence in Europe, completing the merger in 2021 and establishing NTT Data as one of the leading consulting firms in Spain.

In 2024, the company was recognized as a Global Top Employer for its personnel management policies and practices worldwide, earning certifications in Asia-Pacific, Europe, Latin America, and North America. In Spain, it has been awarded Top Employer status for the ninth consecutive year. These accolades highlight NTT Data's commitment to innovation, quality, and customer satisfaction, positioning it as a leader in telecommunications and information technology.

Chapter 2. Cloud environment

Cloud computing has completely transformed how organizations access, store, and manage their computing resources. It is not merely a service delivered over the Internet providing traditional IT infrastructure like virtual machines, storage, and databases, but also a gateway to a wide range of innovations.

The cloud enables the deployment and scaling of advanced technologies such as the Internet of Things (IoT), Machine Learning, and Artificial Intelligence (AI). These capabilities not only drive creativity and innovation but also provide organizations with numerous benefits, allowing them to explore new horizons and solve complex problems more efficiently and effectively.

2.1 Benefits of cloud migration

The migration to the cloud has become a priority for many organizations in recent years as they seek to modernize and leverage the benefits of this digital environment. Below are the key advantages of migration:

2.1.1 High availability

In a cloud environment, the infrastructure is designed to minimize downtime, ensuring resources are always accessible without interruptions. Cloud service providers like Azure offer uptime guarantees through service level agreements, giving organizations confidence that their services will remain available even during unforeseen events, enabling them to operate with confidence and ensure business continuity.

2.1.2 Scalability

Resources in a cloud environment are scalable, meaning they can be dynamically adjusted to meet changing demands. This allows organizations to handle traffic spikes without compromising performance and optimizes costs through the "Pay-as-you-go" model, where only used resources are billed, eliminating expenses from unnecessary contracted resources.

2.1.3 Reliability

Reliability refers to a system's ability to recover from failures and continue functioning continuously. The infrastructure is resilient due to its decentralized design, distributing resources across multiple global regions. If one data center experiences downtime, other centers can continue operating, with the system often automatically switching to another region to ensure service continuity without manual intervention.

2.1.4 Predictability

Predictability enables reliable project planning and execution, ensuring efficient and cost-effective operations. Performance predictability helps anticipate the necessary resources to provide a positive customer experience, while cost predictability allows organizations to forecast and control cloud computing expenses. Real-time monitoring and data analysis help identify patterns and improve resource implementation planning.

2.1.5 Security

The cloud adopts a shared responsibility model between the cloud service provider and the client, offering more effective and focused defense. This model ensures agile responses to vulnerabilities and allows security to be tailored to specific requirements. By dividing responsibilities, the provider secures the cloud infrastructure against attacks, while the client focuses on protecting their data and managing access to their applications.

2.1.6 Resource management

Integrated tools and services enable continuous real-time monitoring, ensuring optimal operation and the ability to quickly respond to incidents. Problematic resources are automatically identified, and repair processes can be initiated without manual intervention.

2.2 Deployment models

Cloud computing does not follow a one-size-fits-all approach. The diversity and evolution of business needs have led to the development of various cloud deployment models, each designed to address specific challenges and scenarios.

2.2.1 Private

A private cloud is an infrastructure dedicated to a single entity, offering greater control and privacy compared to a public cloud implementation. Organizations can customize their resources, providing flexibility and adaptability to business requirements. The infrastructure can be hosted on-premises or by an external provider, but always within a dedicated private network, making it ideal for entities requiring high security, such as government and financial institutions.

2.2.2 Public

In this model, computing resources such as servers and storage are owned and managed by a cloud service provider, who handles maintenance, reducing this burden for organizations. Public cloud offers near-unlimited scalability, with resources available on demand to meet business needs at any time. High reliability is ensured through a broad network of servers that guarantee service continuity and resource availability.

2.2.3 Hybrid

A hybrid cloud combines private and public infrastructure, allowing organizations to move data and applications between both environments as needed, offering flexibility and deployment options. This model enables businesses to keep sensitive data in their own data center while running other workloads in a public cloud, enhancing security. It is ideal for organizations with existing IT infrastructure that prefer not to transition entirely to the public cloud due to costs and complexity, opting for a hybrid approach to optimize costs and operations.

2.2.4 Multicloud

This model involves using multiple services from different providers, both public and private. Unlike the hybrid cloud, which combines a private and a public cloud, multicloud allows businesses to choose the most suitable services from various providers for specific tasks or locations. This approach leverages benefits such as data and service redundancy, cost optimization, and the ability to utilize the strengths of different providers. However, it also introduces challenges related to integration, management, and security.

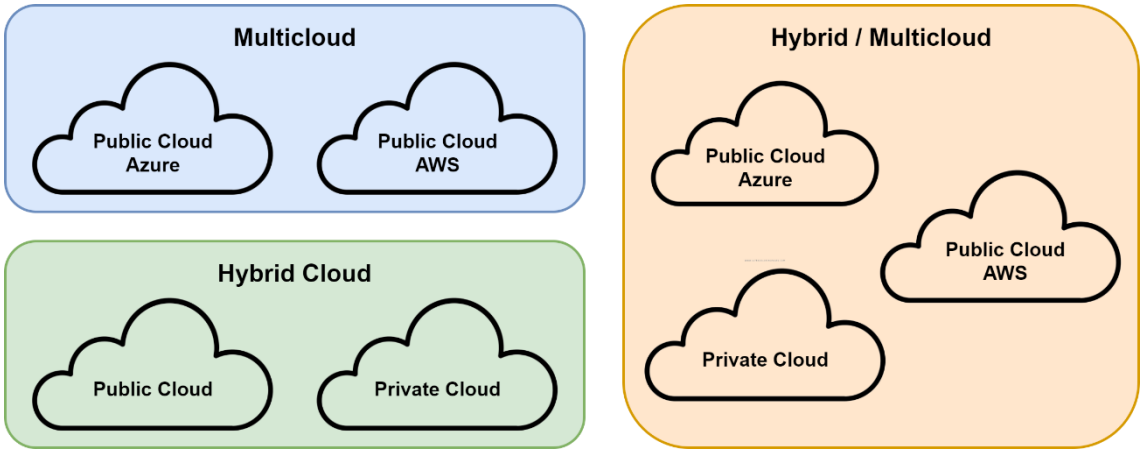

Figure 1 - Hybrid cloud and Multicloud (Own elaboration)

2.3 Service models

In addition to the deployment models, there are cloud service models, classified according to the level of flexibility and control they offer to the client.

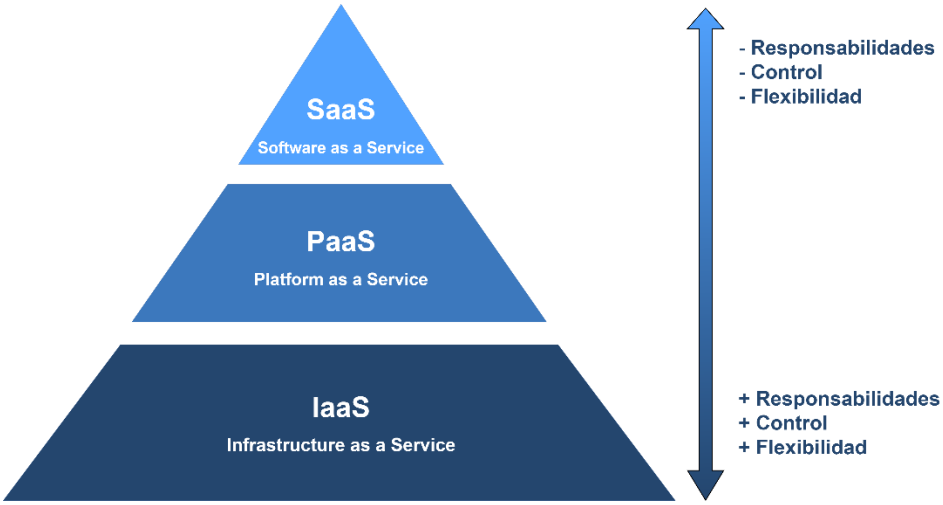

Figure 2 - Cloud service models (Own elaboration)

2.3.1 Infrastructure as a Service (IaaS)

IaaS is the most flexible category, providing a high degree of control over resources. The provider manages the physical hardware, network connectivity, and physical security, while the client is responsible for installing and configuring the operating system, managing the network, and storing data.

With IaaS, organizations rent hardware from a cloud data centre, configuring it to meet their specific needs, enabling efficient resource implementation and scaling without investing in costly physical infrastructure. Virtual machines are a classic example of IaaS, where users have full control over their configuration and management, installing operating systems, software, and applications as needed.

2.3.2 Platform as a Service (PaaS)

PaaS represents an intermediate point between IaaS and SaaS. In this model, the provider manages the physical infrastructure, security, internet connectivity, operating systems, development tools, and business intelligence services.

A key advantage is that users don't need to manage licenses or apply updates and patches, as PaaS provides a ready-to-use platform, allowing development teams to quickly create, test, and deploy applications. A well-known example of PaaS is Microsoft Azure App Service, a fully managed platform that simplifies the creation, deployment, and scaling of web and mobile applications.

2.3.3 Software as a Service (SaaS)

SaaS is the most complete cloud service model from a product perspective, as it involves renting or using a fully developed application. A well-known example of SaaS is Office 365, where users access fully developed cloud-based applications.

Though this model is the least flexible, it is the easiest to implement, requiring the least technical knowledge or experience. Users can access the applications via the internet, without needing to manage the underlying infrastructure or worry about updates or maintenance, as the provider handles all these aspects.

2.3.4 Shared Responsibility Model

After discussing the various service models, it is clear that cloud environments follow a model where responsibilities are divided between the provider and the client, based on the service model used. This approach improves security and data protection, as each party is relieved of certain responsibilities, allowing them to focus on their respective tasks.

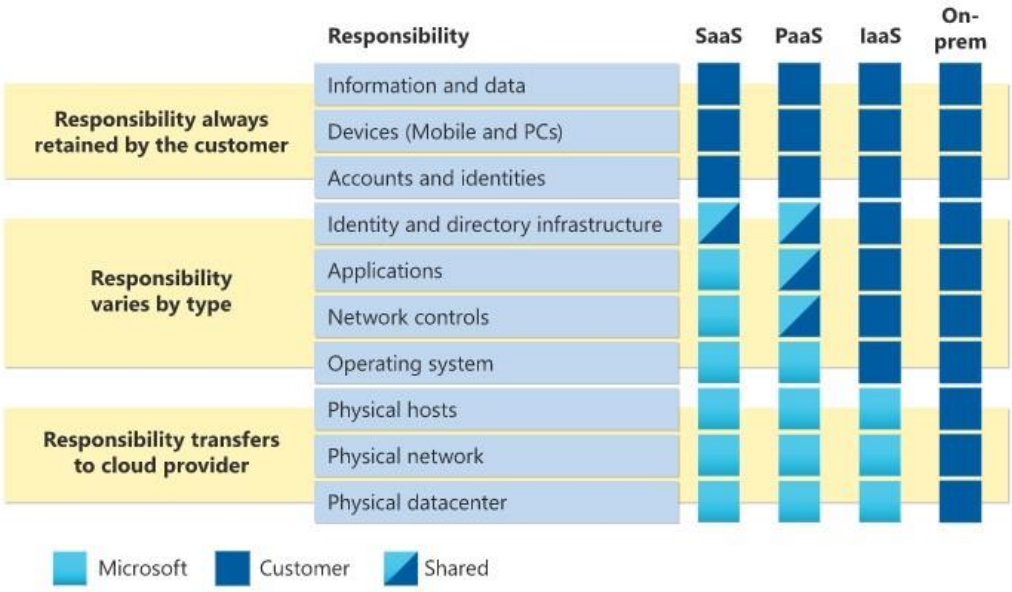

Figure 3 - Shared responsibility model (from Microsoft Learn) [1]

In Illustration 3, the areas of responsibility between the client and provider are shown based on the type of implementation. Starting from the right, it is observed that while in a local data center, the client is responsible for all infrastructure and data security, transitioning to cloud infrastructure shifts some responsibilities to Microsoft.

Regardless of the service model, the client is always responsible for the data, identities, and access management, while the provider is always in charge of physical security of the infrastructure, such as servers and networks. The responsibilities for managing the operating system, applications, data, and identities are then distributed based on the model.

2.4 Major cloud computing providers

Currently, three companies dominate the cloud computing market, each offering a wide range of products and services tailored to different project needs.

Amazon Web Services (AWS):

- Holds the largest market share and number of features.
- Ideal for companies that require proven efficiency.
- Offers great flexibility and ease of use, allowing the selection of operating systems, programming languages, web application platforms, databases, and other services based on specific needs.

Microsoft Azure:

- Holds the second-largest market share, closely rivalling AWS.
- Its main advantage lies in its integration and ease of use for organizations already embedded in the Microsoft ecosystem.

Google Cloud Platform (GCP):

- Has a smaller market share compared to AWS and Azure, but its growth is more significant.
- Known for strong and cost-effective services in data analytics, machine learning, and high-performance computing.

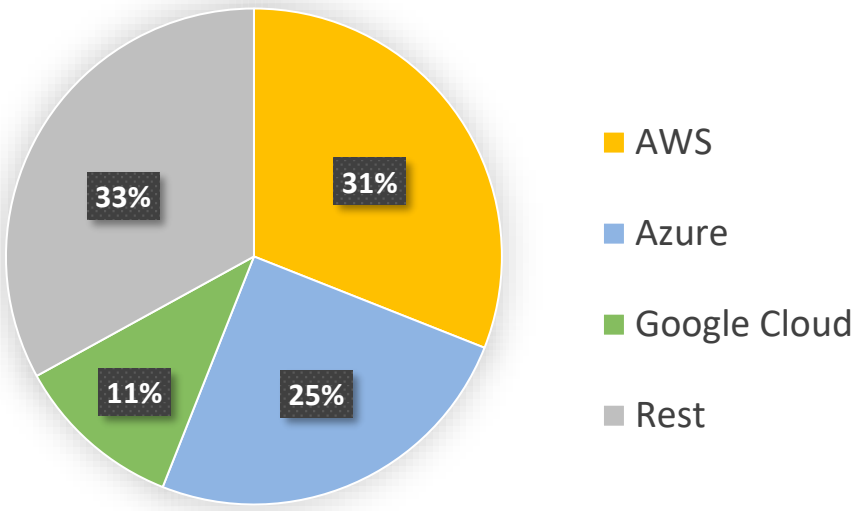

Figure 4 - Market shares, Q4 2023 (data sourced from Statista) [2]

Chapter 3. Evolution of Identity Management

3.1 Digital identity

A digital identity is the unique representation of a user, entity, or device within a system. It consists of information and attributes that allow the identification and authentication of a subject or device within a system, ensuring that only authorized entities have access to specific resources.

These attributes may include the username, email address, login credentials, cryptographic keys, digital certificates, and other data used to verify the identity of the entity. For any identity management system, this is a fundamental component, as it serves as the basis for creating users and assigning groups and roles based on their characteristics.

3.2 Identity Provider (IdP)

An IdP is an entity that creates, maintains, stores, and manages identity information for users and provides authentication and authorization services. In addition, they are responsible for verifying the identities of users through different factors. They not only authenticate human users but can also authenticate any entity connected to a network or system, including computers and other devices.

The role of the IdP is crucial because the digital identity must be securely stored, especially in a cloud environment where the user's identity determines whether or not they can access confidential data.

There are two types of IdPs:

Local IdP:

- Active Directory
- RADIUS
- LDAP

Cloud IdP:

- Entra ID
- Okta
- Google Workspace

In recent years, there has been a growing trend of moving identity management to the cloud, as more and more organizations seek to take advantage of the benefits offered by this environment. As a result, it is becoming increasingly common for organizations to transition from IdPs such as Active Directory to Entra ID. Many organizations in the process of transitioning from one environment to another opt for a hybrid model where both IdPs coexist.

3.3 Active Directory

Windows Active Directory (AD) is Microsoft's on-premises IdP, essential for identity management in Windows-based network environments. This system organizes and manages information through a hierarchical structure that includes user accounts, computers, and other resources, facilitating their access and management.

AD optimizes identity management by storing detailed data about network objects and providing robust services for their efficient search and administration. Security is strengthened through login authentication and access control.

Key tools for effective management within AD include:

- **Schema**: Defines the structures and rules for creating and managing objects within the directory.
- **Global Catalog**: Centralizes information about all directory objects, facilitating access and search across different domains.
- **Query and Index Mechanism**: Enhances visibility and access to network resources through an efficient system for publishing and searching.
- **Replication Service**: Ensures a complete and up-to-date copy of directory data on all domain controllers, maintaining consistency across the organization. [3]

3.4 Entra ID

Microsoft Entra ID can be considered an extension of AD but in a cloud environment. Like AD, it offers identity and access management capabilities, but being cloud-based, users can access resources and services from anywhere at any time, without being limited by the local network infrastructure.

One of the key advantages of Entra ID is Single Sign-On (SSO), which allows users to use a single identity to access a wide range of resources, including cloud applications, internal services, and even mobile apps. This simplifies the login experience and enhances both productivity and security, as users do not need to log in multiple times, making it more difficult for attackers to steal credentials.

Additionally, Entra ID offers advanced security features, such as Multi-Factor Authentication (MFA) and login activity monitoring, to protect business data and resources from cyber threats. With these tools, administrators can maintain granular control over who has access to which resources and apply consistent security policies across both on-premises and cloud infrastructure.

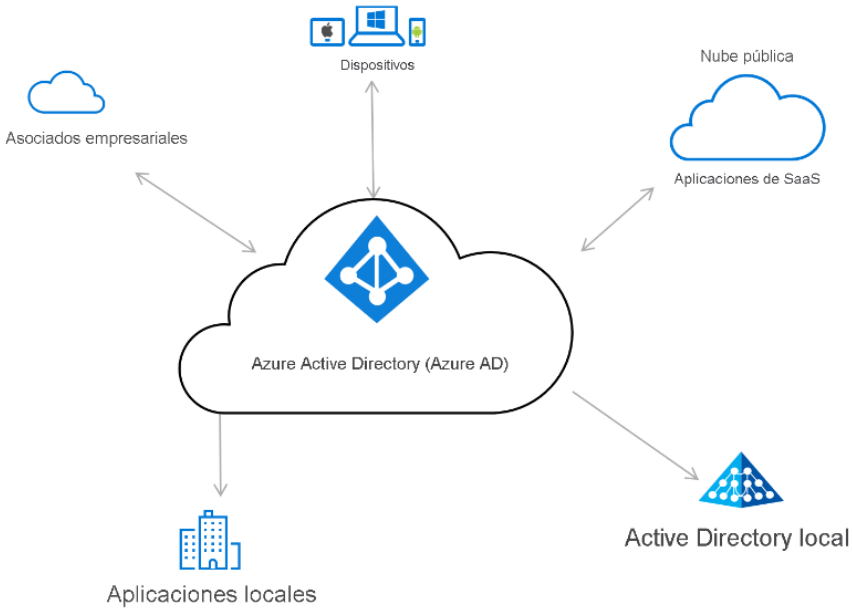

Figure 5 - Entra ID Integrated Environment (by Microsoft Learn 2024)

As shown in Illustration 5, an environment that incorporates Entra ID as the IdP allows for the integration of both on-premises and SaaS applications, which, as previously mentioned, will be accessed via a single sign-on (SSO). Additionally, devices can be registered and hybridized, enabling unified security and access control. It is also possible to establish a hybrid environment

where Active Directory (AD) connects with Entra ID, ensuring that users registered on-premises are automatically synchronized to the cloud. This model achieves centralized identity and access management, improving security and operational efficiency by consolidating administration in a single point.

Chapter 4. Entra ID and Identity Management in a cloud environment

After establishing the foundation of identity management evolution and the transition from AD to Entra ID, it's time to dive deeper into the features, tools, and functionalities that Entra ID offers. This section will cover everything from the licenses required to use specific tools, user management, and group organization, to the implementation of advanced security policies and application management. The goal is to demonstrate how Entra ID drives security, efficiency, and flexibility in the modern enterprise environment.

4.1 Functionality

Entra ID offers a range of licenses that expand the standard functionality of the free directory and provide access to advanced features. The aim is for users to benefit from self-service, enhanced monitoring, security reports, and secure access for mobile users, among other significant improvements.

Some of the most relevant licenses include:

- **Microsoft Entra ID P1 or P2 License**: These licenses grant access to a broader set of features, such as Privileged Identity Management (PIM), which will be discussed later.
- **Microsoft 365, Office 365, and Windows Licenses**: These are assigned to users or groups to provide access to Office or Windows products, and one is required for each user needing access.
- **Monthly Active User (MAU) License**: This license is used with external Entra ID identities and enables tracking external users who sign in, generating monthly reports for billing purposes. It helps track access and associated costs.

4.2 Identity and Access Management (IAM)

Next, we will explore in detail how Entra ID handles user management, group organization, role assignment, and device and IP registration, key aspects for ensuring effective access control and protecting identity.

4.2.1 Users

In Entra ID, a user is a digital identity created to represent an individual in the system and manage authentication and authorization across various resources. Users can be of two types:

- **Members**: These are users who belong to the same organization and have permission to access most of the directory's information.
- **Guests**: These users come from other directories and are B2B (Business to Business) collaborators. They have restricted access in the directory and are typically external collaborators or vendors who need access to specific resources but not the entire infrastructure or billing-related aspects.

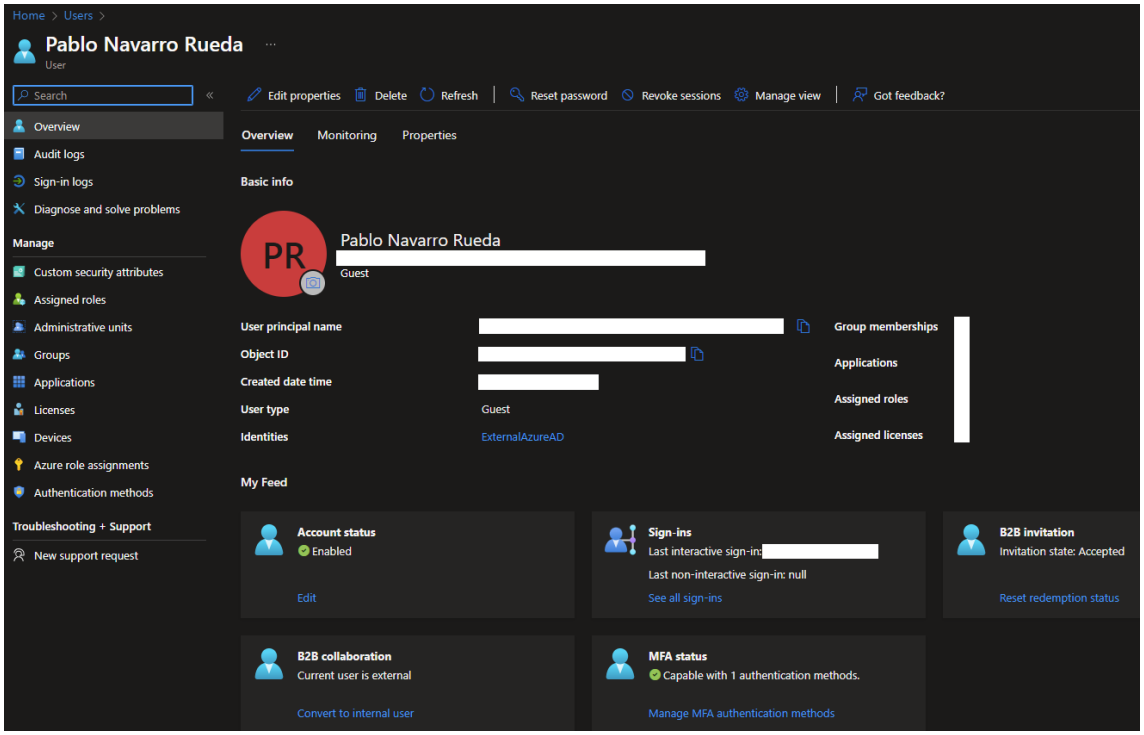

Figure 6 - User in Entra ID (Own elaboration)

4.2.2 Groups

Groups are a key tool for efficiently managing access to resources, applications, and tasks. Instead of managing each user individually, users can be added to a group, allowing administrators to manage all users within the group collectively.

Depending on the group's nature, two types can be distinguished:

- **Security groups**: These groups can include users, devices, services, and other nested groups and are used to manage members' access to shared resources. For example, a group can be used to assign a specific set of permissions to all its members.
- **Microsoft 365 groups**: These groups are intended for collaboration through services like Outlook, OneDrive, Planner, and Teams.

Regardless of their nature, groups can be classified into two types based on how users are added:

- **Static Groups**: Members are manually assigned by an administrator and remain in the group until they are removed. These are useful when the membership of the group does not change frequently.
- **Dynamic Groups**: Members are automatically assigned based on rules tied to user attributes, such as department, location, or role. When any of these user attributes change, the system evaluates the group's rules to determine if the user should be added or removed automatically. These are useful when group membership frequently changes.

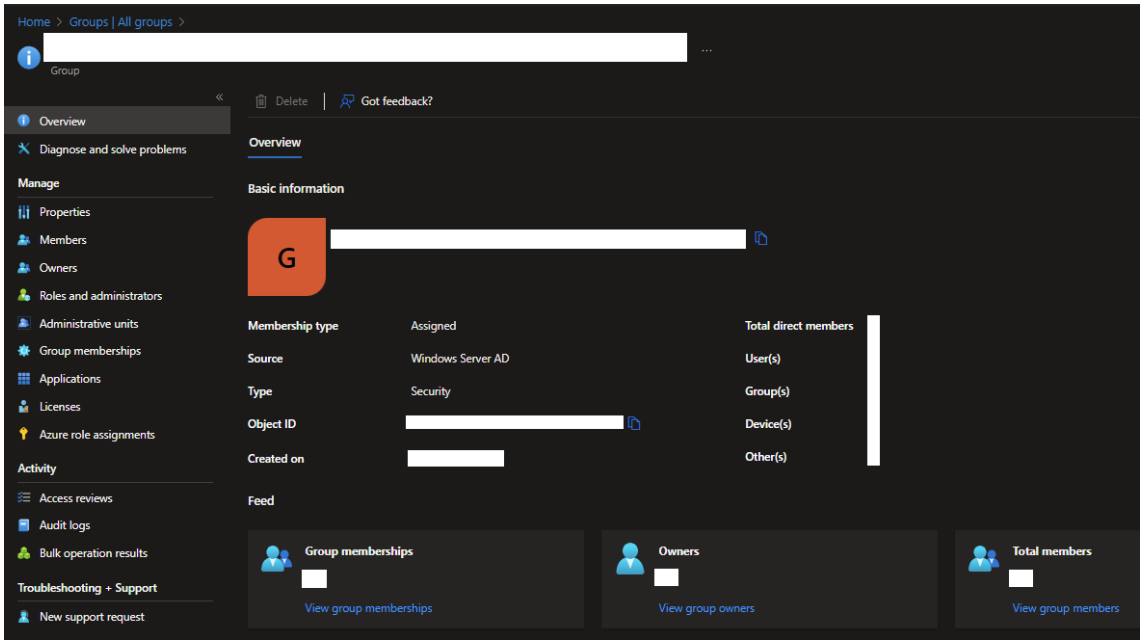

Figure 7 - User in Entra ID (Own elaboration)

4.2.3 Roles

A role represents a set of permissions that determine what actions a user can perform within the Entra ID environment. When a user needs to interact with specific resources, a role is assigned to them, defining their capabilities and limitations based on the associated permissions.

There are two main categories of roles in Entra ID:

- **Built-in Roles**: These are about 60 pre-defined roles provided by Microsoft, which cannot be modified. Each of these roles comes with a fixed set of permissions, allowing users to perform specific tasks within the environment.
- **Custom Roles**: These roles are created and managed by the administrators of the environment. This flexibility allows for the creation of roles that are better suited to the organization's needs and structure by assigning exactly the permissions required. This is useful for avoiding the need for users to have multiple roles simultaneously to perform their daily tasks.

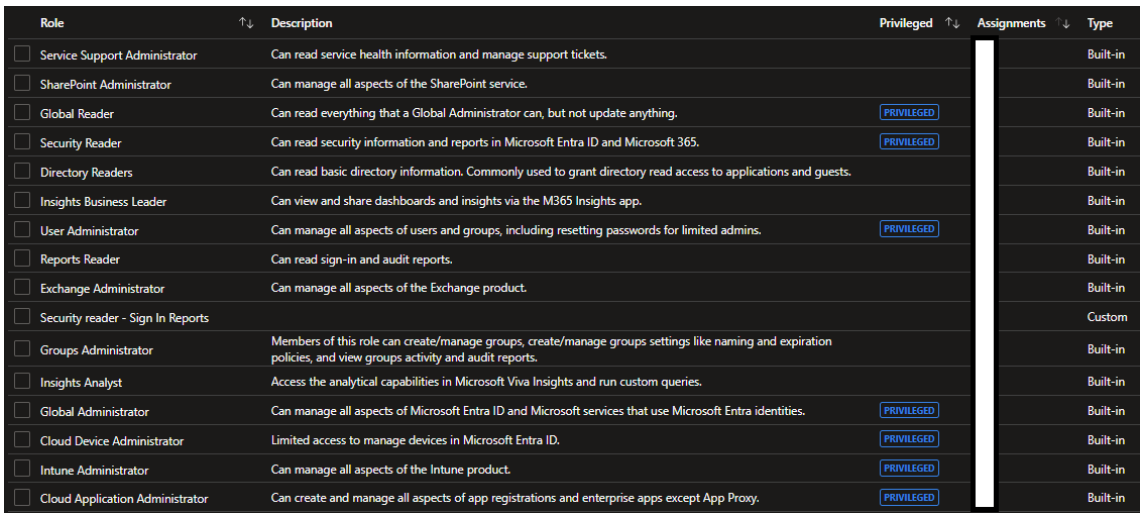

Figure 8 - Entra ID Roles (Own elaboration)

4.2.4 Device and IP registration

As previously mentioned, Entra ID allows not only the registration of users but also of devices. This can be done by registering them directly on the platform or through a hybrid integration. This registration process enables devices to securely connect and authenticate within the environment, facilitating their management and control by administrators.

It also allows the registration of "named locations," which are groups of IPs designated as trusted. By doing so, user logins from these IP ranges can be easily recognized during analysis.

Further, we will explore how these two features enhance security and control within the environment when working in conjunction with the Conditional Access tool.

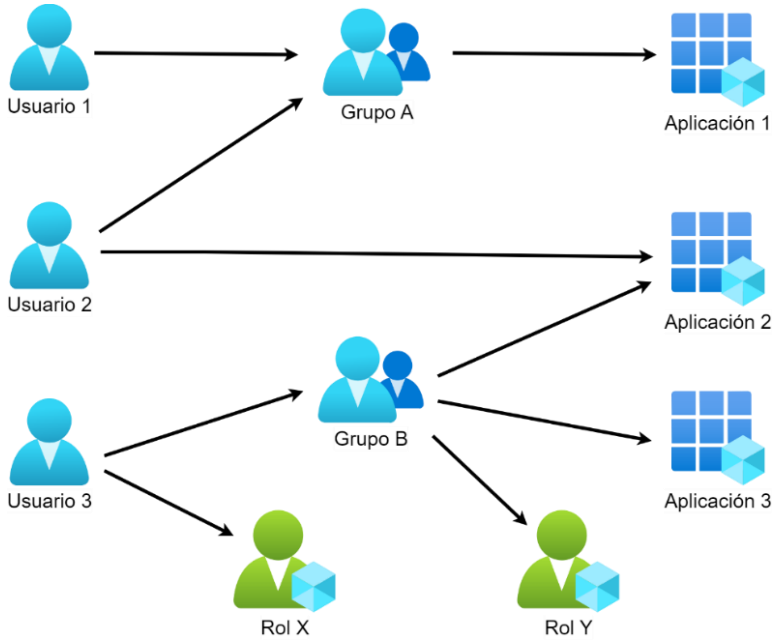

Figure 9 - Interplay of GID areas (Own elaboration)

In Illustration 9, we can see how the studied areas function together. In this representation, User 1 is assigned to Group A, granting exclusive access to Application 1, which is designed specifically for its members. User 2, also part of Group A, is additionally assigned to Application 2, allowing access to both Application 1 and 2. Finally, all users in Group B have access to Applications 2 and 3, with Role Y assigned. User 3, belonging to Group B, also has Role X individually assigned, extending their access permissions within the system.

4.3 Application management

When registering an application, it is configured to recognize Entra ID as its Identity Provider (IdP), meaning authentication and authorization are delegated to the IdP. This allows the application to leverage the authentication and authorization services provided by Entra ID to manage user access. It is essential to establish a secure and efficient connection between both, ensuring seamless interaction. Furthermore, it simplifies the user experience by offering a single access point for all registered applications.

Once registered and configured, the application redirects users to a login portal where they authenticate using their accounts registered in the IdP (in this case, Entra ID). After successful authentication, Entra ID will notify the application whether the authenticated user is authorized to access the resources, based not only on the validity of the credentials but also on the permissions and roles assigned to the user.

4.3.1 App Registrations

App Registration is the formal process of registering an application that uses the OpenID Connect protocol for authentication in a specific Entra ID tenant. This registration serves as a representation of the application within the environment, facilitating communication and connectivity with services provided by the IdP. Once the App Registration is created, a unique identifier known as the Client ID is assigned, exclusive to the tenant where the application is registered.

Through the App Registration menu, the application administrator can manage various key aspects, such as secrets and certificates that add an extra layer of security to the application. It is also where critical configurations are made, such as response and logout URLs, permissions, and access to necessary APIs. These permissions, known as scopes, define what resources the application can request and access on behalf of the user, ensuring controlled and secure access to protected resources.

4.3.2 Enterprise Applications

An Enterprise App is the operational manifestation of the application within the tenant, using SAML as the authentication and authorization protocol. The Enterprise App is closely related to an App Registration and a Service Principal. While an App Registration defines the application itself, providing a unique identifier (Client ID) and configurations for permissions and secrets, the Enterprise App is the operational manifestation of that application within the tenant.

The Service Principal acts as the identity of the Enterprise App within the tenant. It defines the specific permissions and roles that the Enterprise App will have, allowing administrators to manage how the application interacts with other resources and services.

Through the Enterprise Apps menu, the application administrator can define which users will have access to the app and the roles they will possess. They can also access the login attempts of users to maintain control.

4.4 Security tools

Next, we will analyze the security tools incorporated in Entra ID, focusing on those designed to strengthen and protect the environment, as well as those that enable administrators to investigate potential threats.

4.4.1 Multi-Factor Authentication (MFA)

MFA requires users to verify their identity using multiple authentication methods, adding an extra layer of security. When MFA is required for access, users must not only enter their password but also provide a second form of authentication through one of the methods supported by Entra ID. This makes it harder for attackers to gain access to organizational resources, even if they have compromised a user's password. It is a simple but effective measure that helps protect organizations, particularly against phishing attacks or situations where passwords may be stolen.

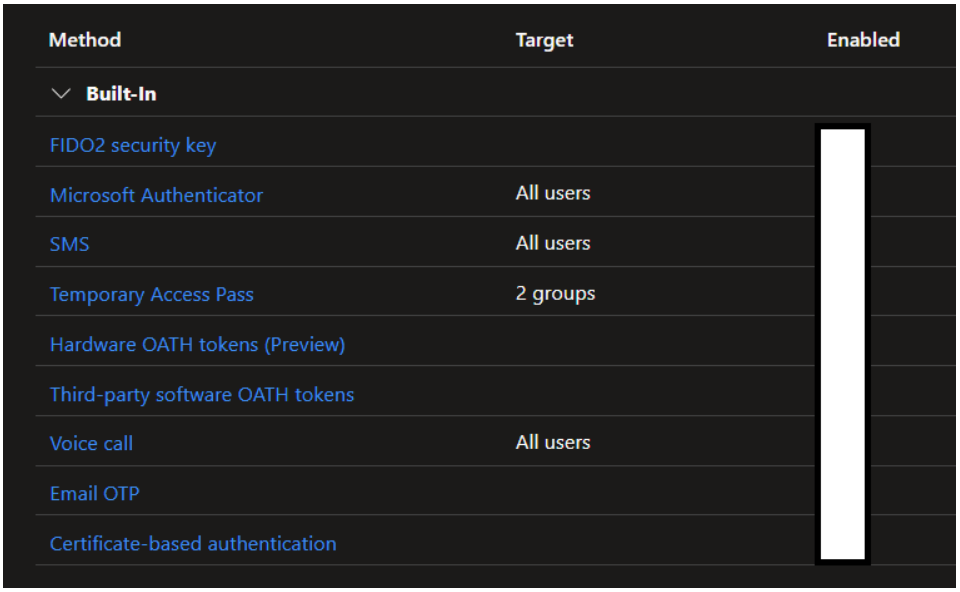

Figure 10 - Authentication methods supported by Entra ID (Own elaboration)

4.4.2 Privileged Identity Management (PIM)

PIM is a service in Entra ID available through P1 or P2 licenses, designed to mitigate risks associated with excessive, unnecessary, or misused access permissions.

A key feature of PIM is its ability to provide time-based and approval-based role activation. This ensures that users can activate privileged roles only when necessary for specific tasks, with access subject to administrator approval. This feature not only reduces the risk of privilege abuse but also limits exposure to potential threats by restricting privileged access to critical times and situations.

PIM also provides advanced auditing and monitoring capabilities, allowing organizations to track role activation in detail. Administrators can identify who activated a role, when it was activated, and what actions were performed during the activation. These detailed logs are presented through reports and analysis, enabling the detection of anomalies and the improvement of access policies.

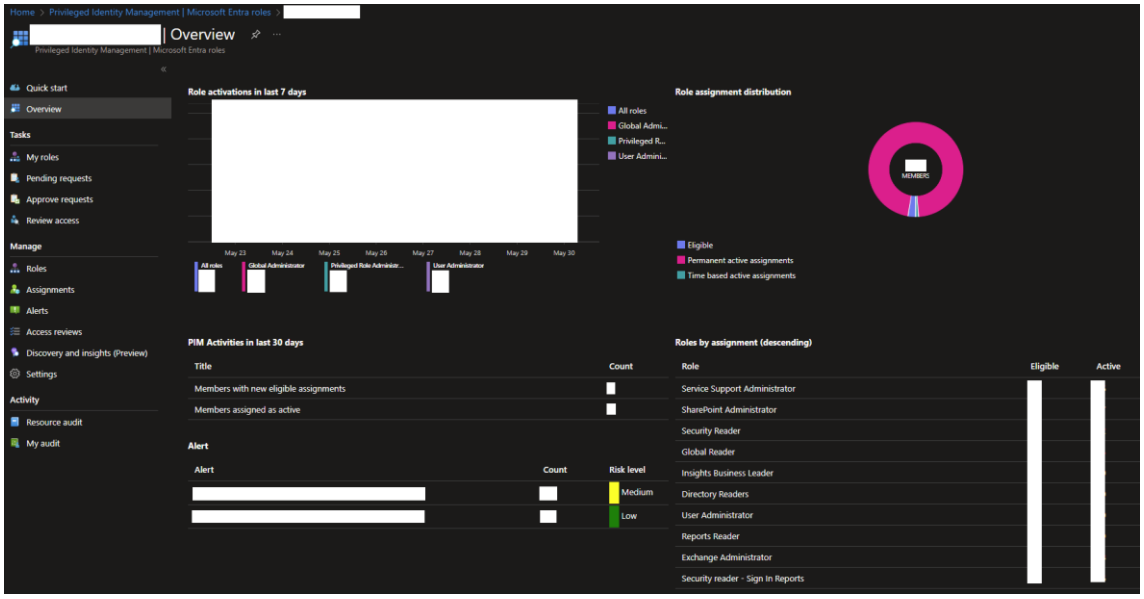

Figure 11 - PIM analysis (Own elaboration)

4.4.3 Risk Detections

Risk Detections are alerts generated by the system when suspicious activity is detected related to user accounts in the directory. These detections are based on continuous analysis of user activity logs, using advanced algorithms from Microsoft to identify patterns and anomalous behaviours that could indicate potential malicious activity or account compromise.

There are two types of risks:

- **Risky Sign-Ins**: A sign-in is classified as risky due to suspicious behaviours, such as logins from anonymous IP addresses, impossible travel to atypical locations, logins from infected devices, logins from IP addresses with suspicious activity, and logins from unknown locations.
- **Risky user**: A user is classified as risky when their account is considered at risk of compromise. This typically occurs when the user has multiple sign-ins marked as risky or when one or more risks are detected on the user's account, such as leaked credentials.

Once a risk is detected, it is classified as low, mid, or high depending on its severity. Administrators must investigate the risk using the corresponding reports and take appropriate actions. These actions may range from dismissing the risk if it is deemed a false positive, requesting a password reset to strengthen account security, or temporarily blocking the user's account if necessary to protect organizational resources.

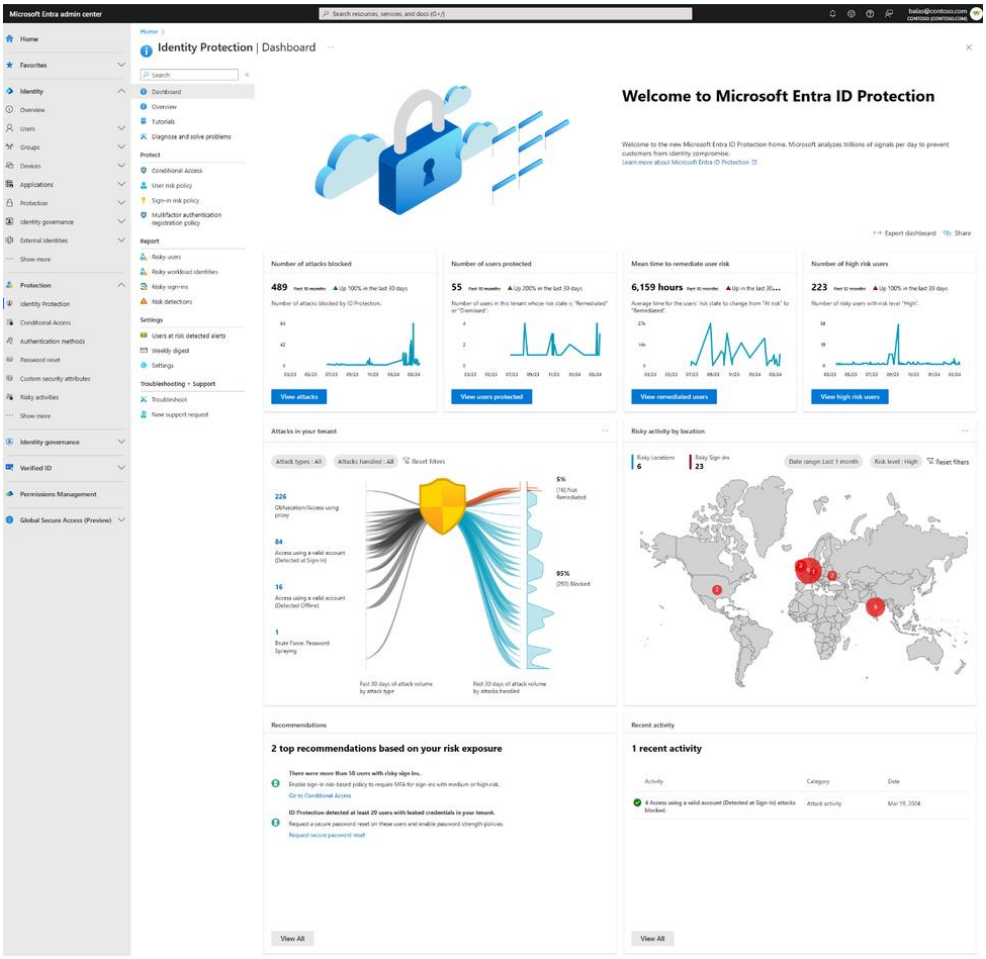

Figure 12 - Risk analysis (by AdminDroid Blog)

4.4.4 Self Service Password Reset (SSPR)

Tool designed to empower users by providing a guided and secure process to reset their passwords using approved authentication methods.

It reduces the workload of the support team, eliminating the need for technical staff to reset passwords for users, such as those flagged as risky. It also minimizes user downtime by enabling quick resolution of password-related issues, allowing them to resume work promptly. SSPR can be configured to align with company security policies, ensuring new passwords meet required complexity and length standards.

4.4.5 Identity Secure Score

Numerical measure evaluating the security level of an environment, expressed as a percentage of compliance with Microsoft's security recommendations. These guidelines are based on best practices, with each having a maximum score. The closer the environment's configuration aligns with these directives, the higher the score achieved.

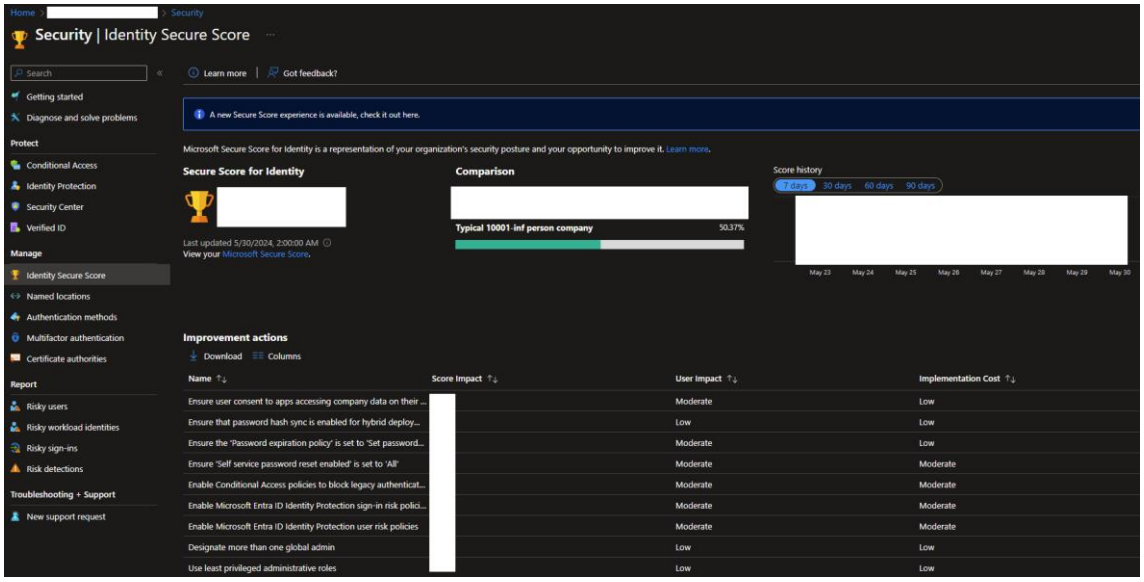

Figure 13 - Identity Secure Score (Own elaboration)

As shown in Illustration 13, Entra ID provides a detailed table of security recommendations and the potential impact of their implementation, helping improve the overall security posture.

4.4.6 Log analysis tools

Entra ID provides various logs tracking actions within the environment, crucial for administrators to identify potentially malicious activities. Two key logs are:

- **Sign-in logs**: Record attempts to log in, including both user-initiated and non-interactive background access.
- **Audit logs**: Record changes made to groups, users, applications, or security policies. Enable proactive threat detection, identification of security issues, and monitoring of unauthorized changes.

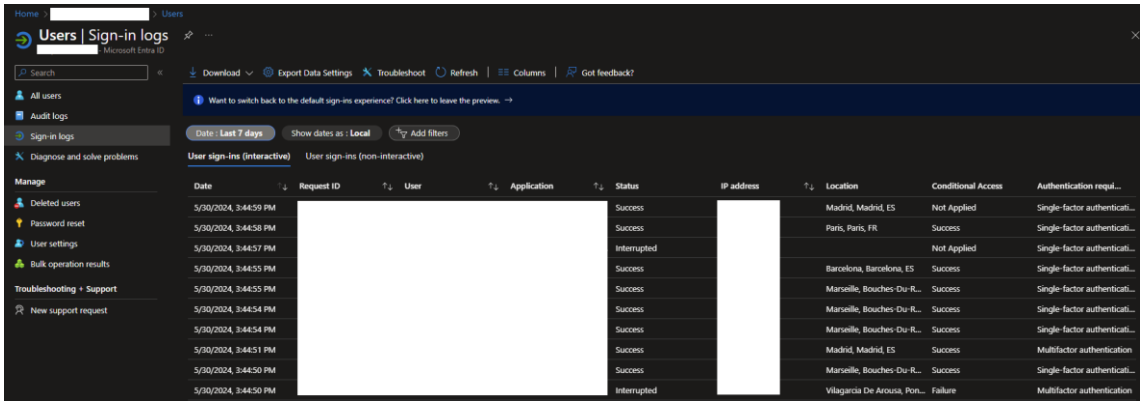

Figure 14 - Sign-in logs (Own elaboration)

Illustration 14 demonstrates how Sign-in logs appear within the environment, resembling the format of Audit logs. For both types, administrators can apply filters to the columns shown in the image, enabling analysis of logs for specific applications or within a defined time frame.

4.4.7 Conditional Access

Conditional Access policies are a key tool for safeguarding access to environment resources. These policies operate on if-then statements, where access is granted or denied based on specific conditions. This enables control and monitoring of who, where, and how users access the organization's resources.

This tool aligns with the "Zero Trust" security model, which assumes no implicit trust for any user or device, whether inside or outside the corporate network. Each access request must be verified and meet at least one authentication condition at every step, ensuring continuous and adaptive protection against potential threats.

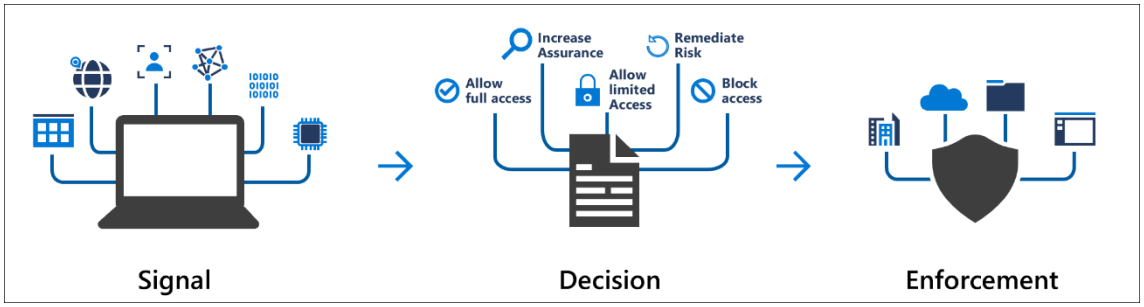

Figure 15 - Conditional Access functionality (by Microsoft Learn) [4]

As shown in Illustration 15, Entra ID evaluates multiple factors to make a decision regarding access to resources:

- **User Context**: Considers group memberships and whether users are internal or external guests.
- **Device**: Policies can include or exclude devices based on trust status, hybrid configuration, or registration.
- **Location**: Assesses whether the request originates from a named location. Administrators can allow or deny access depending on whether the location is deemed trustworthy.
- **Applications and resources**: Policies can be configured to apply based on the specific applications or resources being accessed.
- **Risk**: Allows policy enforcement based on the user's risk level. For example, policies can block access to applications for users identified as high-risk.

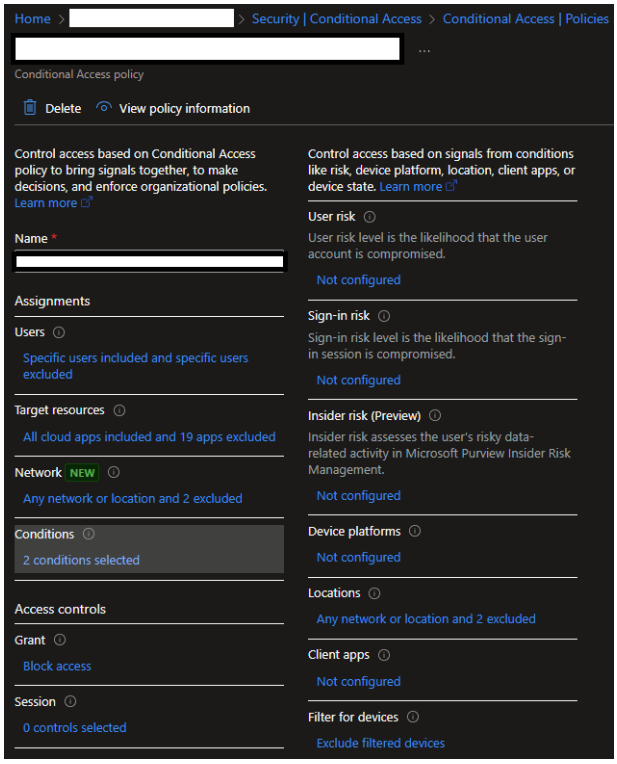

Figure 16 - Conditional Access policy application conditions (Own elaboration)

Based on all the information evaluated when accessing a resource, it is determined which policies should be applied for that access. Policies can take actions such as blocking access to the resource, allowing it directly, or allowing it while requiring an additional action, such as a specific MFA.

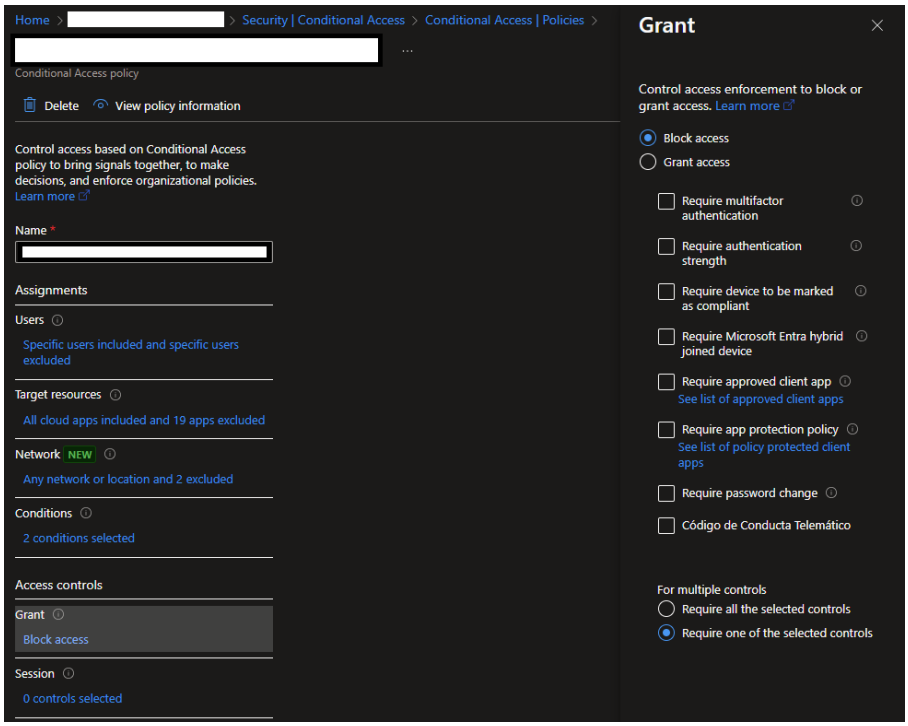

Figure 17 - Actions a Conditional Access policy can perform (Own elaboration)

To maximize the benefits of this tool, it is crucial to create a set of policies within the environment that cover all possible access scenarios, thereby preventing vulnerabilities that attackers could exploit. Policies should complement each other, and administrators must carefully analyze that

everything allowed by one rule is assessed by at least one other rule. By following these recommendations, comprehensive control over system access is achieved, meeting security requirements and ensuring the protection of company resources.

Chapter 5. OAuth 2.0 Flows

After thoroughly analyzing how Entra ID works and the tools it incorporates, it is time to explore OAuth 2.0 flows and understand their utility and operation. To do so, it is essential to grasp two key concepts:

- **Authentication**: The process of confirming that someone is who they claim to be.
- **Authorization**: The process of determining whether a user has access to certain data.

5.1 OAuth 2.0 and OpenID Connect

OAuth 2.0 is a protocol focused on the authorization of applications to access resources on behalf of a user. The protocol ensures access is granted with a defined scope, allowing detailed specification of the permissions granted to the application.

OpenID Connect is an authentication protocol built on top of the OAuth 2.0 protocol, commonly used for authorization in web and mobile applications. Its primary purpose is to add an additional layer over OAuth 2.0, enabling both protocols to manage authentication and authorization, achieving full control over access.

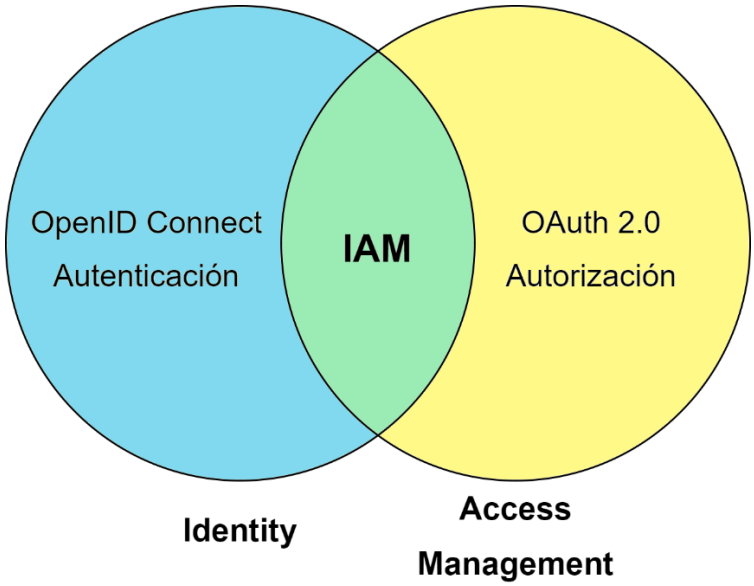

Figure 18 - Joint use of OpenID Connect and OAuth 2.0 (Own elaboration)

The OAuth flows that will be analyzed and later used in the case study address not only authorization but also authentication. This dual focus is essential to protect both identity and access.

5.2 How an OAuth 2.0 Flow works

OAuth 2.0 flows operate through the issuance of access tokens, which are unreadable strings containing information such as the token's duration, the permissions it grants, and its scope, among other details. Tokens act as digital keys, enabling access to protected resources without repeatedly exposing users credentials.

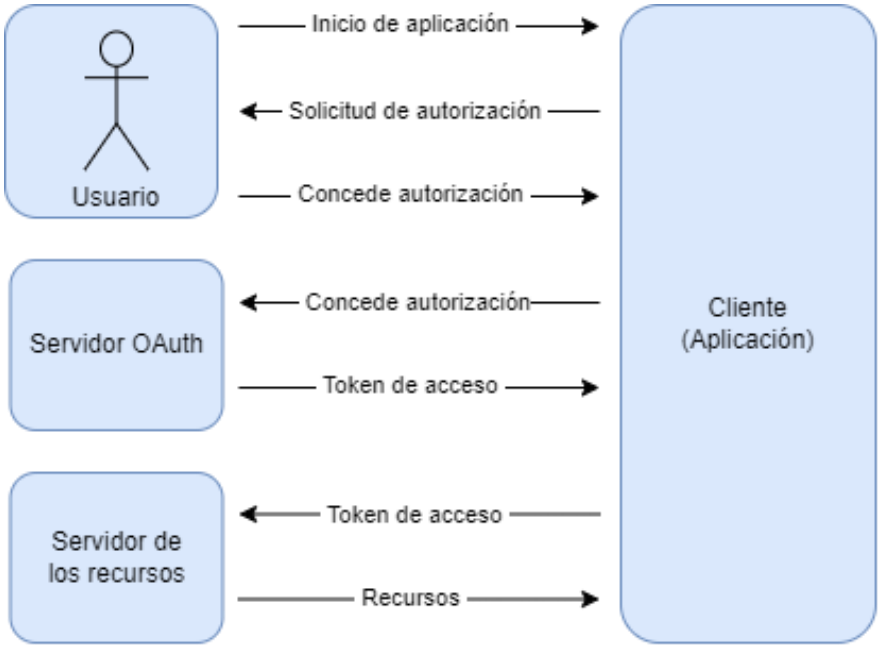

Figure 19 - General operation of an OAuth 2.0 Flow (Own elaboration)

Illustration 19 provides a general overview of the steps followed in a flow. The user accesses the application and is redirected by the app to the login page, where they must authenticate with their credentials. After authentication is completed, the client application requests authorization to access the resources. Once authorization is granted, the application requests the access token from the OAuth server, which is an IdP like Entra ID. The server then issues the token, and with it, the client can request access to resources as many times as necessary before it expires. In the case study, the OAuth server will be Entra ID.

5.2.1 Refresh token

As mentioned, when the user authenticates and authorizes, they receive an access token that can be used as a key to access protected resources. However, for security reasons, the access token has a limited lifespan and expires after that period, requiring the entire process to be repeated to obtain a new one.

The refresh token prevents this by allowing the application to renew the access token without re-entering the user's credentials. This process ensures uninterrupted access to resources, as it is entirely seamless for the use.

Given that the refresh token enables obtaining a new access token, it is crucial to implement appropriate security measures to protect the environment. Good security practices include:

- Storing both tokens in the backend of the application to prevent exposure.
- Encrypting both tokens, both at rest in the backend and during transmission.
- Implementing token rotation, where each time the refresh token is used to obtain a new access token, a new refresh token is issued, and the previous one becomes invalid. This minimizes the impact if a refresh token is compromised.
- Allowing the revocation of both tokens to immediately stop an attack if one of them is compromised.

Figure 20 - Example of tokens (by Okta) [5]

5.2.2 Token request

After completing authentication and authorization, the client requests the access token from the OAuth server. This request is made through an HTTP GET request sent by the client to the server, containing various parameters that include details about the token's scope, desired duration, requester information, and more.

Below are the most relevant and commonly used parameters, as not all are necessary for every request:

- **response_type**: This parameter does not specify any requirement for the access token itself but tells the OAuth server what kind of response is expected. Depending on the flow, the server may need to return different responses. For example, this parameter will be set to "code" when expecting an authorization code or "token" when expecting an access token.
- **scope**: A list of permissions, such as read or write access, required by the client. These permissions must be granted in the access token.
- **client_id**: A unique identifier assigned to the client by the OAuth server during the application's integration with the IdP. Its purpose is to identify the client initiating the flow.
- **redirect_uri**: Indicates the client's URL where the OAuth server should redirect the user after completing the flow. For security, the OAuth server stores this URL during the application's integration and compares it with the value in the request to prevent redirection attacks.
- **state**: A unique and random value used as a security measure by the client. The client generates this random value and includes it in the initial request. If the OAuth server responds with a different value, the client rejects the response as illegitimate.
- **client_secret**: Appears only in flows that use a client secret and is used to authenticate the client to the OAuth server. This value is a shared secret between the client and the server and protects the communication.

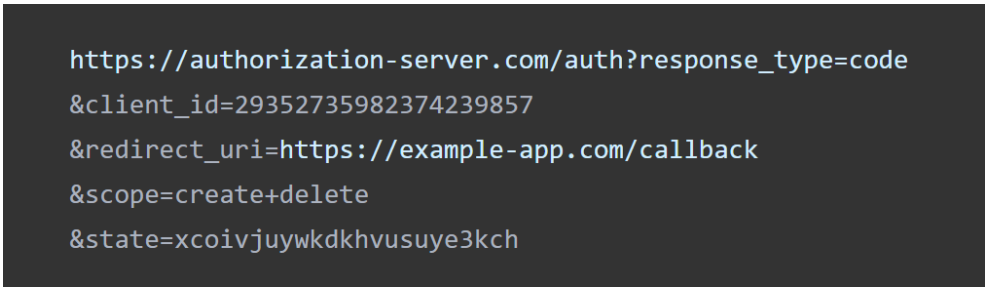

Figure 21 - HTTP Request (by Okta) [6]

5.2.3 Extension flows

OAuth is an open and flexible standard that offers developers the ability to customize authorization flows to meet the specific needs of their applications. This means that developers can create new flows by modifying the standard ones, known as extension flows. This may include integration with existing authentication systems or the implementation of custom authorization logic.

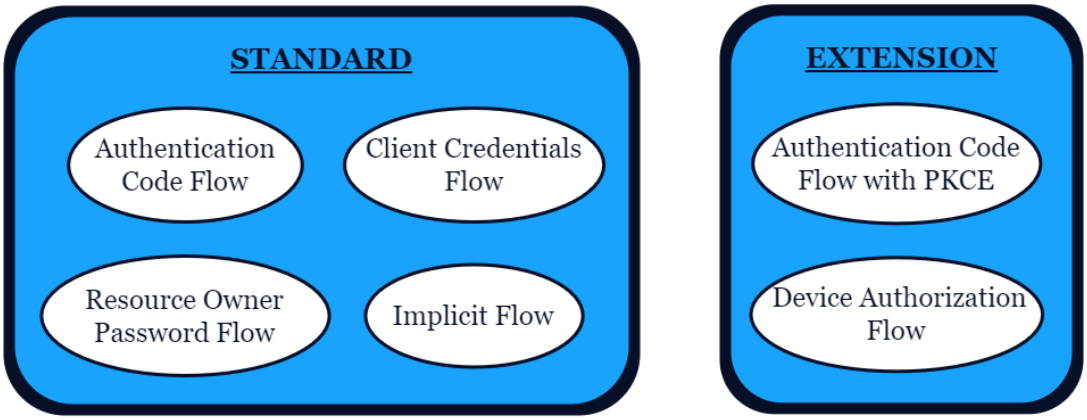

Figure 22 - Standard and Extension flows (Own elaboration)

In Illustration 22, the flows to be studied next are shown. Based on the RFC 6749 standard, which defines the specifications of OAuth 2.0, four standard flows are established, with two well-known extension flows added from the community. [7]

5.3 Types of Flows

After studying different aspects of the flows, we will now analyze the most well-known OAuth 2.0 flows. Although the goal of all flows is to allow the application to obtain the access token to access the resources, each flow represents a different way of achieving this.

Application developers are responsible for selecting the flow that best suits their application, ensuring a secure and efficient token acquisition process. It is always recommended to use the most secure flows for applications that access the most sensitive data of the organization.

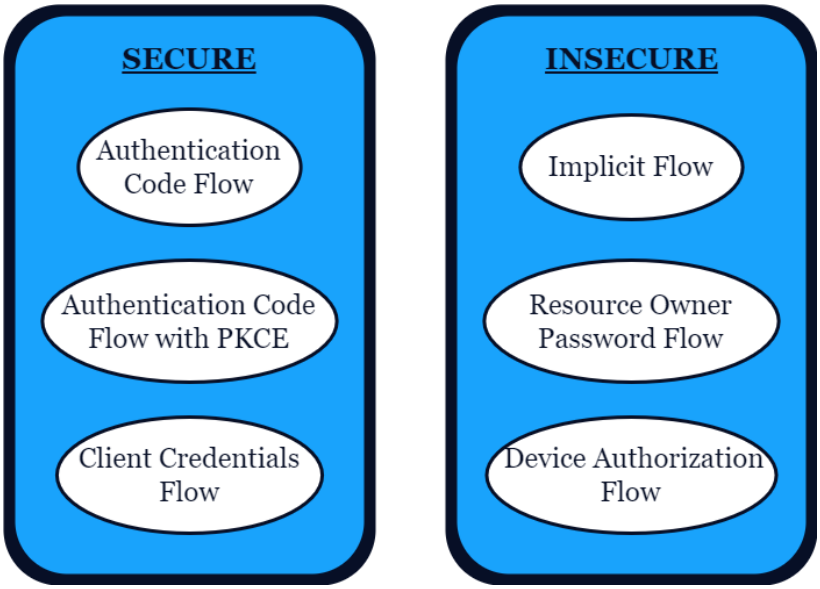

Figure 23 - Flows classified by security (Own elaboration)

5.3.1 Authorization Code Flow

This flow is considered one of the most secure, as it uses a security code called code and a client secret, which the OAuth server requests to issue the access token. These protect against interception attacks. Additionally, the application uses its backend as a secure storage for the token and other sensitive data, thus preventing exposure in the browser. All of this is combined with HTTPS to encrypt data during transmission, making it practically impossible for an attacker to capture the token during the flow.

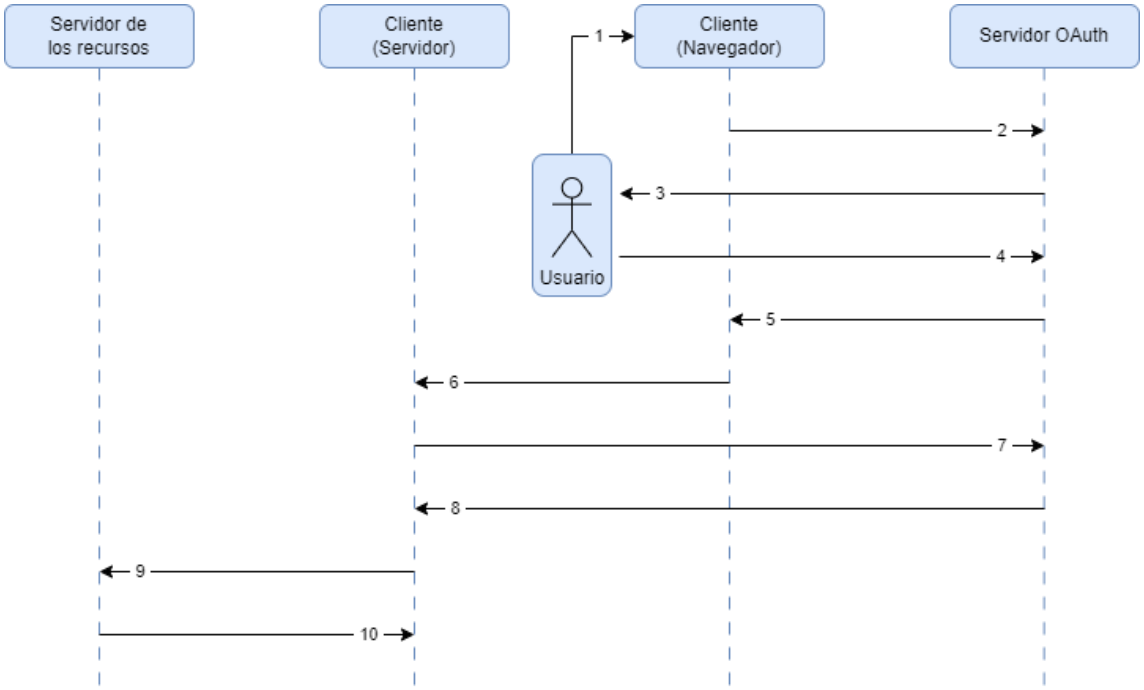

Figure 24 - Authorization Code Flow (Own elaboration)

In Illustration 24, the detailed operation of the flow is shown, and it will be explained step by step:

1. **Application Start:** The user starts the application in their browser.
2. **Authorization Request:** The client requests authorization from the OAuth server with an HTTP request that contains all the parameters previously analyzed.
3. **Redirection to Login:** The user is redirected to the login page of the OAuth server.
4. **User Authentication and Authorization**: The user logs in to the OAuth server and grants consent for the application to access their data.
5. **Issuance of the Code:** The server redirects the user back to the application at the specified redirect_uri, attaching the code. The code is transmitted through the browser but does not pose a security risk since it can only be used once, and the client_secret is also required to obtain the access token.
6. **Sending the Code to the Backend:** The application sends this authorization code to its backend via a secure channel.
7. **Exchange for Access Token:** The client's backend sends a request to the OAuth server to obtain the access token, including the client_secret and the code.
8. **Issuance of the Access Token:** The OAuth server verifies the client_secret and the code. If everything is correct, it issues the access token (and optionally a refresh token).
9. **Access to Resources:** The application uses the access token to access the resources.
10. **Token Validation:** The server verifies the authenticity of the access token, and the application gains access to the protected resource.

5.3.2 Authorization Code Flow with PKCE

This extended flow further enhances the security of one of the most secure flows. By incorporating a dynamic secret, known as PKCE (Proof Key for Code Exchange), this flow protects against code interception attacks, making it an ideal option for applications that cannot securely store a client secret, such as native apps and SPAs (Single-Page Apps). A practical example of its utility is the login process in a mobile app that uses identity providers like Google or Facebook.

This new secret works by having the application generate a code verifier, which is a cryptographically random code, and from this, it generates a code challenge. Initially, the application will share the challenge with the OAuth server and will only share the verifier at the end of the flow when requesting the token. The OAuth server will resolve the challenge by obtaining the verifier and comparing it to the one received. It will only issue the token if both match.

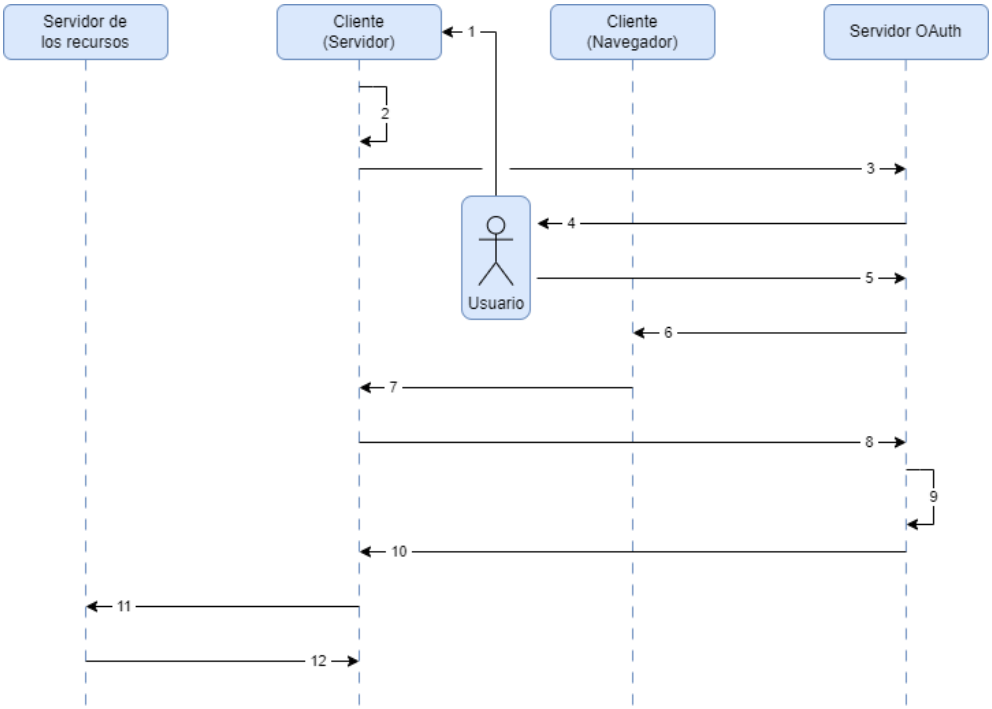

Figure 25 - Authorization Code Flow with PKCE (Own elaboration)

In Illustration 25, the detailed operation of the flow is shown, and it will be explained step by step:

1. **Application Start**: The user starts the application in their browser.
2. **PKCE Creation**: The application creates a code verifier and from it generates the code challenge.
3. **Authorization Request**: The client requests authorization from the OAuth server with an HTTP request that contains all the previously analyzed parameters, now including the challenge.
4. **Redirection to Login**: The user is redirected to the login page of the OAuth server.
5. **User Authentication and Authorization**: The user logs into the OAuth server and grants consent for the application to access their data.
6. **Code Issuance**: The OAuth server stores the challenge and redirects the user back to the application, attaching the code in the URL.
7. **Sending the Code to the Backend**: The application sends this authorization code to its backend via a secure channel.

8. **Exchange for Access Token**: The client's backend sends a request to the OAuth server to obtain the access token, including the code, code verifier, and client secret.
9. **Verification**: The OAuth server resolves the challenge and compares the obtained result with the verifier, also comparing the code and client secret with the stored ones.
10. **Access Token Issuance**: If everything is correct, the access token is issued.
11. **Access to Resources**: The application uses the access token to access the resources.
12. **Token Validation**: The server verifies the authenticity of the access token, and the application gains access to the protected resource.

5.3.3 Client Credentials Flow

This flow allows an application to authenticate and authorize using its own credentials when calling a web resource, without requiring user intervention. This feature makes it especially useful for Machine-to-Machine (M2M) applications, where interactions between servers occur in the background, without direct user involvement. A clear example of using this flow is a backend application that needs permission to access a third-party API. For now, this flow is considered secure, as long as proper security practices are followed, such as securely storing the client secret in the client's backend.

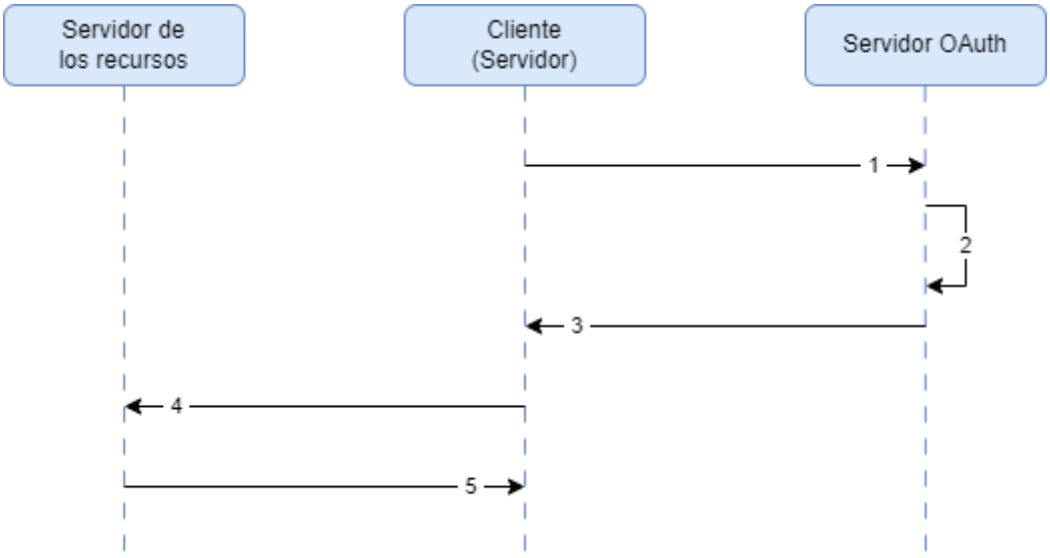

Figure 26 - Client Credentials Flow (Own elaboration)

In Illustration 26, the detailed operation of the flow is shown, and it will be explained step by step:

1. **Authorization Request:** The client sends its credentials to the OAuth server.
2. **Credential Validation:** The OAuth server checks if the client's credentials are correct.
3. **Token Issuance:** If everything is correct, the OAuth server issues the access token and sends it directly to the client's backend.
4. **Access to Resources**: The application uses the access token to access the resources.
5. **Token Validation:** The server verifies the authenticity of the access token, and the application gains access to the protected resource [8]

5.3.4 Implicit Flow

This flow was originally designed for browser-based and native applications, as it allowed for direct access token retrieval without the need for an additional code exchange. Unlike the Authorization Code Flow, where the client had to request the access token by presenting the code and client secret, in this flow, the client can obtain the access token without the need for any code or client secret, simply by authenticating with the credentials..

Due to the lack of security resulting from the absence of a code and client secret, along with the fact that the access token is transferred from the OAuth server to the client through the browser URL, which presents a risk of interception attacks, this design, although overly simple, presents several vulnerabilities. As a result, its use is no longer recommended by organizations such as OAuth and Microsoft when aiming to follow best security practices.

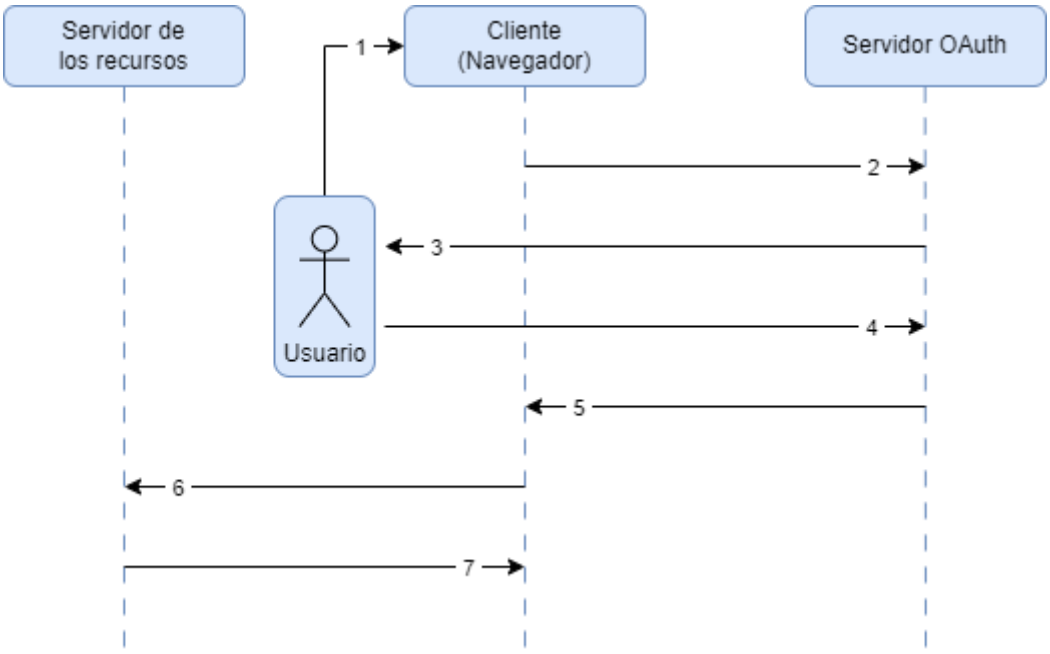

Figure 27 - Implicit Flow (Own elaboration)

In Illustration 27, the detailed operation of the flow is shown, and it will be explained step by step:

1. **Application Start**: The user starts the application in their browser.
2. **Authorization Request**: The client requests authorization from the OAuth server with an HTTP request that contains all the previously analyzed parameters.
3. **Redirect to Login**: The user is redirected to the OAuth server's login page.
4. **User Authentication and Authorization**: The user logs in to the OAuth server and grants consent for the application to access the data.
5. **Access Token Issuance**: The OAuth server issues the access token, which it shares with the client by including it in the URL sent to the browser.
6. **Access to Resources**: The application uses the access token to access the resources.
7. **Token Validation**: The server verifies the authenticity of the access token, and the application accesses the protected resource. [9]

5.3.5 Resource Owner Password Flow

In this flow, the client directly requests the user's credentials, instead of the OAuth server managing the credentials entry. The application can ask the user to enter their credentials through an interactive form and then send them to the backend for future use before exchanging them for an access token with the OAuth server.

For security reasons, this flow is not recommended and is reserved for high-trust applications that ensure the protection of the user's credentials at all times.

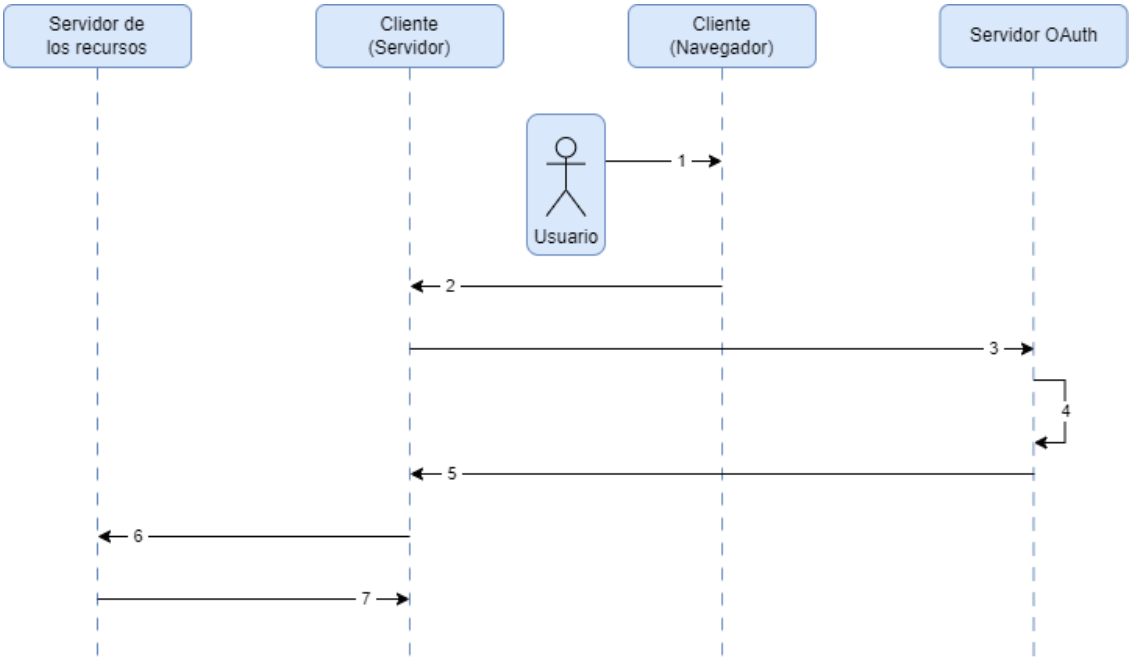

Figure 28 - Resource Owner Password Flow (Own elaboration)

In Illustration 28, the detailed operation of the flow is shown, and it will be explained step by step:

1. **Credentials Entry**: The user starts the application in their browser and enters their credentials in a login form displayed by the browser.
2. **Credential Storage**: The browser sends the credentials to the application's backend via an HTTP request.
3. **Credential Submission**: The backend receives the credentials, validates them, and then sends the credentials to the OAuth server for authentication and authorization.
4. **Validation**: The OAuth server validates the credentials.
5. **Access Token Issuance**: The server generates the token and transmits it to the client for storage in its backend.
6. **Access to Resources**: The application uses the access token to access the resources.
7. **Token Validation**: The server verifies the authenticity of the access token, and the application accesses the protected resource. [10]

5.3.6 Device Authorization Flow

This flow is designed for devices that lack a browser or have input restrictions, such as Smart TVs or IoT devices. It works by the device without a browser displaying a code to the user, who then needs to visit a URL on another device, like a mobile phone or PC, to complete the flow and obtain the access token on the primary device.

However, this method carries security risks, including the potential exposure of the authorization code during the process and vulnerability to intermediary attacks, such as "Man-In-The-Middle" (MITM).

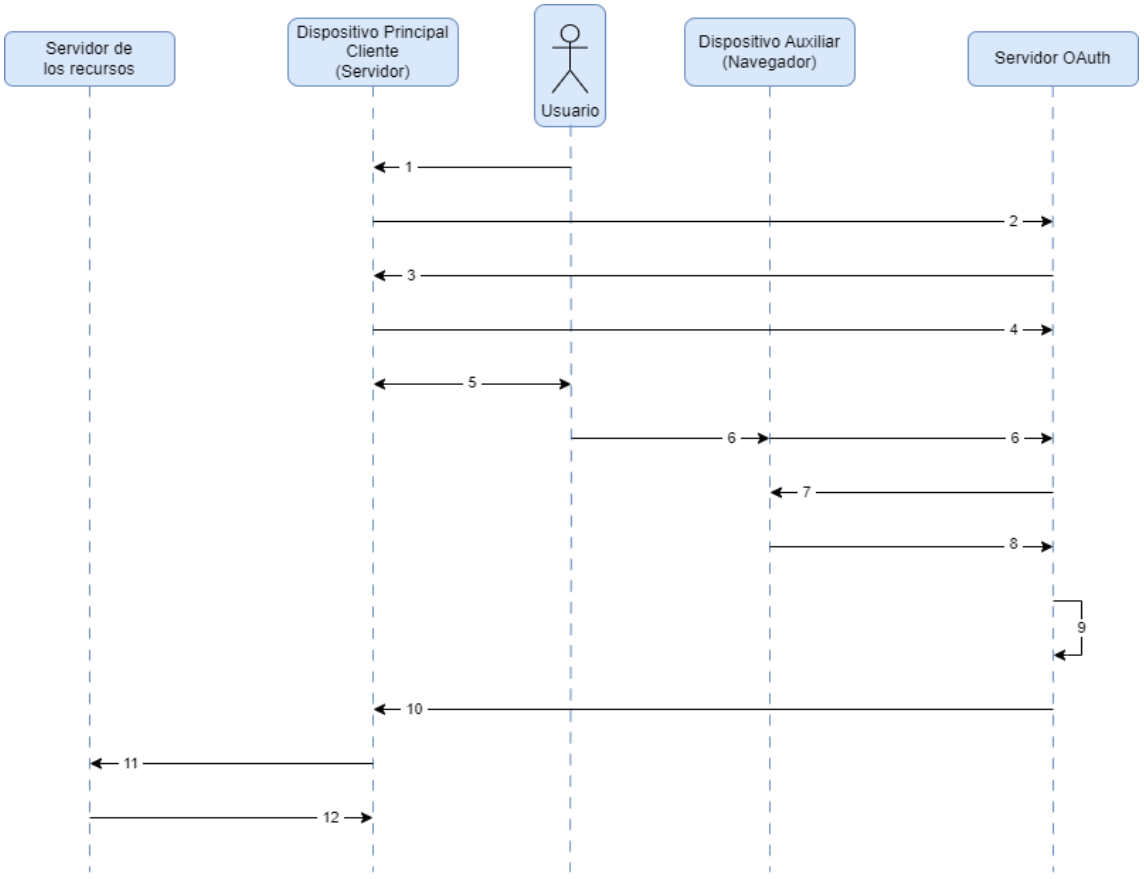

Figure 29 - Device Authorization Flow (Own elaboration)

In Illustration 29, the detailed operation of the flow is shown, and it will be explained step by step:

1. **Application Start**: The user starts the application on the primary device.
2. **Authorization Request**: The client requests authorization from the OAuth server.
3. **Verification URL**: The OAuth server responds with the verification URI and a series of codes such as the device code, user code, their expiration times, and the polling interval for requests.
4. **Polling**: The client starts making periodic requests to the OAuth server to obtain the access token, which will continue until authorization is completed on the secondary device or the codes expire.
5. **Access the URL**: The user accesses the URL by entering it on the secondary device or scanning it if presented as a QR code.
6. **Verification**: The user enters the user code received on the primary device (if accessed via QR, it is done automatically).
7. **Redirect to Login**: The OAuth server redirects the user to the login page, then to the consent page.
8. **Authentication and Consent**: The user logs into the OAuth server and then grants consent for the application to access their data.
9. **Client Authorization**: The function of the secondary device ends once the process is successfully completed.
10. **Access Token Issuance**: The server generates the token and transmits it to the client for storage in its backend.
11. **Access to Resources**: The application uses the access token to access the resources.
12. **Token Validation**: The server verifies the authenticity of the access token, and the application accesses the protected resource. [11]

Chapter 6. Case study

This section covers the integration of a web application with Entra ID to act as its IdP (Identity Provider). This process will detail the necessary actions and configurations from both perspectives: the application's and Entra ID's, ensuring proper and secure functionality.

The application has a simple design and functionality, focusing primarily on demonstrating how to connect it with Entra ID while minimizing risks to the accessed data. The user will log in using their Microsoft credentials and be prompted to grant permissions to the application. Afterward, they are redirected to a welcome page, where they can use Microsoft Graph API, a tool for accessing and manipulating data from Microsoft 365 and other services. By calling the API, another page will display the user's profile data retrieved through a query.

Initially, the application will use the Implicit Flow due to its simplicity. The goal is to analyze the vulnerabilities it presents and how an attacker could capture the access token. After identifying these issues, the process will be secured by switching to the Authorization Code Flow and using Entra ID tools to block unauthorized access

6.1 Integration with Implicit Flow

The integration process begins with the actions needed in Entra ID, followed by designing the application's code. The application’s current functionality will then be demonstrated, concluding with an analysis of its vulnerabilities.

6.1.1 Application creation in Entra ID

The first step is to create the application in Entra ID. Since the application will use OAuth for authentication and authorization, navigate to the App Registration section and select New registration. Enter the application's name and choose the option to "allow access only to accounts in this tenant" for enhanced security.

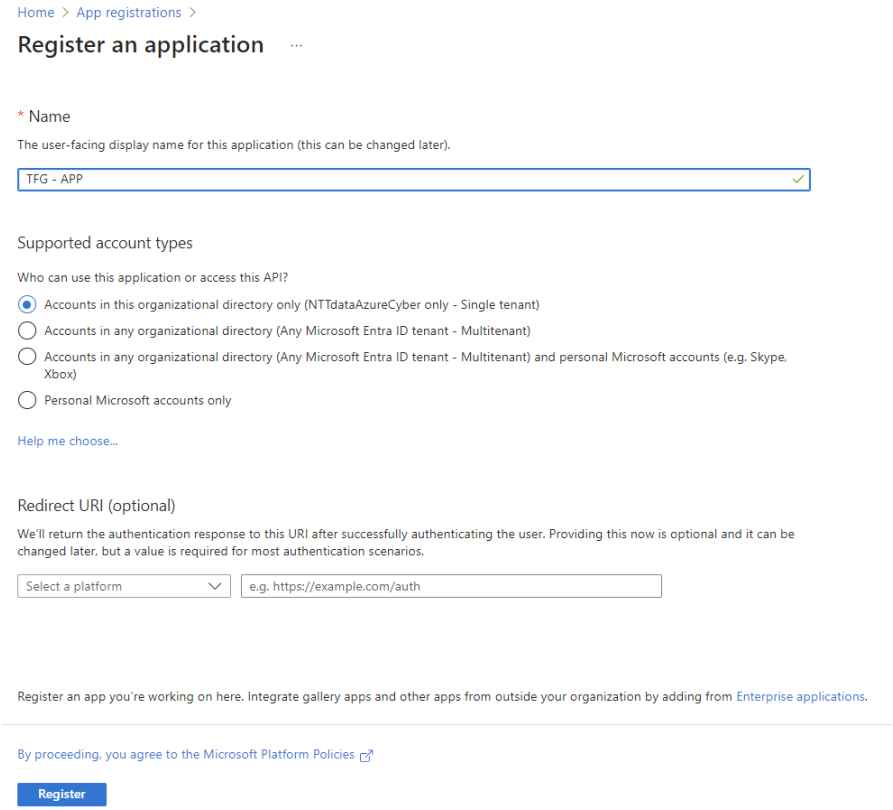

Figure 30 - App Registration creation (Own elaboration)

Once the application is created, certain configurations must be made in the Authentication section. Here, you must define the Reply URL, which is the address where the token will be sent after completing the flow. Additionally, options must be enabled to allow Entra ID to work with the Implicit Flow, as it is disabled by default due to Microsoft no longer recommending its use, as mentioned earlier.

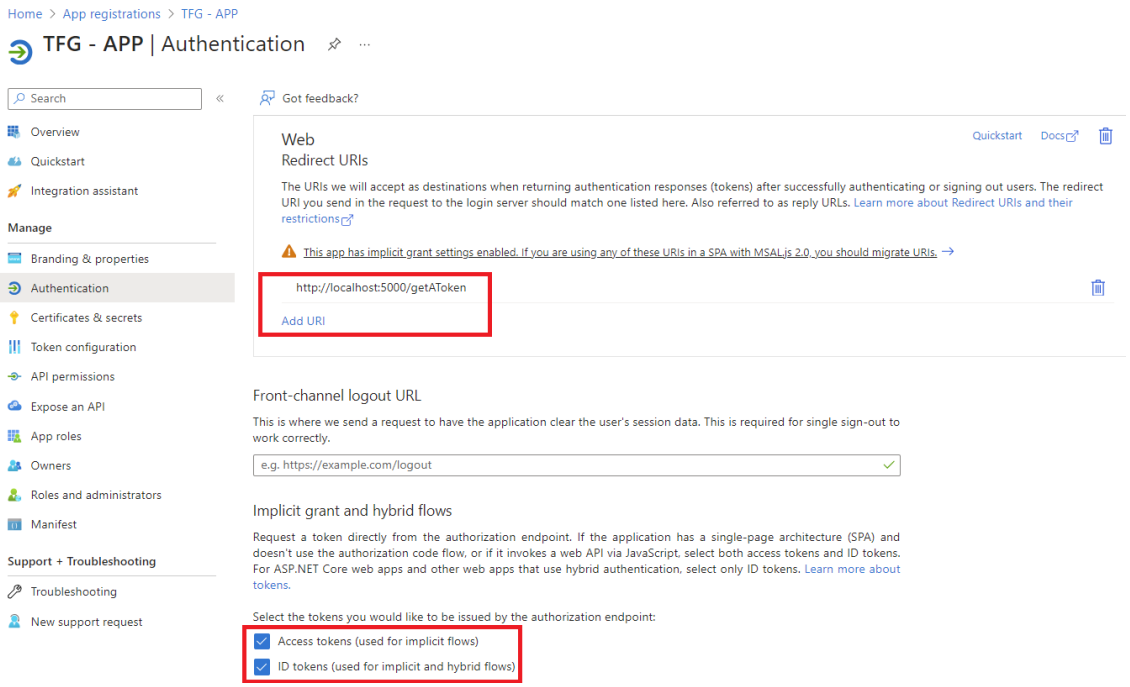

Figure 31 - App Registration configuration for Implicit Flow (Own elaboration)

Finally, permissions must be assigned to the application to specify what it will request from the user after authentication. For this application, only read permissions are needed. Following the principle of Least Privilege Access, the application will be granted only the user.read permission. This principle ensures that only the minimal necessary permissions are provided for the user to perform their tasks, thereby minimizing risks.

To assign permissions, navigate to the API Permissions section and select Add a permission. Choose the required API, which in this case is Microsoft Graph. Then, ensure that the permissions are set as Delegated since the application only needs them when the user is logged in. Search for and select the user.read permission to finalize the configuration.

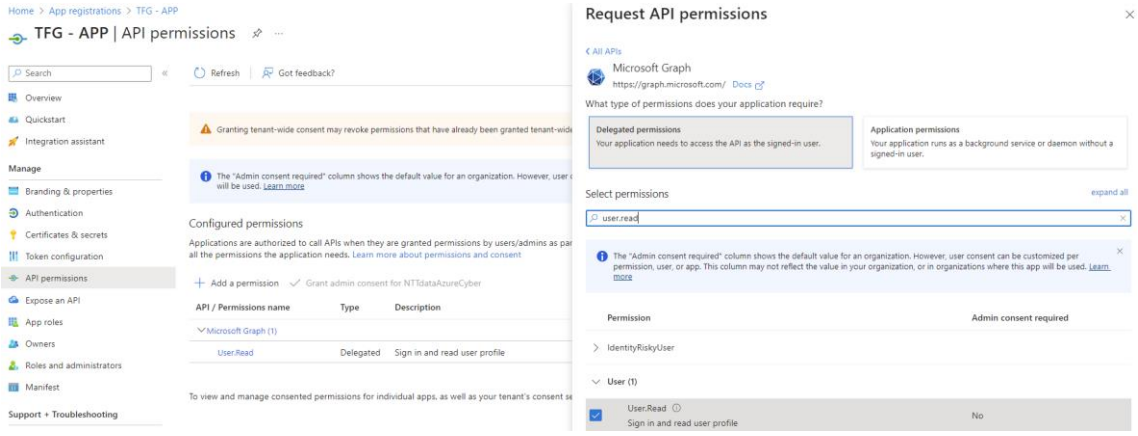

Figure 32 - API Permission configuration (Own elaboration)

6.1.2 Code

The first step is to create a ".env" file where the environment variables will be stored. In this case, the "CLIENT_ID" of the application, which is obtained from Entra ID, and an "AUTHORITY" variable formed by the URL https://login.microsoftonline.com/ followed by the Tenant ID, which is also obtained from Entra ID, both from the properties section of the App Registration.

```
1. CLIENT_ID=3dc9137e-a8cb-4263-b278-382fe7612a4b
2. AUTHORITY=https://login.microsoftonline.com/1e2fadca-913b-4243-91b7-e4f0cf339ab5
```

Next, a file named "app_config.py" must be created to configure the parameters required for the application to connect to Entra ID. First, the variables "AUTHORITY" and "CLIENT_ID" will be defined to extract the values from the environment variables defined in ".env". Additionally, the redirect path, the API Graph endpoint, the required permissions, and the session type to be used will be configured.

```
 1. import os
 2.
 3. AUTHORITY = os.getenv("AUTHORITY")
 4. CLIENT_ID = os.getenv("CLIENT_ID")
 5.
 6. # Ruta de redirección después del login
 7. REDIRECT_PATH = "/getAToken"
 8.
 9. # Endpoint de la API
10. ENDPOINT = 'https://graph.microsoft.com/v1.0/me'
11.
12. # Permisos requeridos
13. SCOPE = ["User.Read"]
14.
15. # Establece almacenar las sesiones en el sistema de archivos
16. SESSION_TYPE = "filesystem"
```

With these two files, critical parameters are separated from the application code, adding a security layer since an attacker gaining access to the code will not see these values.

Once the configuration files are created, the main code is implemented in a file called “app.py”, where the following libraries are imported:

- **requests**: Used to make HTTP requests.
- **flask**: Imports basic Flask functionalities for creating the web application.
- **flask_session**: Used to manage sessions in the Flask application.

Additionally, the configurations defined in app_config are imported, and the application version is set as 1.0.0.

```
import requests

from flask import Flask, redirect, render_template, request, session, url_for
from flask_session import Session

import app_config
__version__ = "1.0.0"
```

Next, a Flask instance is created, and the application configuration is loaded from the “app_config” object. A check is performed to ensure that the “REDIRECT_PATH” variable is not set as the root ("/"), as this could cause security issues. Finally, session management is initialized, allowing the application to maintain the user's session state throughout their interaction.

```
app = Flask(__name__, static_url_path='', static_folder='static')
app.config.from_object(app_config)
assert app.config["REDIRECT_PATH"] != "/", "REDIRECT_PATH must not be /"
Session(app)
```

Since the app runs on localhost, ProxyFix middleware is used to correct request headers.

```
from werkzeug.middleware.proxy_fix import ProxyFix
app.wsgi_app = ProxyFix(app.wsgi_app, x_proto=1, x_host=1)
```

The routes to handle requests must now be defined, starting with /login. In the login function, the URL is constructed to redirect the user to Microsoft's login page using the application's configuration parameters, including the client_id, the response type (set to “token” when working with Implicit Flow), the redirect URL, the required permissions from the scope, and a prompt to select the account. Finally, the function redirects the user to this authorization URL.

```
@app.route("/login")  # Define la ruta para manejar las solicitudes a /login
def login():
    auth_url = (
        f"{app.config['AUTHORITY']}/oauth2/v2.0/authorize"
        f"?client_id={app.config['CLIENT_ID']}"
        f"&response_type=token"
        f"&redirect_uri={url_for('auth_response', _external=True)}"
        f"&scope={' '.join(app_config.SCOPE)}"
        f"&prompt=select_account"
    )
    return redirect(auth_url)
```

The route to handle the authentication response must also be defined. To do this, it checks if the access token has been received through a POST request. If the token is received, it is stored in the session, and the user is redirected to the main page (index). If not found, the user is redirected back to the login page. If the request is a GET, an auth_response.html template is rendered.

```
@app.route(app_config.REDIRECT_PATH, methods=["GET", "POST"])
def auth_response():
```

```
    if request.method == "POST":
        access_token = request.form.get("access_token")
        if access_token:
            session["access_token"] = access_token
            return redirect(url_for("index"))
        else:
            return redirect(url_for("login"))
    return render_template("auth_response.html")
```

When a request reaches the main route, the application must first check if “CLIENT_ID” is set. If it is not, an error page is loaded. Then, it verifies whether the user is logged in by checking if an access token exists in the session. If no token is found, the user is redirected to the login page. If all checks are successful, the main page is rendered, displaying the logged-in user's name.

```
@app.route("/")
def index():
    if not app.config["CLIENT_ID"]:
        return render_template('config_error.html')
    if "access_token" not in session:
        return redirect(url_for("login"))
    return render_template('index.html', user={"name": "User"}, version=__version__)
```

For requests to the logout route, the access token must be removed from the session to ensure the user is no longer authenticated. This is achieved using “session.pop”. Afterward, the user is redirected to the main page, which, as explained, will send the user back to the login page if no token is detected in the session.

```
@app.route("/logout")  # Define la ruta para manejar las solicitudes a /logout
def logout():
    session.pop("access_token", None)
    return redirect(url_for("index"))
```

The last route to define is the one handling calls to the Graph API. First, it checks if the access token is present in the session. If not, the user is redirected to the login page. If the token is present, a GET request is made to the API, passing the token in the authorization header. The API response is received in JSON format and passed to the display.html template to display it on-screen.

```
# Define la ruta que maneja solicitudes a /call_downstream_api
@app.route("/call_downstream_api")
def call_downstream_api():
    if "access_token" not in session:
        return redirect(url_for("login"))
    api_result = requests.get(
        app_config.ENDPOINT,
        headers={'Authorization': 'Bearer ' + session["access_token"]},
        timeout=30,
    ).json()
    return render_template('display.html', result=api_result)
```

Finally, the application is executed using app.run(), which starts Flask’s development web server, enabling the application to accept connections.

```
if __name__ == "__main__":
    app.run()
```

The application's various pages are designed using HTML, with a CSS file to enhance visual presentation. One particular HTML page is of great importance: it loads after authentication and authorization are completed and executes a script that extracts the token from the URL.

The first action of the script is to use several functions to locate the access token in the URL and store it in the variable token.

```
<script type="text/javascript">
function getTokenFromUrl() {
    const hash = window.location.hash.substring(1);
    const params = new URLSearchParams(hash);
    const token = params.get("access_token");
```

Next, it checks whether the token was successfully found. If so, it dynamically creates an HTML form to send the token to the backend through a POST request.

```
if (token) {
    const form = document.createElement("form");
    form.method = "POST";
    form.action = "{{ url_for('auth_response') }}";

    const hiddenField = document.createElement("input");
    hiddenField.type = "hidden";
    hiddenField.name = "access_token";
    hiddenField.value = token;

    form.appendChild(hiddenField);
    document.body.appendChild(form);
    form.submit();
```

If no token is found, the script redirects the user to the login page to restart the authentication process.

```
} else {
    window.location.href = "{{ url_for('login') }}";
}
```

6.1.3 Functionality

With everything completed, the application is now functional, and the next step is to verify that it works correctly. To do this, the application will be run locally on port 5000 by executing the following command: flask run --host=localhost --port=5000

The user accesses the application through the URL http://localhost:5000 and is immediately redirected to the Microsoft login page, where they must sign in using an account from the tenant where the application has been registered.

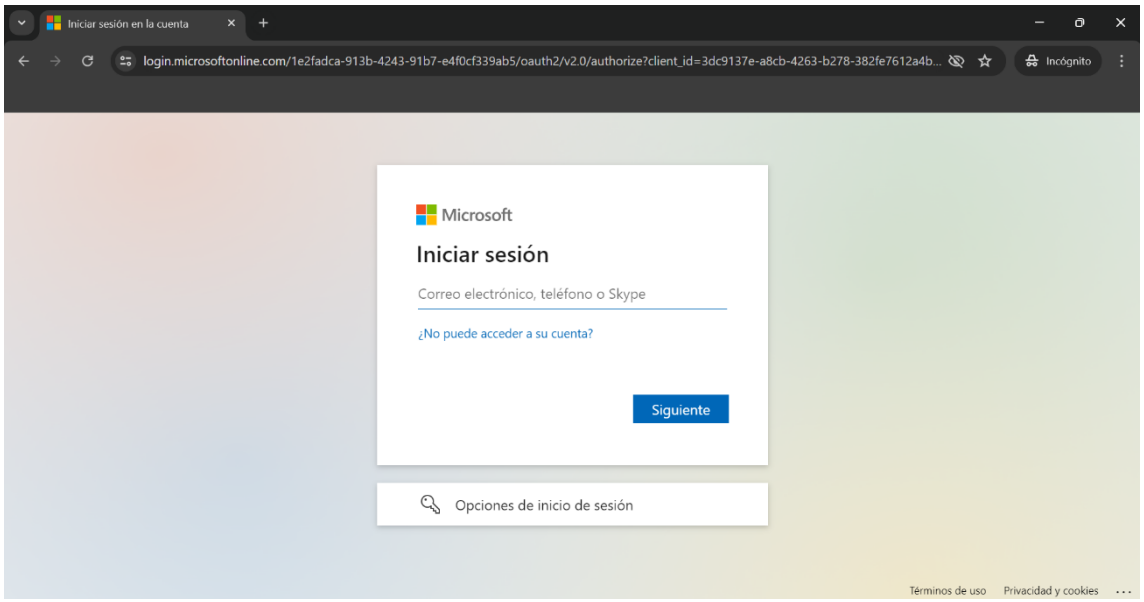

Figure 33 - Microsoft login page (Own elaboration)

Once the user successfully accesses the application, a pop-up will appear, informing them of the permissions required by the application and asking for authorization.

At this point, the flow is complete, and Entra ID has issued the access token, which is now in possession of the client. The user is redirected to the main page, where two options are presented: log out or call the Graph API.

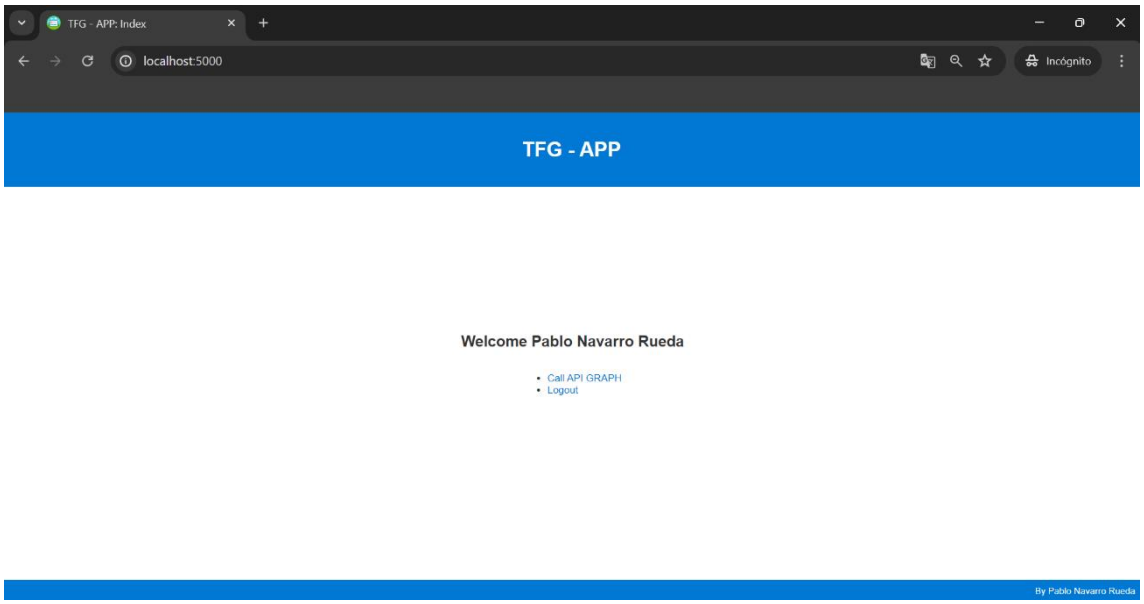

Figure 34 - Main page (Own elaboration)

If the logout option is selected, the user will return to the Microsoft login page and will have to sign in again. If the option to call Graph is selected, the application uses the access token it has just stored to access the user's data and redirects the user to the Display HTML page, where the user data is shown, obtained through a query to Graph.

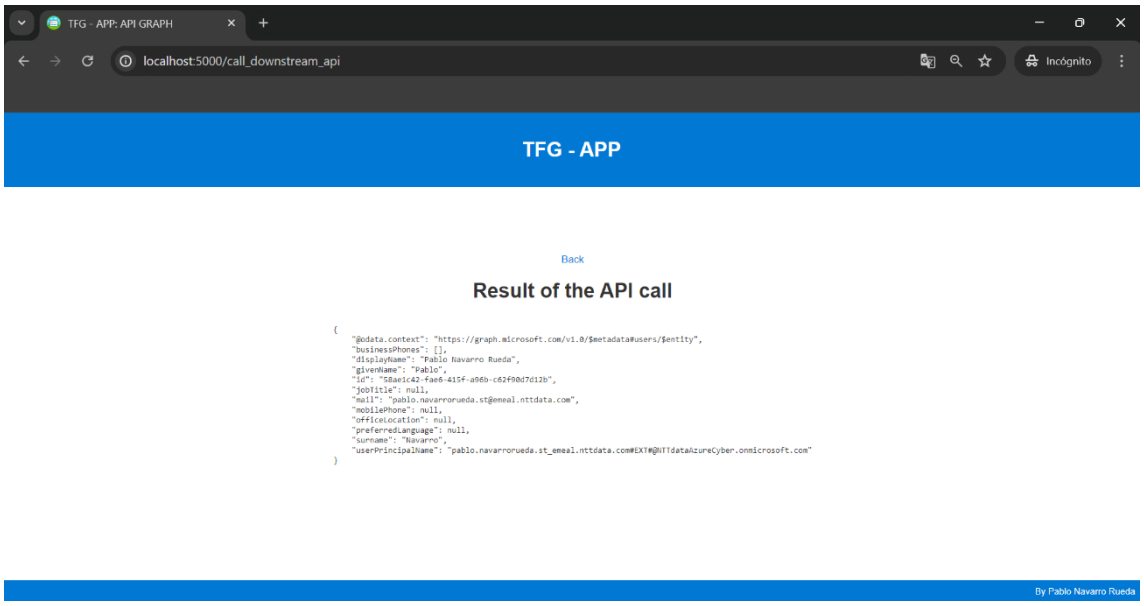

Figure 35 - Call to Graph (Own elaboration)

6.1.4 Implementation Issues

Although the functionality of the application has been verified, the current configuration presents several security risks, primarily due to the use of Implicit Flow. One of the main risks of this flow is the inclusion of the access token in the redirect URL, allowing the client to extract it.

Next, we will simulate an environment in which an attacker manages to bypass HTTPS encryption by connecting to the same network as the user of the application. In such a situation, the attacker could carry out a Man-in-the-Middle attack by intercepting the traffic and obtaining the access token.

To demonstrate how an attacker could achieve this, we use Wireshark, a tool for analyzing network traffic. Since the application's backend runs on localhost, Wireshark must be configured to listen on the loopback channel, thus placing itself between the client's browser and the backend server.

First, capture is started, and the application is accessed as done previously. When doing so, various traffic packets are captured in Wireshark. By applying the "http" filter to display only the relevant packets, one particularly significant packet can be identified.

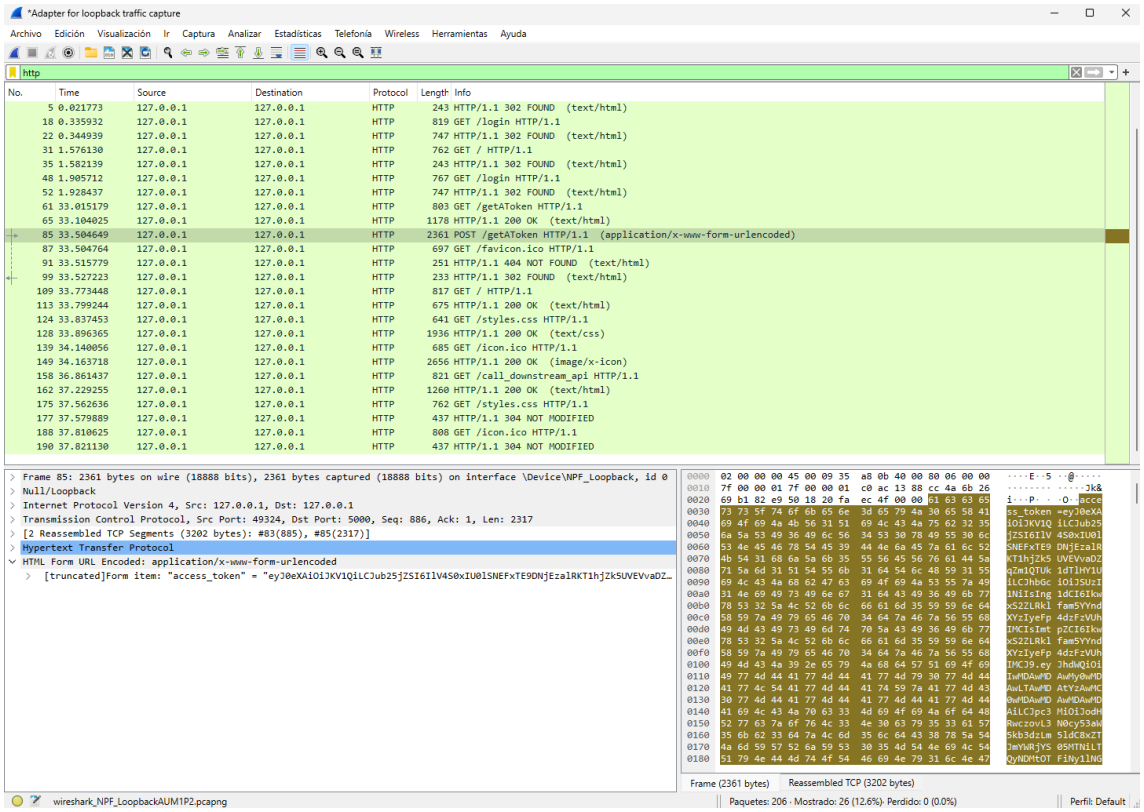

Figure 36 - Wireshark capture of traffic with Implicit Flow (Own elaboration)

The request shown in detail in the previous image is a POST request made by the client's browser to the backend. This is evident since the source is port 49324 (browser) and the destination is port 5000 (backend), previously assigned with the execution command. The purpose of this request is that once the access token is extracted from the redirect URL, the browser must send this token to the server for secure storage and future use without needing to obtain a new one. At the bottom right of the illustration, it is clearly visible how the content of the request includes the full access token "access_token=...".

6.1.5 Token analysis

The attacker obtaining the access token is critical due to the sensitive information it contains. To understand the data that can be extracted from the captured token, an analysis has been conducted using an Okta web application that analyzes JSON Web Tokens (JWT).

```
HEADER:
{
  "typ": "JWT",
  "nonce": "UxKLHSIR4AqLOC613jTJOXcfNTTEoh6jfmPMI5u9GcU",
```

```
  "alg": "HS512",
  "x5t": "L1KfKFI_jnXbwWc22xZxw1sUHH0",
  "kid": "L1KfKFI_jnXbwWc22xZxw1sUHH0"
}
```


The header of the token contains information about the token type and the algorithm used for signing. As observed, the access token is of type JWT and uses the HS512 algorithm for signing, which utilizes the SHA-512 hash function. Additionally, the header includes the nonce, a unique value to prevent replay attacks; x5t, which is the public key of the x.509 certificate used to sign the token; and kid, the key identifier, used to determine the corresponding signature when multiple signatures are available.


```
PAYLOAD:
{
  "aud": "00000003-0000-0000-c000-000000000000",
  "iss": "https://sts.windows.net/1e2fadca-913b-4243-91b7-e4f0cf339ab5/",
  "iat": 1718643360,
  "nbf": 1718643360,
  "exp": 1718649050,
  "acct": 1,
  "acr": "1",
  "acrs": [
    "urn:user:registersecurityinfo"
  ],
  "aio":
"AWQAm/8XAAAAQ4yc5Z2R7GpUj2NV9uvSf8StDeAdBsjAKEijXVBb16WVh1nXoQ8IqTCmEZPEM8sBBab2DsHPFI6Vs
VFO4V1Q/AxPL/LuTULbySyV3ayDjraHD/HnM7Vi3Br1XybBeyCp",
  "altsecid": "5::1003200357669F25",
  "amr": [
    "pwd"
  ],
  "app_displayname": "TFG - APP",
  "appid": "3dc9137e-a8cb-4263-b278-382fe7612a4b",
  "appidacr": "0",
  "email": "pablo.navarrorueda.st@emeal.nttdata.com",
  "family_name": "Navarro",
  "given_name": "Pablo",
  "haswids": "true",
  "idp": "https://sts.windows.net/3048dc87-43f0-4100-9acb-ae1971c79395/",
  "idtyp": "user",
  "ipaddr": "84.120.67.242",
  "name": "Pablo Navarro Rueda",
  "oid": "58ae1c42-fae6-415f-a96b-c62f90d7d12b",
  "platf": "3",
  "puid": "100320035F3C3317",
  "rh": "0.AU4Ayq0vHjuRQ0KRt-TwzzOatQMAAAAAAAAAwAAAAAAAAAOAV0.",
  "scp": "User.Read profile openid email",
  "signin_state": [
    "kmsi"
  ],
  "sub": "sOzmD2O5O6SqeJdtFiLnMkrA2NeOOOfVkt6tiHSxOQY",
  "tenant_region_scope": "EU",
  "tid": "1e2fadca-913b-4243-91b7-e4f0cf339ab5",
  "unique_name": "pablo.navarrorueda.st@emeal.nttdata.com",
  "uti": "iskJYtMBIESuknTlXriAAA",
  "ver": "1.0",
  "xms_idrel": "5 16",
  "xms_st": {
    "sub": "ytJBjmYR2YhgXjPPfF2o_v223KWL7fd8FRlyyF8NNmQ"
  },
  "xms_tcdt": 1634716040,
  "xms_tdbr": "EU"
}
```


Regarding the payload information, it can be classified into three main categories that could be exploited by an attacker:

- **Personal Data**: This includes all information that directly identifies the user, such as the email field, which displays the user's email; the fields name, given_name, and family_name, which show the user's full name; the unique_name field, which displays the User Principal Name (UPN) in Entra ID; and the ipaddr field, which contains the user's IP address at the time the token was issued.
- **Session Data**: This includes information about the authentication and session status, which could be used for replay or session hijacking attacks. The iat field indicates the token's issue timestamp; nbf and exp define the token's start and expiry validity times; acr shows the level of authentication strength; amr indicates the authentication method used (in this case, a password); aio and rh contain session-specific information; and sub is the unique identifier of the user within the context of the token.
- **Application Context**: This contains identifiers and permissions that indicate how and where the token is used, providing information on the environment and potential vulnerabilities to exploit. The iss field shows the authority issuing the token; idp displays the user's identity provider; appid and app_displayname contain the application ID and the name of the application in the tenant; scp is the scope, specifying the permissions granted by the token; and tenant_region_scope and tid indicate the tenant region (Europe in this case) and its identifier.

With all this information in the hands of an attacker, various attacks such as identity spoofing, phishing, replay attacks, or session hijacking can be carried out. These attacks could lead to unauthorized access to sensitive resources or data theft. To prevent this, a series of protective measures will be implemented to secure the token and access controls.

6.2 Transition to Authorization Code Flow

After demonstrating the security risks of using Implicit Flow, changes will be made to shift the application to Authorization Code Flow. As previously mentioned, a client secret will be added, and Entra ID will no longer send the access token directly. Instead, it will issue a code, which the client must store. To request the token, the client will need to send a request containing both the client secret and the code, ensuring that the attacker would need to obtain both values to exploit the system.

6.2.1 Changes in Entra ID

To configure the new flow, several adjustments must be made on the Entra ID side, starting with the creation of a client secret. This can be done in the App Registrations section by searching for the application and accessing its settings. Once inside, navigate to the Certificates & Secrets section and select New client secret under the Client secrets tab. You will need to optionally provide a description for the secret and set its duration. For security reasons, it is recommended to keep the duration as short as possible and renew it regularly, with 6 months being a reasonable duration.

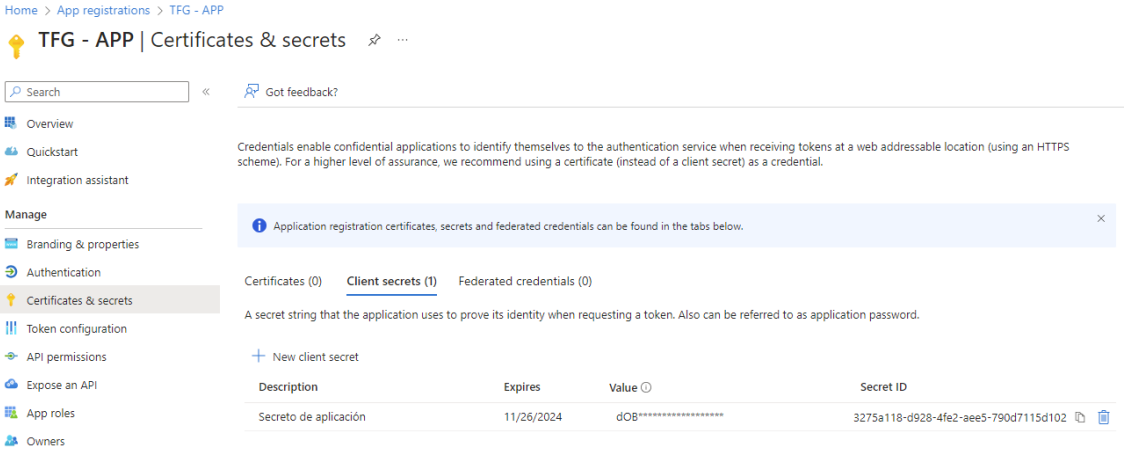

Figure 37 - Creating a client secret in Entra ID (Own elaboration)

It is crucial to save the secret value immediately after creation, as this value cannot be viewed again, even by an administrator. This secret value should be added to the .env file, as it will be required for certain requests during the flow.

Finally, navigate to the Authentication section and disable the two options previously enabled for Implicit Flow, as these must not be enabled for the application to use Authorization Code Flow.

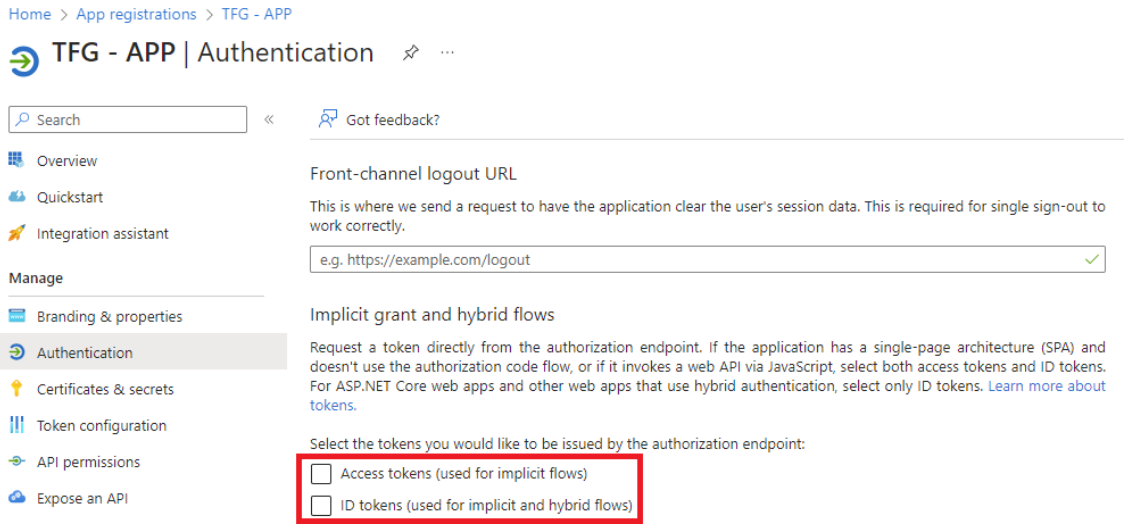

Figure 38 - Configuration for Implicit Flow (Own elaboration)

6.2.2 Changes in the Code

On the code side, the client secret value needs to be added to the .env file so that the application can access it.

```
1. CLIENT_ID=3dc9137e-a8cb-4263-b278-382fe7612a4b
2. AUTHORITY=https://login.microsoftonline.com/1e2fadca-913b-4243-91b7-e4f0cf339ab5
3. CLIENT_SECRET=dOB8Q~44-E2bO5jYRHD5d60sQIdkZjgIN~Zyhcw.
```

A CLIENT_SECRET variable also needs to be created in the app_config.py file, which will get its value by extracting it from the .env file

```
1. CLIENT_SECRET = os.getenv("CLIENT_SECRET")
```

In the main code of app.py, several important changes must be made. First, the identity.web library needs to be imported, which will allow the application to handle authentication and authorization securely.

```
1. import identity.web
```

With this library, the auth object is created to manage the user session; its role is to centralize and simplify authentication, securely handling the token.

```
1. app.jinja_env.globals.update(Auth=identity.web.Auth)  #Agrega variable global
2. #Inicialización del Objeto de Autenticación
3. auth = identity.web.Auth(
4.     session=session,
5.     authority=app.config["AUTHORITY"],
6.     client_id=app.config["CLIENT_ID"],
7.     client_credential=app.config["CLIENT_SECRET"],
8. )
```

The login function changes completely. Now, the authorization URL is constructed through the auth.log_in() method, which ensures the correct parameters required and recommended by Microsoft are followed when implementing Authorization Code Flow.

```
1. def login():
2.     return render_template("login.html", version=__version__, **auth.log_in(
3.         scopes=app_config.SCOPE,
4.         redirect_uri=url_for("auth_response", _external=True),
5.         prompt="select_account",
6.     ))
```

The handling of the authentication response also changes. It is now managed through the auth.complete_log_in(request.args) method, which handles receiving the code, storing it on the server, requesting the access token using the code and the client secret, and securely receiving the token directly on the backend.

```
1. @app.route(app_config.REDIRECT_PATH)
2. def auth_response():
3.     result = auth.complete_log_in(request.args)
4.     if "error" in result:
5.         return render_template("auth_error.html", result=result)
6.     return redirect(url_for("index"))
```

The check on the main page to see if the user is correctly logged in now also checks for CLIENT_SECRET in addition to CLIENT_ID. Moreover, the auth.get_user() method is used to check if the user is logged in correctly, redirecting to the login page in case of an error.

```
1. @app.route("/")
2. def index():
3.     if not (app.config["CLIENT_ID"] and app.config["CLIENT_SECRET"]):
4.         return render_template('config_error.html')
5.     if not auth.get_user():
6.         return redirect(url_for("login"))
7.     return render_template('index.html', user=auth.get_user(), version=__version__)
```

Finally, the logout process also changes. Instead of removing the access token from the session and redirecting the user, the auth.log_out() method is now used, which handles the entire process more securely.

```
1. @app.route("/logout")
2. def logout():
3.     return redirect(auth.log_out(url_for("index", _external=True)))
```

With the shift to Authorization Code Flow, the auth_response HTML and the script to extract the token from the URL are no longer necessary. Now, the token reception is directly managed in the auth_response method on the backend, as previously discussed.

6.2.3 Functionality

The change in the flow is implemented transparently for the user, so there is no change in the application's functionality. The only addition is a login page, from which the user is redirected to the Microsoft login page by clicking the button. This is a purely visual change to provide a welcome page for the app, as it was previously impossible due to the need for an immediate redirection to the Microsoft login portal as part of the flow.

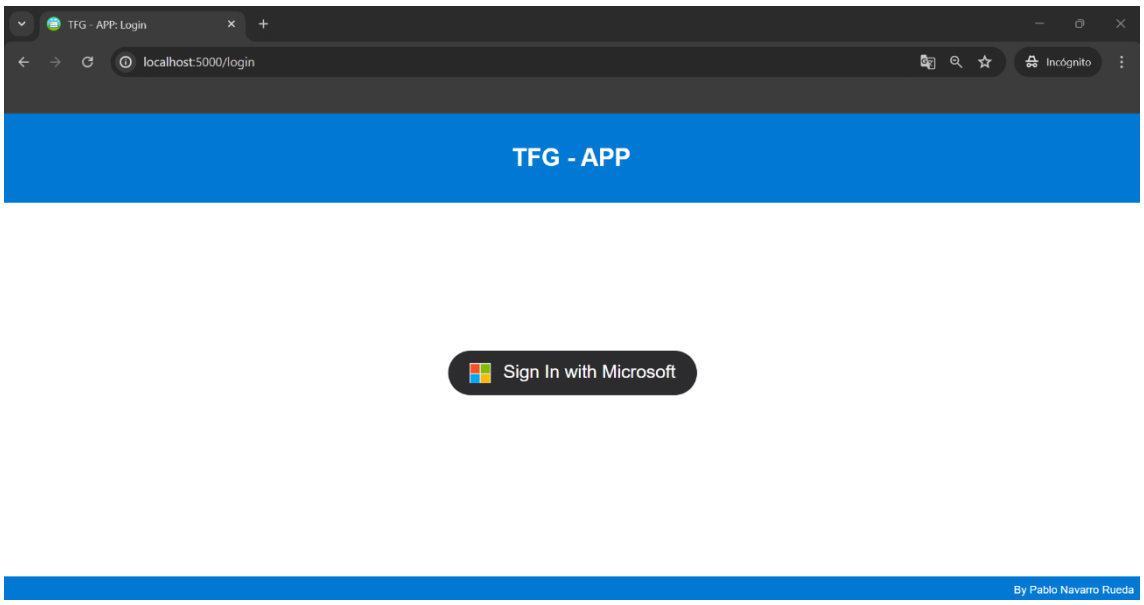

Figure 39 - New Home Page (Own elaboration)

6.2.4 Security analysis

As done for Implicit Flow, an attempt will be made to capture the access token using Wireshark to intercept it. The configuration of Wireshark and the steps to follow are exactly the same. After starting the capture and accessing the application, the intercepted packets appear again, but this time the token does not appear in any of the captured requests.

This is because, unlike Implicit Flow, where the token was sent to the client's browser and allowed the attacker to intercept it when it was passed from the browser to the server for storage, with Authorization Code Flow, the token travels directly from Entra ID to the client's backend. This proves that the token is no longer exposed in the browser, and merely positioning oneself between the browser and the server is no longer enough to capture it.

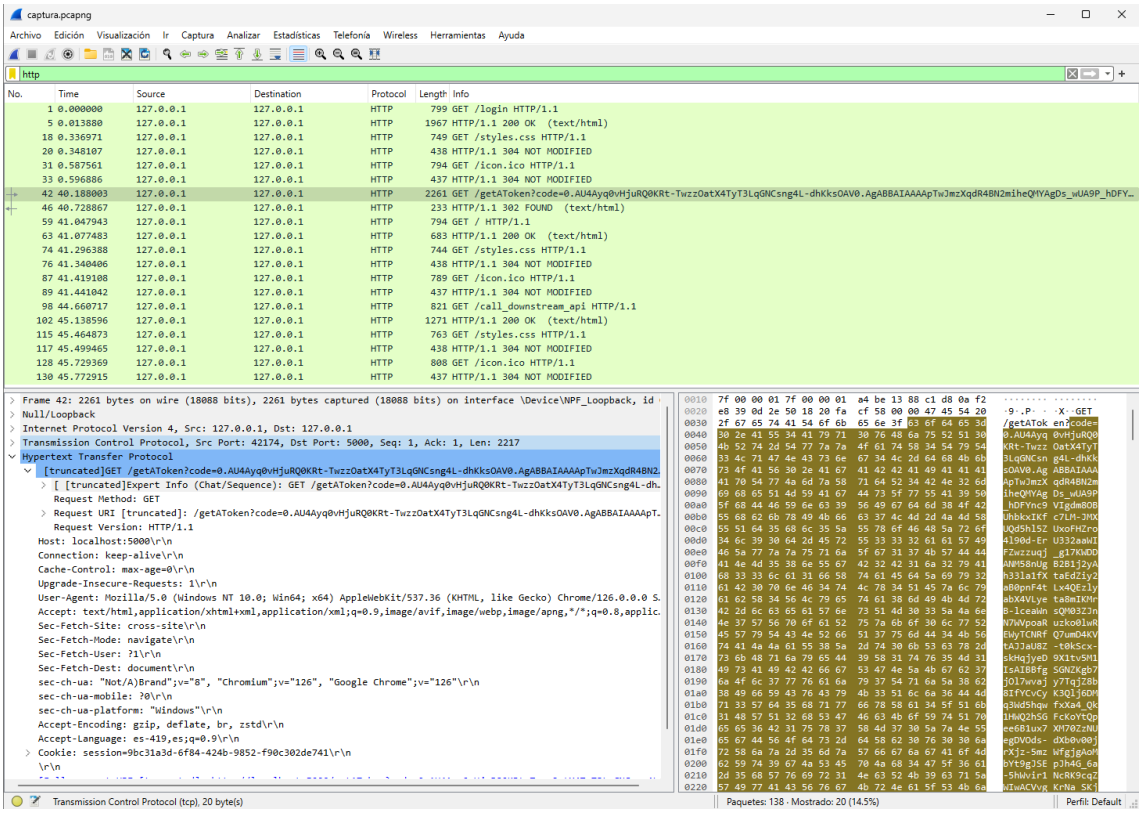

Figure 40 - Wireshark capture with Authorization Code Flow (Own elaboration)

Still, something interesting is captured for the attacker: the code at the moment the client sends it to the backend for storage, just as the access token was previously captured with Implicit Flow. However, this is no longer as sensitive because, to obtain the access token, the attacker must also have the client secret. Having one without the other is useless, and furthermore, the code is single-use, so it does not present a significant risk. In this way, it is demonstrated that with the change made, the vulnerability to Man-in-the-Middle attacks has been greatly reduced.

6.3 Protecting identity and access to the application

Although security has been significantly improved, various tools and features provided by Entra ID can be leveraged to further enhance it. Below, access to the app will be restricted, conditional access policies will be implemented, and MFA will be required to access the application.

6.3.1 Block access except for assigned users

A good starting point to protect the app from unwanted access is to restrict it to specific users. From the "Enterprise apps" menu, the app must be located, accessed, and the "Properties" section must be navigated to. There, the "Assignment required" option should be set to "yes," so that only assigned users can access the application.

A good practice to better control who can access is to create a group that includes all authorized users, which simplifies management. This group is called "Access TFG-APP," and users needing access can be added from the "Groups" menu. The group can then be assigned to the app by going to "Enterprise apps," accessing the application, and selecting "Add user/group" in the "Users and groups" section to search for and add the created group.

Now, when a user who is not part of the group attempts to access the app, their access will be blocked with the following message.

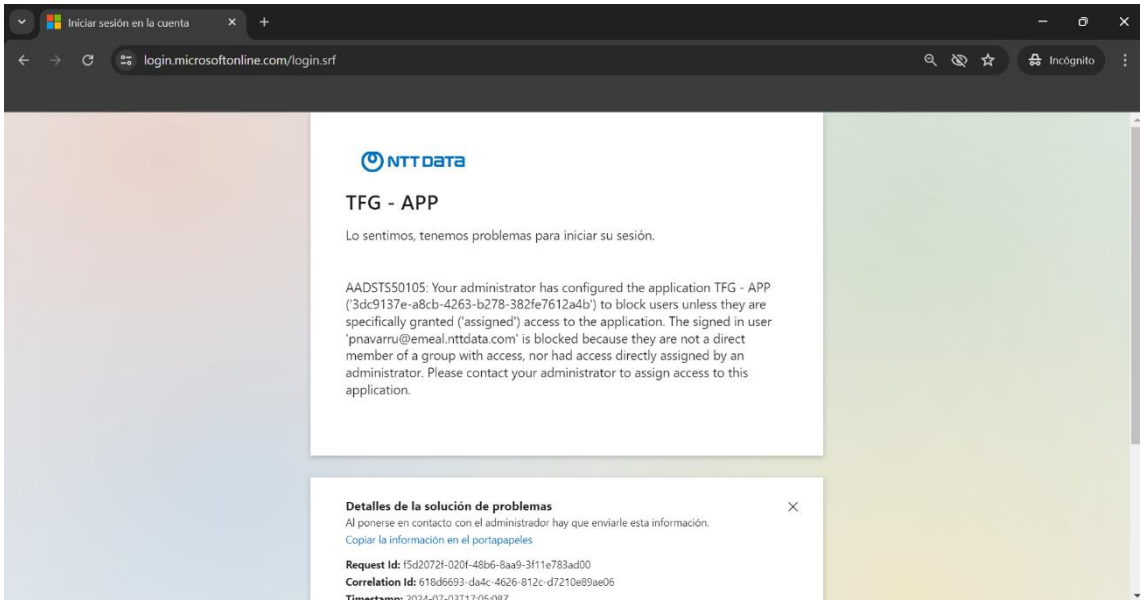

Figure 41 - Blocked access for unassigned user (Own elaboration)

6.3.2 Define conditional access policy for blocking

To achieve a higher level of security, a Conditional Access policy will be defined to deny access unless certain requirements are met. The goal is that if an attacker gains a user's credentials, it will not be sufficient to access the app, as they will also need to meet the conditions.

First, it is necessary to identify the specific cases to block. For this application, access will only be allowed if the user meets one of the following conditions: access from a trusted IP or via a trusted device.

To configure this, a "named location" must first be created, which includes all trusted IPs from which users will be allowed to access, such as the organization’s IPs that require access to the application. For trusted devices, the company must provide devices to its employees, and these must be marked as trusted (compliant). Access will only be allowed from these corporate devices to prevent access from a personal device that could be compromised.

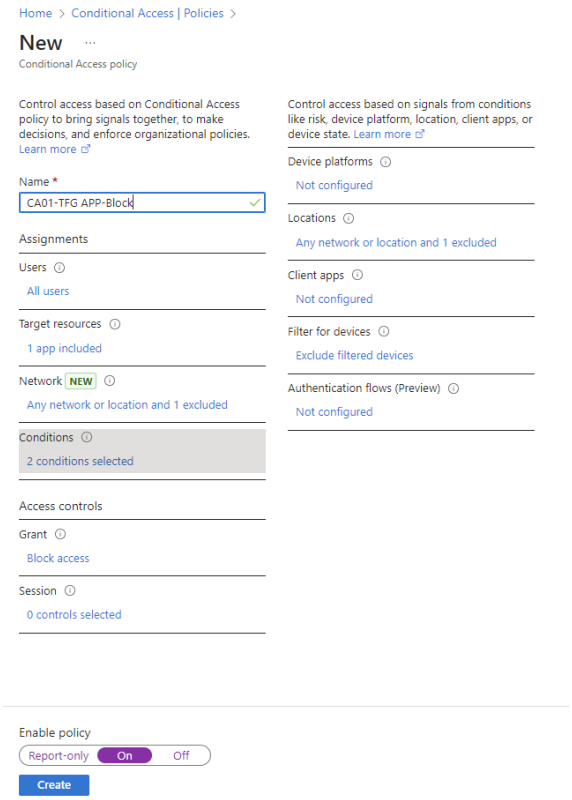

Figure 42 - Creating the blocking policy (Own elaboration)

In Illustration 42, the policy configuration is shown, where in the "Grant" section it is set to block access to all users attempting to access the application (the app is selected in "Target resources"). However, to prevent blocking when the established conditions are met, the created "named location" is excluded, so if access comes from one of those IPs, the policy will not apply. Additionally, compliant devices are excluded, ensuring that only legitimate access is permitted.

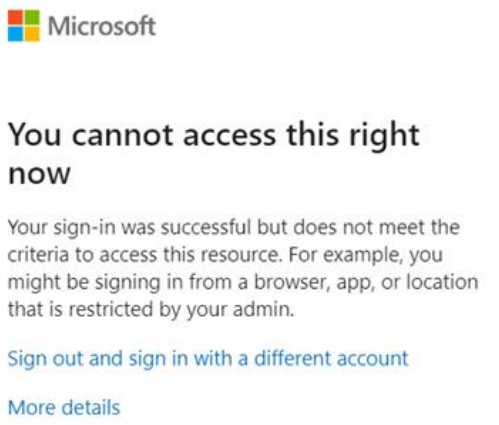

Figure 43 - Blocking message by Conditional Access policy (Own elaboration)

Now, when a user attempts to access and is blocked by this policy, they will receive the error shown in Illustration 43 after completing the login. The message explains that although the credentials they entered are correct, they do not meet the conditions required to access the resource.

6.3.3 Define conditional access policy to require MFA

After establishing a general blocking policy, a good practice is to protect all access with MFA. To achieve this, a policy will be created to request MFA from all users who want to access. This way, even if an attacker manages to obtain a user's credentials and accesses via a compliant device or from trusted IPs, they will not be able to access unless they complete the required MFA.

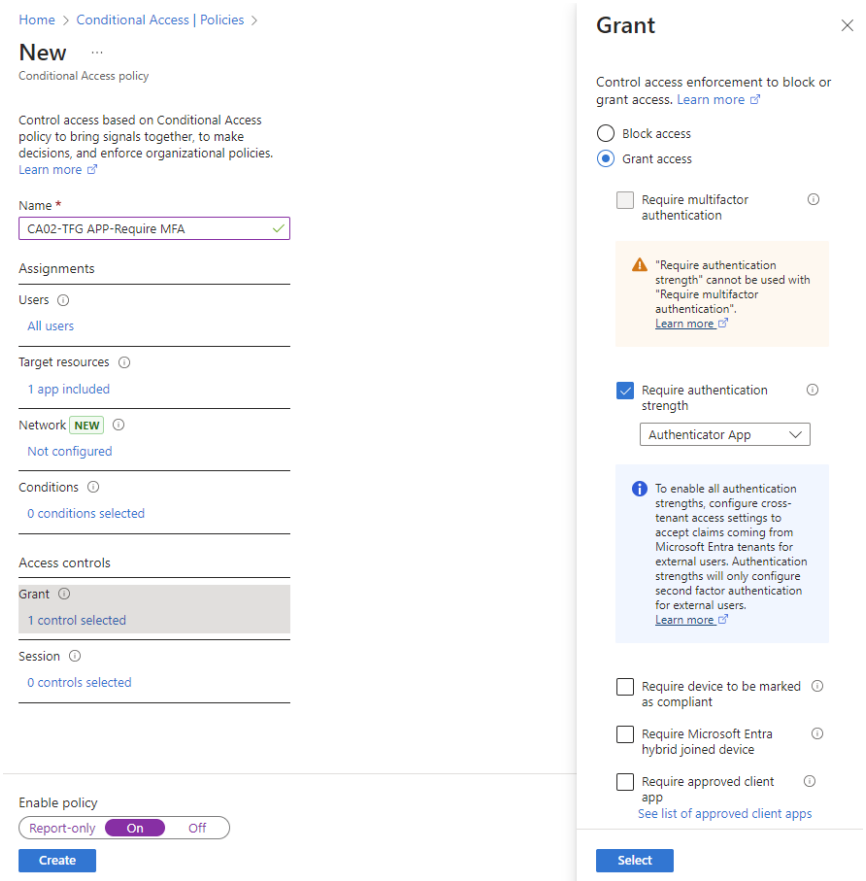

Figure 44 - Creating policy to require MFA (Own elaboration)

In Illustration 44, the policy configuration is shown. As with the previous one, this policy needs to be specifically applied to the app and to any user trying to access. In the "Grant" section, instead of setting it to block, it is configured to allow access as long as the condition is met. Instead of selecting "Require multifactor authentication," it is recommended to use "Require authentication strength" for better security, as this option limits the allowed MFA methods. In this case, only authentication via the Authenticator app will be permitted, as it is considered one of the most secure methods.

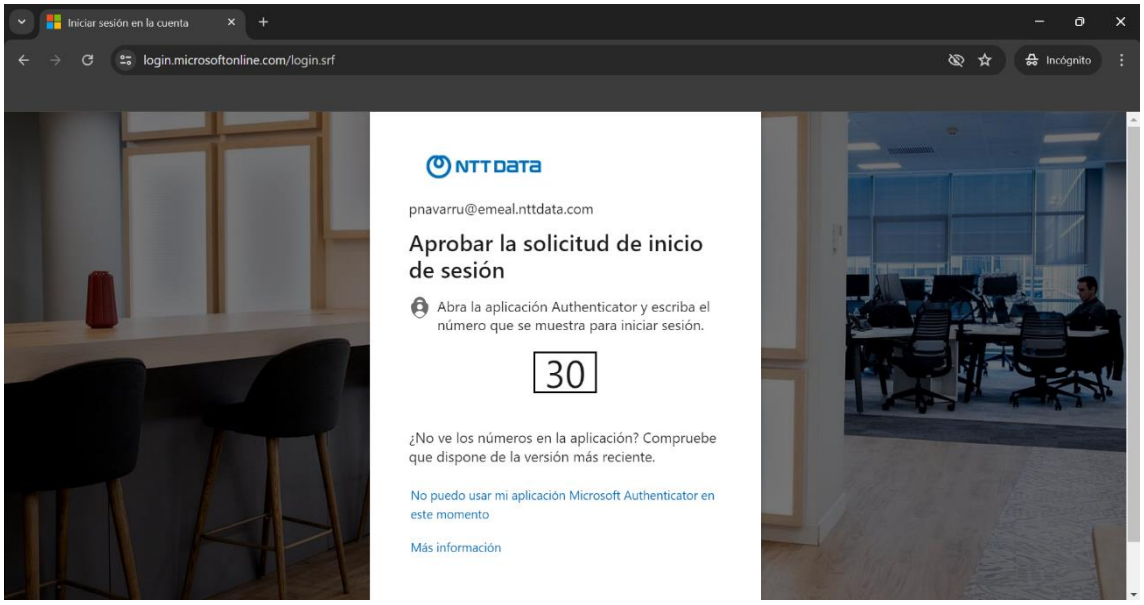

Figure 45 - Authenticator App MFA required (Own elaboration)

As shown in Illustration 45, each time a user wants to access, they will need to complete the MFA challenge by entering the number displayed on the browser screen into the mobile app.

Chapter 7. Conclusions

7.1 Conclusions and future work

This thesis has explored authentication and authorization in web applications through integration with Entra ID, demonstrating how the proper implementation of OAuth flows, such as Authorization Code Flow, significantly enhances security. The transition from Implicit Flow to Authorization Code Flow has been effective in mitigating vulnerabilities, reducing the risk of Man-in-the-Middle attacks, and ensuring that the access token is not exposed in the client's browser. This work highlights the importance of proper security practices to protect web applications in a cloud environment.

Regarding future work, Identity and Access Management (IAM) will continue to be a crucial component in securing a company's digital assets. As more applications migrate their authentication and authorization processes to the cloud, leveraging its benefits, it will be essential to maintain and improve security practices.

One way to expand this work is by conducting a detailed study of more advanced attacks on applications integrated with Entra ID, attempting to exploit vulnerabilities within more secure flows. Additionally, the integration of extra security mechanisms could be considered, such as the use of refresh tokens, the adoption of passwordless authentication technologies, and the implementation of more robust access policies.

Another area of interest is the use of artificial intelligence for attack detection and prevention. Entra ID has recently incorporated "Identity Protection," an AI-based tool that enhances the ability to detect and mitigate threats in real-time by analyzing user behavior during the authentication process. Applying AI-driven solutions in IAM would open new avenues for increasing the security of cloud assets.

7.2 Relationship with the coursed studies

In the final year of my degree, specializing in telematics, I took courses on security and web application development. This thesis delves deeply into authentication and authorization in a web application, integrating it with Entra ID as an identity provider. The project addresses how Entra ID handles these crucial processes and emphasizes the importance of using the appropriate OAuth flow when aiming for the highest possible security for the data managed by the application.

Throughout the thesis, it has been demonstrated how to ensure that the data handled by the application remains protected through the correct implementation of recommended security practices. This includes the configuration and secure handling of access tokens, protection against Man-in-the-Middle attacks, and proper management of user sessions. In this way, the knowledge acquired in security and web development courses is applied and consolidated, providing a practical, real-world understanding of these concepts.

Acknowledgements

This project marks the conclusion of a four-year journey, a milestone achieved thanks to the support of many people who have accompanied me along the way.

I would like to begin by thanking my tutor, José Enrique López Patiño, for teaching me the fundamentals of cybersecurity and guiding me in discovering this fascinating field.

I am also grateful to NTT DATA, not only for providing me with the opportunity to develop professionally but also for the great atmosphere in the Cybersecurity team. In particular, I want to thank my teammates, Dani, Mario, and Alberto, for making every day at work enjoyable and enriching for my growth as a professional. Without a doubt, this team will be the best memory I will take from my internship.

Lastly, I would like to thank my friends, family, and my couple, who have been by my side throughout this journey. They are my main source of support and have kept me strong during the most challenging times.

Literature

[*] Okta, «Client Credentials Flow,» [En línea]. Available: https://auth0.com/docs/get-started/authentication-and-authorization-flow/client-credentials-flow. [Último acceso: 15 Mayo 2024].

[*] Okta, «Implicit Flow with Form Post,» [En línea]. Available: https://auth0.com/docs/get-started/authentication-and-authorization-flow/implicit-flow-with-form-post. [Último acceso: 11 Mayo 2024].

[*] Okta, «Resource Owner Password Flow,» [En línea]. Available: https://auth0.com/docs/get-started/authentication-and-authorization-flow/resource-owner-password-flow. [Último acceso: 15 Mayo 2024].

[*] Okta, «Device Authorization Flow,» [En línea]. Available: https://auth0.com/docs/get-started/authentication-and-authorization-flow/device-authorization-flow. [Último acceso: 16 Mayo 2024].

[*] Cloudflare, «¿Qué es un proveedor de identidad (IdP)?,» [En línea]. Available: https://www.cloudflare.com/es-es/learning/access-management/what-is-an-identity-provider/. [Último acceso: 23 Abril 2024].

[*] R. Neto, «Flujos de autorización con OAuth 2.0 y OpenID Connect,» 28 Febrero 2023. [En línea]. Available: https://rafaelneto.dev/blog/flujos-autorizacion-oauth-2-0-openid-connect/. [Último acceso: 30 Abril 2024].

[*] ICHI.PRO, «La guía completa de los protocolos OAuth 2.0 y OpenID Connect,» [En línea]. Available: https://ichi.pro/es/la-guia-completa-de-los-protocolos-oauth-2-0-y-openid-connect-59286684942618. [Último acceso: 30 Abril 2024].

[*] Okta, «Authorization Code Flow with Proof Key for Code Exchange (PKCE),» [En línea]. Available: https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow-with-pkce. [Último acceso: 10 Mayo 2024].

[*] Okta, «OAuth 2.0 Authorization Code Flow,» [En línea]. Available: https://www.oauth.com/playground/authorization-code.html. [Último acceso: 10 Mayo 2024].

[*] Microsoft, «What is Microsoft Entra Privileged Identity Management?,» 27 Octubre 2023. [En línea]. Available: https://learn.microsoft.com/en-us/entra/id-governance/privileged-identity-management/pim-configure. [Último acceso: 27 Abril 2024].

[*] Microsoft, «Define identity licensing,» [En línea]. Available: https://learn.microsoft.com/en-us/training/modules/explore-identity-azure-active-directory/11-define-identity-licensing. [Último acceso: 8 Mayo 2024].

References

[1] Microsoft, «Shared responsibility in the cloud,» 29 Septiembre 2023. [En línea]. Available: https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility. [Último acceso: 10 Abril 2024].

[2] F. Richter, «Amazon Maintains Cloud Lead as Microsoft Edges Closer,» Statista, 2 Mayo 2024. [En línea]. Available: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/. [Último acceso: 24 Mayo 2024].

[3] Microsoft, «Active Directory Domain Services Overview,» 17 Agosto 2022. [En línea]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview. [Último acceso: 25 Abril 2024].

[4] Microsoft, «What is Conditional Access?,» 19 Marzo 2024. [En línea]. Available: https://learn.microsoft.com/en-us/entra/identity/conditional-access/overview. [Último acceso: 22 Mayo 2024].

[5] Okta, «Refresh Tokens,» [En línea]. Available: https://www.oauth.com/oauth2-servers/making-authenticated-requests/refreshing-an-access-token/. [Último acceso: 10 Mayo 2024].

[6] Okta, «The Authorization Request,» [En línea]. Available: https://www.oauth.com/oauth2-servers/authorization/the-authorization-request/. [Último acceso: 10 Mayo 2024].

[7] IETF, «The OAuth 2.0 Authorization Framework,» Octubre 2012. [En línea]. Available: https://www.rfc-editor.org/rfc/rfc6749.html. [Último acceso: 2 Mayo 2024].

Annexes

Anexo I – Glossary of terms

Cloud: Computing environment based on delivering computing, storage, and application services over the internet, enabling remote access to resources and dynamic scalability.

On-premise: IT infrastructure and resources physically located within an organization's premises, managed and maintained internally.

Azure: Microsoft's cloud services platform providing a wide range of services for computing, storage, databases, networking, and analytics.

Identity Access Management (IAM): Set of policies and technologies to secure and manage identities and access within an organization.

Identity Provider (IdP): Entity that provides authentication and identity management, allowing users to access multiple services with a single identity.

Active Directory (AD): Microsoft directory service for managing identities and resources within a Windows network.

Microsoft Entra ID: Microsoft's cloud-based identity and access management platform, previously known as Azure Active Directory.

Tenant: Dedicated instance of Entra ID that an organization uses to manage its identities and access.

Enterprise Applications: Applications configured in Entra ID for authentication and authorization, including SaaS applications and internally developed applications.

App Registrations: Formal registration of an application in Entra ID, configuring its access, permissions, and other necessary aspects for integration with Entra ID.

Service Principal: Identity created in Entra ID for an application, enabling it to interact with other resources and services within the directory.

Named location: Location defined by IP addresses used in access policies to either allow or block access.

Multifactor Authentication (MFA): Authentication method requiring more than one form of verification to prove the identity of a user.

Privileged Identity Management (PIM): Tool that helps manage, control, and monitor access to privileged roles in Microsoft Entra ID.

Risk Detections: Functionality in Microsoft Entra ID that identifies and notifies of suspicious login activities.

Conditional Access: Security tool that allows setting policies to control access to resources based on specific conditions.

Zero Trust: Security model that does not implicitly trust any entity, continuously verifying the identity and context of each access to resources.

Authentication: Process of verifying the identity of a user or system, ensuring that the entity attempting to access a resource is who they claim to be.

Authorization: Process that follows authentication to determine whether a user or system has permission to access a resource or perform a specific action.

OAuth 2.0: Authorization protocol allowing applications to gain limited access to user resources on a serve.

OpenID Connect (OIDC): Authentication protocol based on OAuth 2.0 that allows identity verification based on authentication performed by a server.

Client: Application that interacts with the user and with Entra ID.

Browser (Frontend): Program that allows users to access and view web content, interpreting and executing HTML, CSS, and JavaScript code.

Backend: The application's server where requests are processed.

Authorization Code Flow: OAuth 2.0 flow allowing applications to obtain access tokens via an authorization code exchanged for a token.

PKCE (Proof Key for Code Exchange): OAuth 2.0 extension that enhances the security of the Authorization Code Flow by mitigating code interception attacks.

Client Credentials Flow: OAuth 2.0 flow in which an application obtains an access token using its own credentials without user involvement.

Implicit Flow: OAuth 2.0 flow in which applications receive tokens directly in the browser without an intermediate authorization code, for single-page applications.

Resource Owner Password Flow: OAuth 2.0 flow where an application directly manages user credentials to obtain tokens from the authorization server.

Device Authorization Flow: OAuth 2.0 flow designed for devices without a user interface, such as IoT devices, to obtain access tokens.

URL: Specific address used to access resources on the Internet, indicating the location of a resource and the protocol to retrieve it.

HTML: Standard markup language for creating and designing web pages, defining the structure and content of a page using tags and attributes.

Script: Set of programming instructions that are automatically executed within an environment, such as a webpage or application.

Python: Interpreted high-level programming language known for its readability and simplicity.

Flask: Python microframework for quickly creating web applications.

Microsoft Graph: API for accessing Microsoft 365 data and other Microsoft services, enabling integration and manipulation of this data in applications.

Client secret: Key used by applications to authenticate their identity with the service provider and request access tokens securely.

Wireshark: Network analysis tool that captures and examines real-time data, used for diagnosing issues and detecting network intrusions.

Loopback: Virtual network interface used by systems to send and receive data to themselves. The IP address 127.0.0.1 is commonly used for local network tests.

Man-in-the-Middle (MitM): Type of attack where a third party intercepts and alters communication between two parties without their knowledge. The attacker positions themselves between the sender and receiver, potentially stealing data or injecting malicious information.

JSON Web Token (JWT): An open standard for a compact, self-contained token used to securely transmit information between two parties as a JSON object.

Anexo II – Sustainable Development Goals

Relationship of the project with the Sustainable Development Goals (SDGs).

Sustainable Development Goals	High	Mid	Low	Do not proceed
SDG 1. No poverty				X
SDG 2. Zero hunger				X
SDG 3. Good health and well-being				X
SDG 4. Quality education				X
SDG 5. Gender equality				X
SDG 6. Clean water and sanitation				X
SDG 7. Affordable and clean energy				X
SDG 8. Decent work and economic growth	X			
SDG 9. Industry, innovation and infrastructure		X		
SDG 10. Reduced inequalities				X
SDG 11. Sustainable cities and communities				X
SDG 12. Responsible consumption and production				X
SDG 13. Climate action			X	
SDG 14. Life below water				X
SDG 15. Life on land				X
SDG 16. Peace, justice and strong institutions		X		
SDG 17. Partnerships for the goals				X

SDG 8. Decent work and economic growth: Improving web application security enhances productivity in the workplace, thus promoting economic growth.

SDG 9. Industry, innovation, and infrastructure: Migrating organizations to identity management in cloud environments improves operational efficiency, fosters innovation, and contributes to the development of modern and resilient infrastructure.

SDG 13. Climate action: One of the advantages of cloud environments is the optimization of resources, which reduces energy consumption and consequently lowers the carbon footprint, helping in the fight against climate change.

SDG 16. Peace, justice, and strong institutions: Securing web applications and creating a strong and secure identity management environment strengthens institutions, promoting justice and trust in the technological field.