Authorization and OAuth 2.0 in Frends Integration Platform

UNIVERSITY OF TURKU
Department of Computing
Master of Science (Tech) Thesis
Software Engineering
June 2024
Juho Ollila

Supervisors:
Ossi Galkin (Frends Technology Oy)
Ville Leppänen (University of Turku)
Jarko Papalitsas (University of Turku)

UNIVERSITY OF TURKU
Department of Computing

Juho Ollila: Authorization and OAuth 2.0 in Frends Integration Platform

Master of Science (Tech) Thesis, 70 p.
Software Engineering
June 2024

Authorization has an integral role in many digital systems, such as APIs and integration platforms. OAuth 2.0 is one of the most popular authorization methods available, and many people use it daily.
This thesis investigates various authorization models used in integration platforms, by comparing these methods across different platforms, identifying typical privacy and security threats against APIs and integration platforms, and analyzing how the Frends Integration Platform addresses these threats.
The research questions addressed in this thesis:

- What are the biggest differences in authorization in different integration platforms?
- What are the typical privacy and security threats that authorization mechanisms are designed to prevent in integration platforms?
- How does Frends Integration Platform prevent the typical privacy and security threats related to authorization and APIs?
- How different OAuth 2.0 authorization flows can currently be implemented in Frends Integration Platform and could they be improved somehow?

Through the research, the thesis reveals differences in how integration platforms handle authorization, which are primarily affected by the architecture and the primary use case of the platforms. It also identifies key privacy and security threats against these platforms such as unauthorized access, data breaches, and man-in-the-middle attacks, and identifies Frends Integration Platforms security methods against these threats. Additionally, the thesis provides research on how OAuth 2.0 authorization flows can be implemented into an API deployed in Frends.

Keywords: OAuth 2.0, authorization, integration platform, authorization models

Contents

List of Figures

1 Introduction

In today's technology-driven world, authorization is used by many almost every day in one way or another. Authorization can be done in many ways with multiple different kinds of methods, for example, OAuth 2.0[1], Security Assertion Markup Language or SAML[2], JSON Web Tokens or JWTs[2], and API Keys[3]. All of the methods have their own advantages, disadvantages and use cases. And it is not uncommon that multiple methods are used at the same time to ensure security. It is essential for the developers to know which type of authorization method is best for their specific application, for example, client-to-client integration.

In many types of applications like integration platforms and APIs, OAuth 2.0 is currently the industry standard for authorization. Integration platforms are software systems designed for implementing, deploying and managing integrations and APIs. In the Frends Integration platform, OAuth 2.0 is one of the most used authorization methods, but it is yet to be documented fully for the platform. Many developers are not aware of all the functionalities OAuth 2.0 provides and which of the functionalities have been implemented in the platform. There are many different ways that authorization can be done in the Frends Integration Platform, sometimes referred to as Frends. In Frends, all authorization types can be implemented, but some require the developer to write their own tasks, instead of using ready-made tasks supported by the platform. The thesis looks at how the authorization flows for OAuth 2.0 can currently be implemented on the platform and how that process could be improved upon in Frends version 5.6.

Authorization helps in protection against different privacy and security threats. The thesis identifies some of the typical threats that authorization in integration platforms aims to prevent. It also explores how some of these threats are addressed in Frends, and what methods Frends implements to protect against them. The platform is being constantly tested and it has been extensively security tested, so all threats should be addressed one way or another.

1.1 Research questions

- **RQ1**: What are the biggest differences in authorization in different integration platforms?
- **RQ2**: What are the typical privacy and security threats that authorization mechanisms are designed to prevent in integration platforms?
- **RQ3**: How does Frends Integration Platform prevent the typical privacy and security threats related to authorization and APIs?
- **RQ4**: How different OAuth 2.0 authorization flows can currently be implemented in Frends Integration Platform and could they be improved somehow?

1.2 Structure of the thesis

System-to-system Authorization is described in Chapter 2. It provides general information on authorization, different authorization methods and OAuth 2.0. This chapter works as background information for the whole thesis. In Chapter 3 authorization is being looked at from the point of view *integration platform*, by comparing the authorization in the different platforms, which answers **RQ1**. Also, this chapter identifies typical privacy and security threats, answering **RQ2** Chapter 4 is about the Frends Integration Platform. It provides basic information on the platform, platform architecture, and information on integration development on the platform. This chapter works as the base for the next two chapters. Chapter 5 discusses authorization in Frends. The chapter analyses how the privacy and security threats identified before are handled in the platform. This chapter aims to answer **RQ3**. In Chapter 6 Frends is being evaluated from the point of view of OAuth 2.0. It describes different OAuth 2.0 authorization flows, and how they can be implemented currently on the platform, the best practices for using them, and if there are any improvements to be made. This chapter aims to answer the final research question **RQ4**. The main contributions, practical applications, and limitations are discussed in the Chapter 7. Chapter 8 contains the conclusion of this work, summarizing the findings, and identifying possible future research.

2 System-to-system Authorization

2.1 General information

2.1.1 Core concepts

Authorization is the process of giving the user or a system the level and type of access to different resources. It determines who can do what with the given resources. Once the user or system is authenticated with their credentials, authorization comes into play.[4] The data access, the level and ability to make changes etc. are collectively known as client privileges.

In organizations, authorization does not usually give everyone the same access to all the data. Different clients have access to different resources. By restricting a client from being able to access only resources needed for their work, the organization can protect their data better. Restricting access to files can also help the clients not have to look through every file in the system to find resources relevant to them, which makes using the service more efficient. Many organizations follow the principle of least authority when giving clients access to resources. This means clients have only access to resources required for their work and nothing more.[5]

2.1.2 Authorization vs Authentication

Authorization and Authentication are often used interchangeably, but they represent different functions. *Authentication* is the process of verifying the identity of users.

Authentication methods usually require the use of credentials, such as username and password, but also nowadays often require the user to verify their identity by using for example their phone or biometric data, such as fingerprint or face scanning.[6][7]

There are many differences in how authentication and authorization work. While authentication prompts the user to validate their credentials, authorization uses different policies and rules to determine if the user has access to the resources. Authentication uses generally ID Tokens to transmit info, while authorization uses Access Tokens.[7]

Authorization is often done as a static process, while authentication is more of a dynamic process. For example, giving a certain client permission to access particular resources is a static authorization process. Different authorization models, introduced in Section 2.2, are used to make these authorization decisions. After the static authorization process is done, authentication is used to determine and verify the identity of the client. The client can then access the resources it has been given access to with the authorization model.

When talking about authentication and authorization, users are often thought of as actual persons, but in reality, they do not have to be. The user in authentication or authorization can be for example another system requiring access to a separate system, for example, an integration connecting to an external API. The system still needs to be authenticated and authorization needs to be used to determine the permissions of the specific system. In this thesis clients or users usually refer to other systems, rather than actual people.

Authentication in different systems is nowadays often done by using OpenID, which is an authentication protocol. Authorization on the other hand is often done with OAuth 2.0. OpenID and OAuth 2.0 can be used with each other, which they usually are. There are also many other authentication and authorization protocols available.

2.2 Authorization models

There are various different authorization models or methods available. Authorization models are ways of managing access control to resources. Most of the authorization models talk about authorization for users, but the user does not have to be a person, it can for example be another system. One of the most basic authorization models is Discretionary Access Control or DAC. Then comes Mandatory Access Control or MAC, which is much more secure than DAC, but is harder to implement and is not used that widely. The most popular authorization model used today for most use cases is OAuth 2.0, which can be used in combination with many authentication models. JWT tokens are used quite extensively nowadays in authorization, while API keys and SAML tokens are used for both authentication and authorization. There are also many other authorization models not covered in this thesis.

2.2.1 DAC

Basic Information

Discretionary Access Control or DAC is an authorization model, in which security decisions are decentralized to resource owners. In DAC, the resource owners determine who can access their resources and what level of access is permitted to them. This type of access control is used widely in many types of computing environments, such as large-scale network systems, or individual workstations. [8] One of the most commonly seen systems, which use the DAC model is the sharing option in most operating systems. The owner can set the read and write permissions for each user individually within the table. DAC is a fairly basic authorization model, which is why it is not used that much anymore in bigger deployments or networks.

Advantages of DAC

DAC offers a lot of flexibility for the owner. They can tailor the access permissions according to their specific needs and requirements. This flexibility is useful in situations where the different users have different levels of responsibilities and trust. The DAC model also grants the resource owners a high level of autonomy over their resources. The owners can make their own decisions based on their knowledge of the different resources and the risks associated with them.

Systems, which use the DAC model, are often quite straightforward to implement, understand and use, which makes them suitable for individual users and smaller-scale deployments. When a client is in the access control list and has the right permissions, they can access the resource.[9]

Disadvantages of DAC

With the simplicity of the DAC model comes disadvantages compared to more modern authorization models. One security risk in DAC comes from the fact that it is easy for the resource owner to grant excessive permissions to users or overlook possible vulnerabilities, which could lead to unauthorized access to the resources or lead to data breaches.

Larger systems which use the DAC model can become quite hard and complex to manage efficiently and securely. Resource owners might struggle to maintain an accurate list of user permissions, which can lead to users having wrong permissions. For example, if a client is in a group, which has different access rights than the client, managing the permissions becomes complex.

Malicious software and programs can be a problem for the DAC model. Malicious programs can exploit the authorizations of a particular client, which grants the program the client's privileges to access resources. And especially if the client has excessive permissions, the whole system can be vulnerable by the result of a single client.

DAC does not have control over resources, once they have been acquired by a user. The resource owners do not have any control over the flow of information in the system. Resources can be copied from one system to another without any restrictions, which can grant access to the resource without the knowledge and approval of the resource owner. [10][9]

2.2.2 MAC

Basic Information

Mandatory Access Control or MAC[11] is an authorization model, which regulates access to resources based on predefined security policies. The sensitivity is defined with security labels. The security label contains a security level and zero or more security categories. The security level tells the classification of the file for example *Restricted, Confidential, Internal.* The security category then can tell for example which project or group the file belongs to. Every user has their own security labels, which grant them access to only the information their labels allow. The security labels are generally managed by a centralized security administrator or the system administrator of the organization.[11] MAC is usually used for very large systems and platforms such as governments to secure classified resources. It is also used by the insurance industry and banks.[12]

Advantages of MAC

The MAC model offers several advantages for better system security. When compared to the DAC model, MAC offers a higher level of security by enforcing strict access policies and rules. MAC can reduce the risk of data breaches, unauthorized access, and privilege escalation by assigning labels to the resources and users and limiting access based on these labels.

One of the main advantages of the MAC model is that it allows precise control over resource access, which enables administrators to define strict access permissions for individual processes and users. This precise control enhances security greatly by making sure that users only have access to the resources which are necessary for their individual tasks, minimizing the potentiality and scale of data breaches. It also offers some protection against data leakage by enabling administrators to control the flow of data within the system. They can restrict the flow of information between systems if needed. [11][13][14]

Disadvantages of MAC

There are several disadvantages to using MAC and for many applications there are better alternatives available such as OAuth 2.0. To implement the MAC model effectively in a system, requires predetermined planning. It also requires a lot of management due to constantly updating object and account labels. The administrators have to maintain and define a comprehensive set of security labels and rules. This can become complex in larger-scale systems with a lot of different types of resources, and diverse access requirements and user roles.[10][11]

The strict nature of the access rule enforcement in the MAC model limits user flexibility and productivity. If managed poorly, MAC policies may implement restrictions, which could make it difficult for users to access resources and perform certain tasks. [10][11]

2.2.3 RBAC

Basic Information

Role-based access control or RBAC[15] is a popular authorization and access control model, which helps to reduce the complexity of security administration, and supports the review of permissions assigned to users. RBAC is designed to be used with large and structured organizations of people. In RBAC access decisions are based on the *role* of the users. The roles can be for example assigned based on the tasks the user does and their responsibilities within the organization.[15] Nowadays RBAC is used quite a lot in organizations and services, for example, many integration platforms use it to handle access control for the users in the platform, such as Frends Integration Platform and Mulesoft's Anypoint Platform.

Advantages of RBAC

RBAC has many advantages for structured organizations. It simplifies the management of access control by assigning permissions to different roles rather than individual users. This reduces complexity, which is important in large organizations with large amounts of resources and users.[15] It also has the advantage of having users only be able to access the resources they need and having the ability to change the permissions quickly if needed. This increases security by minimizing the risk of unauthorized access to resources.[16][17]

Different regulatory requirements, such as GDPR, are also taken in to consideration with RBAC.. It provides clear documentation on the roles and permissions of all users.[17]

Flexibility is also one of the reasons RBAC has become so popular, for example, DAC and MAC can both be used as RBAC, if used in a certain way. Adding groups to DAC rather than individual users, makes it act as RBAC. Also using a role graph restricted to a tree in MAC simulates the functionality of RBAC.[18]

Disadvantages of RBAC

RBAC does also have some disadvantages. For example in large and complex organizations, the number of different roles can become somewhat unmanageable. This can affect the management of the roles and some of the benefits of RBAC might be lost. Also setting up a properly managed RBAC requires a good understanding of the structure of the organization and the roles and jobs within it. This setup can become resource-intensive. Even after the initial setup the system requires constant management and maintenance to make sure the roles keep up with the changes within the organization.[16][17]

2.2.4 API Keys

Basic Information

An API key is a secret token, which is provided by a client when they make API calls. Only the client and the server are supposed to know the API key. They are only considered to be secure, when they are used in combination with other security mechanisms, for example, HTTPS/SSL.[3] There are many types of secrets that can be considered as API keys. They essentially only have to be a secret key, which both the client and the server keep secure. API keys can be used for authorization and given permission to access certain resources in the server.

API keys can be added to API requests in different ways. They can be included in query strings[3]:

```
GET /resource?api_key=abcdefg123456
```

or in a request header:

```
GET /resource HTTP/1.1
x-apikey: abcdefg123456
```

or in a cookie:

```
GET /resource HTTP/1.1
Cookie: X-API-KEY=abcdefg123456
```

Advantages of API Keys

There are quite a lot of advantages to using API keys in a system. For example, API keys are simple and straightforward to use. They require minimal configuration and setup, compared to for example OAuth 2.0, which is a much more complex authorization model. They are also easy to implement into any existing systems and applications. This allows developers to easily integrate APIs into their projects without the need for large modifications or adding many dependencies. API keys are also suitable for even large systems with many clients. They are easily scalable and they can be easily generated and distributed programmatically. [19]

Disadvantages of API Keys

API keys produce an inherent security risk because they need to be protected and managed well on both sides, on the resource owner and the client side. If the API key is leaked or exposed, it can be exploited by the wrong parties, resulting in possible service abuse or data breaches.

Using API keys lacks any user identification. The API keys are usually tied to the system or application they are meant for, and not for individual users. Identifying users by their actions in environments with multiple tenants becomes hard without third-party services. [19]

2.2.5 JSON Web Tokens

Basic Information

JSON Web Token or JWT[20] is a compact claims representation format, which was introduced in the RFC7519 document. JWT is designed to be used in space-constrained environments, for example, URI query parameters and HTTP Authorization headers. It transmits information securely between the parties as a base64 encoded JSON Object. The token is digitally signed, which makes it trustworthy. The signing can be done by using either a secret or a public and private key pair. JSON Web Tokens typically consist of three different parts, which are separated by dots. These parts are header, payload, and signature.[20][21]

Example encoded JSON Web Token looks like:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibm
FtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

And when the JWT is decoded the header, payload, and signature are:

```
HEADER:
{
    "alg": "HS256",
    "typ": "JWT"
}
PAYLOAD:
{
    "sub": "1234567890",
    "name": "John Doe",
    "iat": 1516239022
}
SIGNATURE:
HMACSHA256(
    base64UrlEncode(header) + "." +
    base64UrlEncode(payload),
    your-256-bit-secret
)
```

Advantages of JWT

JWTs are much smaller than SAML tokens because JSON is less complex when compared to XML. This smaller size makes JWT good for passing in HTML and HTTP environments. They are also quite secure to use. They can easily use public/private keys for signing, or shared secrets for symmetrical signing.

Popularity of JWTs can be seen in that they are used in many authorization models for example OAuth 2.0. Where they are used for Bearer Tokens to encode all necessary parts into the access token, rather than needing to store them in a separate database. They are also very simple to work with, when compared to SAML, because JSON parsers are commonly included in most popular programming languages. JWT is also easier to process for systems such as mobiles.[20][22][21]

Disadvantages of JWT

Using JSON Web Tokens does not have many disadvantages, but there still are some potential disadvantages for certain use cases. For example, there is currently no way to invalidate a token once it has been issued. This can be a problem in cases where immediate revocation of access is necessary, for example in a case of a compromised token.[20]

2.2.6 Security Assertion Markup Language

Basic Information

Security Assertion Markup Language or SAML is an XML-based framework. It is designed for communicating user entitlement and authentication, and attribute information to the service provider. SAML works much the same way as JWT does, as they are both security token formats. SAML is an older technology than JWT, but it is still used in many different services.[23][24]

Advantages of SAML

Even though there are many newer technologies than SAML, it still has some advantages and benefits. It is a widely adopted standard, which makes it compatible with many different systems and platforms. This allows for seamless integration between different applications.

One of the main benefits of SAML is the support for Single Sign-On or SSO. This allows you to log in to multiple different services with the same credentials. SSO is also supported by other technologies than SAML, such as *OpenID Connect*. It is is also quite a flexible framework. It supports many different types of authentication methods, for example, username and password, and multifactor authentication.[23][25][2]

Disadvantages of SAML

There are also disadvantages to using SAML. As a framework, it is quite complex to use. It requires the developer to have a good understanding of XML and the SAML framework.

Due to the age of SAML, it has limited support for modern protocols. Over the years it has been widely adopted to many services, but it may lack support for newer authentication and authorization protocols such as OpenID Connect and OAuth 2.0, which are often better for modern systems and applications.[23][2]

2.3 OAuth 2.0

Open authorization or OAuth 2.0 is one of the most prominent and flexible authorization frameworks used today. It enables users to grant third-party applications limited access to their resources without having to reveal credentials directly.[1] OAuth 2.0 was originally released in October 2012 in RFC 6749 [26]. It was designed to replace OAuth 1.0, which was released in April 2010. OAuth 2.0 was reworked completely so backwards compatibility is not possible with OAuth 1.0. Since 2012, there have been several additions and modifications to the OAuth 2.0 specifications in the form of new RFCs. For example, some grants have been made obsolete and new ones have been introduced over the years. [27][26]

2.3.1 Core concepts of OAuth 2.0

OAuth 2.0 works as a security layer for APIs, allowing third-party applications to access the resources on behalf of the users. The framework defines these four main roles: resource owner, resource server, client, and authorization server. [1]

Resource owner is an entity which is capable of granting access to a secured and protected resource. The resource owner can be for example a system, granting resource access to other systems. If the resource owner is a person, they are referred to as an end-user.

Resource server is the server which hosts the protected resources, and it is capable of accepting resource requests using access tokens.

Client is the application, which is making the protected resource requests. It needs authorization from the resource owner to make the requests. The client can also be another system or a person.

Authorization server is the server, which issues the access tokens for the clients after a successful authentication by the resource owner. [26] Most common authorization servers used are Auth0[28] and Entra ID[29].

2.3.2 OAuth 2.0 protocol flow

The basic OAuth 2.0 protocol flow shown in Figure 2.1 is an abstract representation of a flow, which illustrates the interaction between the four roles within the OAuth 2.0 authorization flows. Most authorization flows use some of the different steps

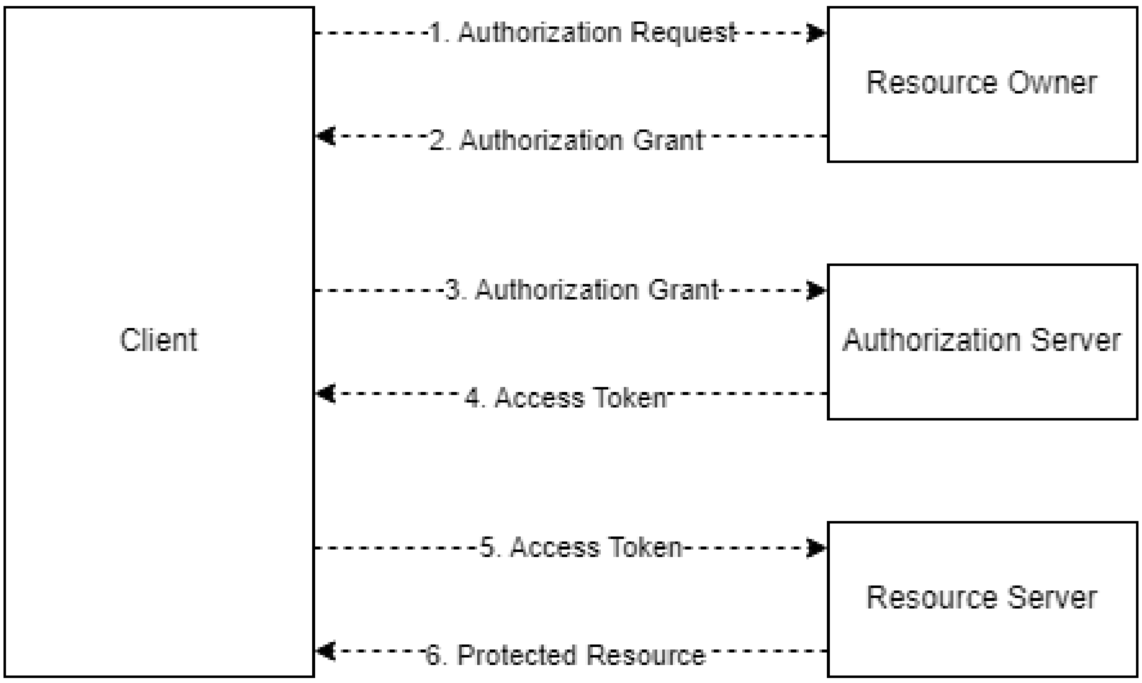


Figure 2.1: Abstract representation of Protocol Flow [26]

described in the protocol flow, some go through all of the steps and some go straight to the access token steps. The different steps are described in the RFC6749 standard document[26]:

1. The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner, or preferably indirectly via the authorization server as an intermediary.

2. The client receives an authorization token, which is a credential representing the resource owner's authorization, expressed using one of the different flow types or using an extension flow type. The authorization flow type depends on the method used by the client to request authorization and the types supported by the authorization server.

3. The client requests an access token by authenticating with the authorization server and presenting the authorization grant.

4. The authorization server authenticates the client and validates the authorization grant, and if valid, issues an access token.

5. The client requests the protected resource from the resource server and authenticates it by presenting the access token.

6. The resource server validates the access token, and if valid, serves the request.

2.3.3 Authorization flows

OAuth 2.0 includes a few different authorization flows or grants as they are sometimes called. In this thesis, they will be referred to as flows. Authorization flows determine which type of authorization is granted to the client. In the latest version of OAuth 2.0 the most common and recommended authorization flows are **Authorization Code** flow and **Client Credentials** flow. There is also Implicit flow and Password flow, but they are not recommended anymore, due to being not secure enough, but still Implicit flow is still quite popular in many different services.[30]

In **Authorization Code** flow, the resource owner gives an authorization code to the client after the authorization server has authorized the resource owner. This flow is used by both confidential and public clients. Nowadays it is recommended that the **PKCE** extension is used with this flow, to make it more secure.[31][32]

Client Credentials flow is often used when the client is also the resource owner, or when the client has been given permission to access the secured resources beforehand. As OAuth 2.0 documentation[33] says "The Client Credentials [flow] type is used by clients to obtain an access token outside of the context of a user."

2.3.4 OpenID

OpenID is an open standard for authentication. It allows applications to verify the users without the need for collecting or storing the user's login credentials.

OpenID is currently one of the major players in digital identity management. It offers a decentralized authentication mechanism for internet users. It simplifies the authentication process and provides the user with the ability to use one set of credentials for multiple different services and sites. OpenID is often used with OpenID Connect, which is an extension of the protocol.[34][35]

2.3.5 OpenID Connect

OpenId Connect is a popular authorization protocol, which is built on top of OAuth 2.0 and OpenID. OpenId Connect is built to standardize authenticating and authorizing users when they sign in to digital services using OAuth 2.0. It is used mainly for authenticating the user, after which OAuth 2.0 is used to check authorization for resources and systems the user can access. For example, when you use your Google account to log in to a site, OpenId Connect is being used for the login service. OAuth 2.0 is focused on authorization, so often some kind of user authentication is needed. While OAuth 2.0 does not require a human user in the process, OpenId Connect makes involving a human user mandatory. OpenId Connect is one of the most popular and flexible choices. [34][35]

2.3.6 Advantages of OAuth 2.0

OAuth 2.0 has many advantages, which is one of the reasons it has become so popular. OAuth 2.0 allows users to log in to services with another service's account, for example logging into StackOverflow with a Google account. SSO is also supported by OAuth 2.0, which makes it so that clients only have to authenticate once to access multiple different systems. SSO is often implemented into OAuth 2.0 with OpenID. OAuth 2.0 also allows a service to access resources on a different service, without having the user need to transfer the resources on their own.[36]

OAuth 2.0 also supports many different flows or grants for authorization, which are all designed for different use cases. Some are more focused on authorization for individual users, and some are designed for system-to-system authorization. All of the different flows and their purpose also make OAuth 2.0 very scalable. So it is also suitable for very large-scale systems with numerous resources and clients.[36][37]

OAuth 2.0 also allows fine-tuning of individual permissions by using scopes, which define the specific permissions granted to a client. This also helps to minimize the risk of data breaches and unauthorized access to resources. Security is also improved by the fact that OAuth 2.0 uses access tokens, such as JWTs, rather than having to expose the client's credentials.[37][38]

Over the years, OAuth 2.0 has become pretty much the industry standard for most authorizations. It is supported by most frameworks, libraries and platforms, and it is used by many of the major tech organizations and companies working around the world. Using a standardized protocol like OAuth 2.0 makes system integration between different services more efficient and simpler. [36][37]

OAuth 2.0 being a collection of multiple frameworks, can in some cases be an advantage. It makes OAuth 2.0 flexible and works with many types of implementations. This can also act as a disadvantage.[39]

2.3.7 Disadvantages of OAuth 2.0

OAuth 2.0 does not have many disadvantages when compared to other authorization methods, but it still has aspects which need improving. OAuth 2.0 has had many additions over the years, which has made it somewhat complicated. OAuth 2.0 protocol has many different grants for different scenarios. Understanding these different grants can be hard for developers, which can delay development.[36][37]

The many modifications and additions to the OAuth 2.0 specification also create a lack of standardization. Building consistent integrations can become hard because there can be many variations in implementations and interpretations across different platforms and providers. [39]

OAuth 2.0 inherently relies on the use of third-party services. For example, it does not have a way for users to authenticate with the service provider. Relying on third-party services can become a security risk, which developers have to be aware of.[39]

OAuth 2.0 has a few vulnerabilities, which are possible with certain flows. Most of the vulnerabilities and exploits are caused by the implementation and not the framework itself. One example of a vulnerability, which is in the framework itself, is a vulnerability which allows the stealing of access tokens using OAuth URL redirects. This is done by manipulating the *redirect uri* parameter to steal the access token from the client's account. This vulnerability is present in the OAuth 2.0 Authorization flow, but it can be prevented by using Proof Key for Code Exchange or PKCE.[40][38]

2.3.8 PKCE

PKCE[40] is an extension for the Authorization Code grant to improve security. It is designed to prevent URL redirects, authorization code injection attacks, and Cross-Site Request Forgery or CSRF attacks. PKCE ensures that the authorization server knows that the system which starts the flow is the same system, which finishes it. Without PKCE the authorization server can not tell if the caller of the flow and the redeemer of the authorization code are the same system.

To ensure that the caller system and redeemer system are the same, PKCE uses a proof key. The proof key is needed to be able to redeem the authorization code, so malicious systems can not redeem it.

2.3.9 Scopes

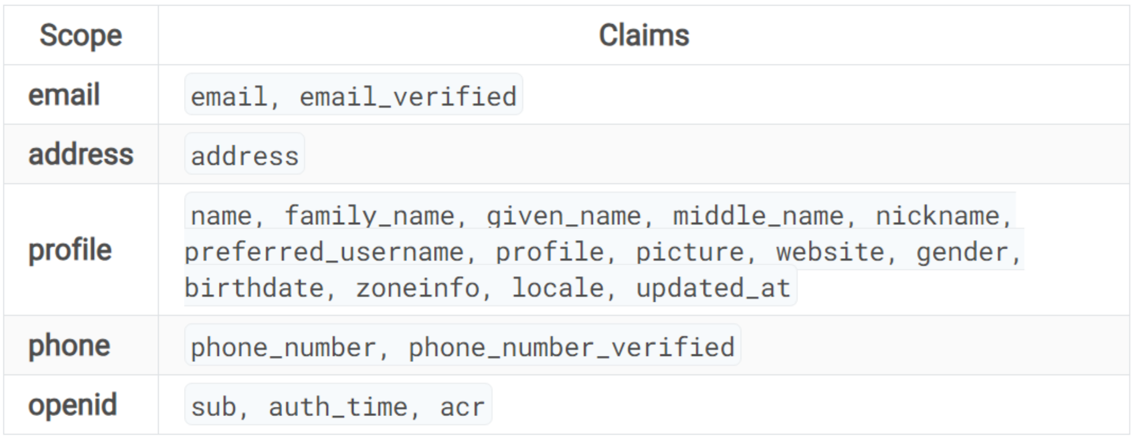
In OAuth 2.0 *scopes* give the ability to limit the amount of access that is granted to specific access tokens. APIs and authorization servers can be implemented to use any scope or combination of scopes. Scopes can be for example used to give access tokens only READ permissions to a resource, while other access tokens might get WRITE access also. So if a client with READ access tries to call an endpoint which requires WRITE access, the request will be denied. When a request is received by an API, the credentials of the caller will be checked whether they are allowed to perform the request or not.[41][42]

2.3.10 Claims

Claims are part of access tokens in OAuth 2.0. They contain more information about the granted access or the client. Usually, claims are used only when the caller is trusted, because the caller can essentially decide themselves what the claims contain, and if the information is correct. In theory, claim values can be anything, such as a string, a list, a number, a Boolean or any other type of value. But usually, there are a few basic claims being used. For example in OpenID Connect, claims are grouped and mapped into certain scopes, which contain a standardized set of claims shown in Figure 2.2.[43]

2.3.11 OAuth 2.1

OAuth 2.1 is an in-progress project, designed to compile and simplify the most common components of OAuth 2.0. Since OAuth 2.0 has been out for so many years, it has gotten quite complicated with all the updates. OAuth 2.1 aims to capture the best current practices of OAuth 2.0 under a single name. OAuth 2.1 is also designed to focus more on security than its predecessor.[27] Many of the specification details are derived from the *OAuth 2.0 Security Best Current Practices*



Scope	Claims
email	email, email_verified
address	address
profile	name, family_name, given_name, middle_name, nickname, preferred_username, profile, picture, website, gender, birthdate, zoneinfo, locale, updated_at
phone	phone_number, phone_number_verified
openid	sub, auth_time, acr

Figure 2.2: OpenID Connect standardized groupings of claims [43]

document[44]. There are some major differences between OAuth 2.0 and OAuth 2.1, as there were major differences between OAuth 1.0 and 2.0. The major differences as stated in OAuth 2.1 documentation[45] are:

- PKCE is required for all OAuth clients using the authorization code flow. PKCE is an extension of the Authorization Code flow, which prevents authorization code injection attacks and Cross-Site Request Forgery or CSRF.

- Redirect URIs must be compared using exact string matching.

- The implicit grant is omitted from this specification. The implicit grant or flow was legacy simplified OAuth flow in OAuth 2.0. It is not recommended anymore and some services even prohibit the use of it completely due to its risks. It has mostly been replaced with PKCE extension.

- The Resource Owner Password Credentials grant is omitted from this specification. Password grant is a legacy way of exchanging user's credentials for an access token. In the latest OAuth 2.0 Security Best Current Practice document, the password grant has been disallowed completely.

- Bearer token usage omits the use of bearer tokens in the query string of URIs. Bearer tokens, which are also known as access tokens, must be kept secure because they allow access to protected resources. Most bearer tokens are JSON Web Tokens or JWTs. Clients use the tokens to make API calls to the resource server. Then the resource server checks for the permissions of the client from the token. At first bearer tokens were allowed to be included in headers, POST bodies, and query strings. OAuth 2.1 plans to remove the ability to use them in query strings. This keeps the bearer token more secure because query strings are actually never private. [46]

- Refresh tokens for public clients must either be sender-constrained or one-time use.
- The definitions of public and confidential clients have been simplified to only refer to whether the client has credentials.

3 Authorization on integration platforms

Integration platforms are software solutions, which usually provide different tools for functionalities such as creating integrations, APIs and workflow automation. There are many different integration platforms available. This chapter goes through some of the most popular ones and how Privacy and Security threats in authorization are handled in them. And also how authorization is done on the platforms. The integration platforms the chapter covers are IBM App Connect[47], Mulesoft[48], Workato[49], and Boomi AtomSphere[50]. This chapter aims to answer both RQ1 and RQ2.

3.1 IBM App Connect

3.1.1 General Information

IBM App Connect is an integration solution, designed to connect *Software as a Service* or SaaS applications and data. It uses a no-code interface to build integration flows. App Connect can be deployed on-premise and on-cloud, or it can be deployed as a fully managed Integration Platform as a Service on Amazon Web Services (AWS).

App Connect has many prebuilt connectors, and it supports many integration technologies, such as API-led, and event-driven messaging. There are prebuilt connectors for many popular systems, for example, Gmail, Slack, Salesforce, and Gitlab.

3.1.2 Authorization

In IBM App Connect, authorization is used to determine who has permission to access, use, and modify the integration server or node and its resources. If administration security is enabled on the integration server or node, authorization is required for performing any tasks, for example configuring an integration server, accessing message flow queues, and taking actions in the IBM App Connect Enterprise toolkit.

Configuring the integration server can be done by modifying the *server.conf.yaml* file, which requires correct permissions to access. Accessing queues requires authorization to determine for example which users can input messages to an input queue of a message flow. The use of the IBM App Connect Enterprise toolkit, which can be used for example to deploy message flows to integration servers, requires the right permission. All these user permissions are determined in the administration security for the specific integration server or node.

Message Flow authorization

IBM App Connect provides a Security Manager, which can be used to control access to a specific message in a message flow. An integration server can be configured to use a security profile, which provides the ability to control message flow access based on an identity This identity is provided in the message, that is carried through a message flow. This security mechanism is independent of the message format and transport type.[51]

If a security profile is not used for authorization in a message flow, a fixed identity can be specified. Fixed identity is usually used for authorization when the message flow interacts with an external system. The fixed identities are stored in a secure repository in the integration server. If message flow security is not enabled, all messages are processed, and the integration server service identity is used as a proxy identity for all of the message instances. [51]

Sometimes message flow authorization also utilizes external security providers. Out of the box, IBM App Connect supports these external security providers, which are also known as *Policy Decision Points* or PDPs[51]:

- Tivoli Federated Identity Manager, which can be used for authentication, mapping, and authorization.
- WS-Trust V1.3 compliant Security Token Service, which can also be used for authentication, mapping, and authorization
- Lightweight Directory Access Protocol or LDAP, which can be used for authentication and authorization

These external PDPs can be used for verifying the permissions of the provided identity tokens. If WS-Trust is configured to be the authorization provider for the flow, the Security Manager uses a Security Token Service, for example, Tivoli Federated Identity Manager, to validate that the provided identity has the correct permission for accessing the message flow.[52]

If LDAP is configured to be the authorization provider for the message flow, the identity verification is a bit simpler than WS-Trust. In this case, the Security Manager calls the LDAP server to validate that the identity token is a member of the LDAP group. These LDAP groups can be configured in the security profile. [52]

3.1.3 Permission Management

For most of the permission management in IBM App Connect, administration security is used. By default, it is turned off, but it is recommended to enable it in the integration node or integration server. With administration security enabled, users can be given permission to perform different tasks against the integration server or node, and its resources. If administration security is left disabled, all users are able to perform actions against the integration node and all the integration servers it manages. If an integration server is not managed by a node, administration security has to be enabled separately on it.[53]

Administration security in IBM App Connect can handle both the authorization side of permission management, as well as authentication. The access of users can be managed by allowing users with specific roles to complete tasks and actions on resources. The authorization is always checked when a request to view or change resources or properties is received. The requests are denied if the user is not authorized. For integration nodes, the platform supports three different modes of authorization, which are file-based authorization, queue-based authorization, and LDAP authorization modes.

3.1.4 API Management

IBM App Connect provides good capabilities for API management. It enables the use of many different authorization methods such as RBAC, OAuth 2.0, API keys, and JWT, alongside having Access Policies for managing different aspects of the API. IBM App Connects API management system functions well with most major identity providers such as IBM Cloud Identity, Entra ID, which was formerly known as Azure AD, and Google Identity. It also supports the use of custom authorization servers. It also has a vault to store API keys for each environment, but it does not have an integrated way to automatically generate API keys.

3.2 MuleSoft's integration platform

3.2.1 General Information

Mulesoft's integration platform AnyPoint is one of the most widely used integration platforms currently on the market. It is developed by MuleSoft. The integration platform is nowadays part of Salesforce Integration Cloud, but this section focuses on the AnyPoint integration platform specifically. The platform is used to connect enterprise and SaaS applications in the cloud and on-premise.[48]

The AnyPoint platform consists of many different components for example AnyPoint Security, AnyPoint Design Center, AnyPoint Management Center, AnyPoint Exchange, Runtime engine and services, API-led connectivity, Application network and DataWeave. Most of the relevant parts of the platform for this section are located in the Runtime engine and services.[54]

MuleSoft's integration platform uses API-led integration development. It allows developers to create, develop and execute APIs within the platform. There are different roles for different APIs, for example, *Process APIs*, which are used for processing data and performing logic on it, *Experience APIs* handle data formatting for different use cases like apps, mobile devices, desktops, etc.[54]

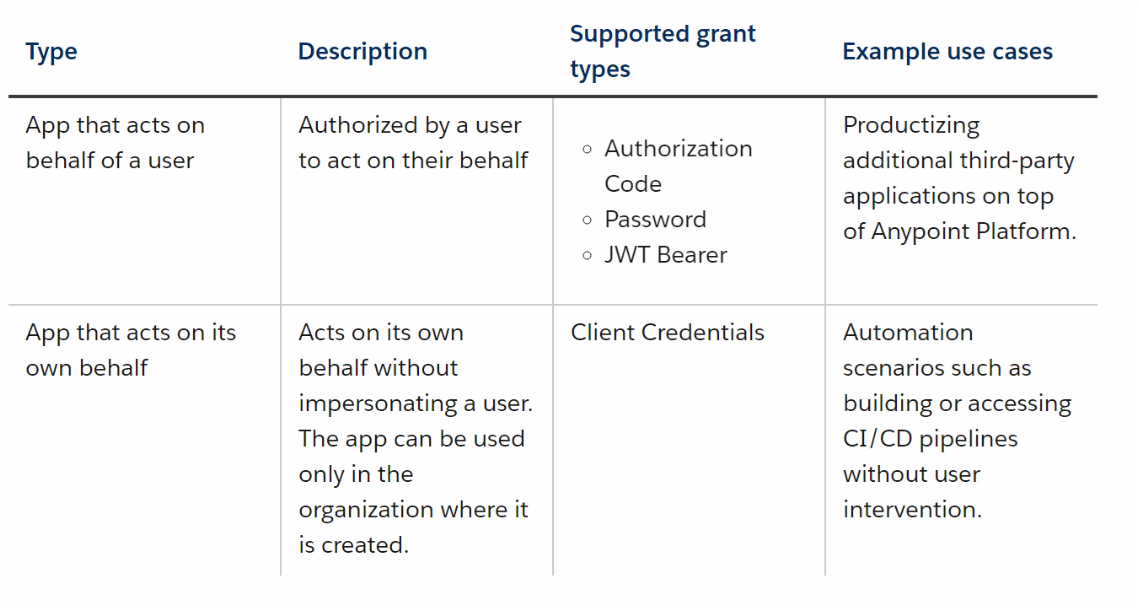
3.2.2 Authorization

Mulesoft's integration platform supports many types of authorization methods. For example, it supports OAuth 2.0 flows like Authorization Code, Implicit, Resource Owner Password Credentials and Client Credentials flows. The platform also uses *Spring Security*[55] for Mule Component authorization. Mule Components are the building blocks that Mule applications are built with.[54]

For component authorization, either Spring Security LDAP provider or Mule security provider can be used. Both of these work much the same way as Message

Flow authorization works on IBM App Connect.[54][52]

External applications connect to the platform using APIs with OAuth 2.0 and OpenID Connect. This feature is called Connected Apps within the platform. All of the actions taken by the connected apps are audited, and the access can be revoked at any time if necessary. Different types of connected apps support different OAuth 2.0 flows as shown in Figure 3.1.[54]



Type	Description	Supported grant types	Example use cases
App that acts on behalf of a user	Authorized by a user to act on their behalf	◦ Authorization Code ◦ Password ◦ JWT Bearer	Productizing additional third-party applications on top of Anypoint Platform.
App that acts on its own behalf	Acts on its own behalf without impersonating a user. The app can be used only in the organization where it is created.	Client Credentials	Automation scenarios such as building or accessing CI/CD pipelines without user intervention.

Figure 3.1: Two types of connected apps[56]

3.2.3 Permission management

Users in the AnyPoint Platform belong to organizations and have permissions set to them. These permissions can be granted to individual users or to teams. Teams are groups of users, which the organization administrator can manage. The permissions can then be assigned according to the organizational structure. AnyPoint Platform used to handle permissions based on roles, but this has since been deprecated and replaced with handling it through the use of teams. However, the authorization method used can still be classified as a sort of Role-Based Access Control or RBAC.[54]

Every user added to an organization is first automatically added to the *Root Organization* team. Then under the root teams, other more specialized *child teams* can be added. These teams have more specific permissions, which are based on the responsibilities of the members of the team. The child teams also inherit the permissions from their parent team, so each child team should have more specialized permissions. Members inside teams also have two different roles: *maintainer*, who can manage team membership and has the permissions assigned in the team structure, and *member*, who only receives the permissions assigned in the structure. If a user requires permissions, which land outside of the permissions of their team, they can be assigned individual permissions, without having to assign the permissions to the whole team.[54]

Inside the AnyPoint Platform permissions can also be granted to external organizations. Both organizations have to add each other to a list of trusted organizations which allows organizations to collaborate, by for example sharing API specifications, or other assets with each other, without having to make the resources public.[54]

3.2.4 API Management

API management is an important part of many integration platforms, so the AnyPoint platform has its own API Manager. The API manager allows the management, governing, and securing of APIs and API keys in the platform. It is often recommended to use some kind of authorization method for APIs to keep them secure. The API manager includes the ability to use many different authorization methods but OAuth 2.0 is one of the most flexible ones supported. The platform also supports generating API keys specific for each environment, and it has a secure vault to store and access the API keys.

The API manager also includes its own OAuth 2.0 Provider, which can be deployed to any MuleSoft Platform. It provides the ability to get access tokens, through the basic four different OAuth 2.0 flows, and enforce and monitor them. The flows supported are authorization code, implicit, resource owner password credentials, and client credentials. The Mule OAuth 2.0 Provider includes the basic OAuth 2.0 endpoint for token validation, authorization, access, and token revocation. It also provides the ability to modify OAuth 2.0 *scopes*, which are designed to limit access to resources protected with OAuth 2.0.

3.3 Boomi

3.3.1 General Information

Boomi provides an Integration Platform as a Service or iPaaS. This service is called Boomi Enterprise Platform. It uses a low-code platform, with a drag-and-drop user interface to create the integrations. These integrations can connect any combination of on-premise and cloud applications, for example, cloud to cloud, SaaS to SaaS, cloud to on-premise or on-premise to on-premise.[57]

Boomi Enterprise platform works as a single-instance, multi-tenant architecture. This enables the sharing of physical computing resources between multiple different tenants and customers securely. Boomi also has its own runtime engine, which allows the integrations to be deployed wherever needed, be it in Boomi's own *Atom Cloud*, in a cloud, or on-premise.

3.3.2 Authorization

Boomi Enterprise Platform supports the use of many different authorization methods for various different purposes. For example, OAuth 2.0 can be used to connect many different applications to the platform. These applications are usually connected with premade Application connectors, which provide the connection solution for specific software applications. There are application connectors for software applications such as Amazon, Salesforce, and Netsuite, among many others. Based on the software being connected to, different authorization methods are used.[57]

A common authorization method for application connectors seems to be OAuth 2.0 Authorization code flow, for example, Google Storage connection uses it. It's often used for both authentication and authorizations. The connector uses the given Client ID, Client Secret, and Scope to get the access token from the identity platform being used. The access token is then sent with every request to the connected application. Many of the application connectors can manage refresh tokens automatically.[57]

3.3.3 Permission Management

Boomi uses *accounts* for permission management. The Boomi documentation[57] describes accounts as workspaces, which are used to perform actions provided by the Boomi Enterprise Platform. East account must have at least one user, and one account can be shared by multiple different users. Each user can be given different Access Roles, which determine which resources inside the workspace they can access. Boomi also offers the option to use Account Groups for bigger accounts. All the users added to a group gain the same authorization and permissions. The platform uses a modified version of RBAC, by having also the groups with certain roles, rather than having only individual users.

Boomi accounts allow the administrator of the account to manage many different aspects of the users' permissions. For example, they can have access to different features and resources in the platform, and the administrators can also manage the sign-on methods the users can use to authenticate themselves. For example, SAML can be enabled to allow users to use single sign-on for the platform. The administrator can also manage the user password policies in the specific account.[57]

3.3.4 API Management

Boomi also includes its own API Management feature. The manager allows the deployment of API components and the exposing of versioned APIs. Boomi supports most authorization and authentication methods used in APIs. Either internal authentication, which requires only a username and password, can be used to check for authorization, or an external provider can be used, which requires the use of OAuth 2.0. Authorization is APIs on the platform is done for example in the Open API UI. There the Authorize section of the UI shows, which authorization methods are allowed for the API, and provides the ability to authenticate with them.[57]

The API management feature also allows for applying different policies for different environments for example development, testing, and production environments. And like many API managers, it also allows for the generation and assignment of client and environment-specific API keys.[57]

3.4 Comparison between authorization solutions

In this section, four different integration platforms' authorization solutions are going to be compared. The integration platforms covered are IBM App Connect, MuleSoft's AnyPoint Platform, Boomi Enterprise platform, and Frends Integration Platform. Frends is covered more in the upcoming Chapter 4, Chapter 5, and Chapter 6, but the other platforms are covered above.

3.4.1 Comparison of component authorization

Integration- and component-level authorization is possible in all of the platforms, but each implement it a little differently. One of the biggest similarities between them is that each platform has a form of audit logs or some kind of activity monitoring implemented. It can be used to track user actions as well as access to components and integrations. It seems that the platforms rely quite a lot on role-based authorization to achieve integration- and component-level authorization.

IBM App Connect offers the control of authorization on the component level by using centralized security policies and management. Administrators can control the access to each individual connector and specific actions within integration flows, but it does not seem to offer the same level of granular authorization management for each individual component as some of the others.

AnyPoint platform also has its own version of authorization for components and integrations. It is heavily focused on API-led development, so OAuth 2.0 plays a big role in the component and integration authorization. This component-level authorization can be achieved with the AnyPoint Platform Access Management. The platform also has a system called AnyPoint Exchange which gives control over who can publish and consume assets from the exchange. This improves component-level security greatly in the platform.

Boomi Enterprise Platform uses a lot of built-in connectors, which each can have their own authorization method implemented. It also supports OAuth 2.0 for secure authorization for integrations and APIs. It also uses component visibility management, to make sure components are only available for users who need them.

Frends Integration Platform also supports integration-level permissions. The authorization and permissions can be set for each individual integration process and API endpoint. APIs often rely on OAuth 2.0 for authorization, and authorization for processes can be configured for each separately and one process can use multiple authorization methods. Frends also supports component libraries, which can have their own permissions management, ensuring that only authorized users can modify or use them.

3.4.2 Comparison of permission management

There are quite a lot of similarities with the permission management for the different integration platforms, but there are also some differences. IBM App Connect, Boomi Enterprise platform and Frends Integration Platform all use roles as a way to manage the majority of the permissions of users. AnyPoint Platform does this a little differently, and rather than using roles, it uses teams to manage larger amounts of permissions more easily.

Even though not all of the platforms use roles, they all use a variation of Role-based Access Control as their authorization method. This allows all of the platforms to have fine-grained control over their users' permissions. In Boomi and Frends, the roles assigned to users are environment-specific, which makes it possible to have different roles for example for a developer in different clients' environments. In one client's environment, the developer might have a developer role, while in another they might have an administrator role.

AnyPoint Platform seems to do permissions and especially the permission structure a little differently. Rather than having a role for each user based on their needs in the specific environment, each user is added to teams which have certain permissions. These teams then can have child teams which have more specified permissions, while inheriting the permissions of the team above them. AnyPoint Platform Access Management also allows the administrators to have a centralized interface to manage all of the user and team permissions in the organization.

IBM App Connect is the only one of the platforms which doesn't force to use permission management. It uses administrator security to manage permissions, but by default, it is turned off, but it is highly recommended to use it. The permissions and roles can be configured separately for each environment.

3.4.3 Comparison of API management

There are more similarities than differences between the different platforms' API management features. Each platform supports the use of the most common authorization methods such as OAuth 2.0, API keys, and JWT. But some platforms have wider support than others. For example, IBM App Connect allows the use of LDAP, while the Frends Integration Platform allows custom authorization and authentication schemes.

As was the case in permission control, the API management feature of all of the platforms also supports RBAC. With this access control, users can be given permission to use the API management system, or they can be given access to for example only view the APIs, but not make changes to them. In many systems, it is often beneficial if not all clients have permission to manage APIs.

API key management is a key part of any API management system. IBM App Connect handles API keys a little differently than the other, it does not have an integrated way to generate API keys. But all of the platforms have a way to store and access the API keys securely, within the platform. In Boomi, AnyPoint and Frends, the API keys can be generated for each development, testing and production environment separately. The API management system gives the ability to give authorization to access certain integrations or APIs with the key. The keys can also be regenerated, rotated and revoked easily.

3.4.4 Summary of the differences and similarities

All of the platforms support some level of integration and component authorization but achieve it differently. IBM App Connect uses centralized security policies for component-level authorization. AnyPoint Platform focuses more on API-led development with significant reliance on OAuth 2.0, it also relies on access management for component-level authorization. Boomi Enterprise Platform employs built-in connectors with individual authorization methods, uses OAuth 2.0, and uses component visibility to achieve component-level authorization. Frends Integration Platform supports integration-level permissions with configurable authorization for each integration process and API endpoint, often using OAuth 2.0.

Permission management in the platforms share similarities but have distinct differences. IBM App Connect, Boomi Enterprise Platform, and Frends Integration Platform use roles to manage user permissions, using variations of RBAC. These roles can be environment-specific in each. AnyPoint Platform manages permissions using teams rather than roles. Users are added to teams with specific permissions, and the teams have child teams with inherited along with more specified permissions. IBM App Connect is unique in the way that it does not mandate permission management by default.

API management features across the different platforms have more similarities than differences. All of the platforms support OAuth 2.0, API keys, and JWT. All also use RBAC for API management. All platforms securely store and manage API keys. While IBM App Connect does not have its own API key generation, Boomi, AnyPoint, and Frends allow API key generation for different environments separately, and provide mechanisms also for key regeneration, rotation and revocation.

3.5 Privacy and security threats in authorization

There are a lot of different privacy and security threats that authorization methods try to prevent in integration platforms. Most of the different threats do not only appear on integration platforms, but they are also common in many different systems such as APIs. Authorization methods are inherently designed to prevent certain types of privacy and security threats, and not all authorization methods prevent all kinds of threats.

OWASP Top Ten[58] contain the most critical security risks to web applications, some of which also apply to integration platforms. OWASP's Top Ten API Security Risks[59] contain more specific threats for APIs, which are relevant in this situation. With the help of both OWASP lists and a few more resources, different threats have been identified, which are relevant to integration platforms and more specifically threats which can be handled with different authorization methods.

These threats can be divided into two different categories: Privacy Threats and Security Threats. Privacy threats include unauthorized access, data leakage, over-privileged users, and privacy policy violations. Security threats include cross-site request forgery, phishing and social engineering, and man-in-the-middle attacks.

3.5.1 Privacy Threats

Unauthorized access

Unauthorized access is one of the most obvious privacy threats that authorization in integration platforms is designed to prevent. If authorization and proper access control are not implemented properly into integration platforms, unauthorized users might gain access to resources they are not supposed to be able to access. Which can lead to data breaches.[59]

Data breaches are a major concern in integration platforms. Integrations usually deal with large amounts of data, which can also contain secure and sensitive information. So preventing data breaches is of great importance. There are many ways integration platforms can prevent or lower the effects of possible unauthorized access and data breaches, for example by using authorization methods like RBAC, which gives users access to only the necessary resources. This way, in the event of wrong parties gaining access to a user's credentials, not all of the resources are compromised. Platforms can also implement monitoring solutions to monitor data accessed by users. If suspicious activity is monitored, they can act on it quickly. To prevent credential theft, platforms should also implement systems like two-factor authentication, which might prevent threats like credential theft.[59][60][58]

Data Leakage

While integrations handle large amounts of resources there are many ways data leakage can happen. Integrations should keep the data being handled encrypted whenever possible, and when not possible, only authorized users should be able to view the data. If data is being shown for example in logs or error messages, unauthorized users might be able to gain access to it.[61]

If proper authorization methods and encrypting are not being used by the systems in which the data being handled is transferred, there is a possibility of it being intercepted. Even if the integration platform has good authorization mechanisms, it is possible that the receiving end does not.[61]

Over-Privileged Users

If integration platforms do not use proper access control methods, there is a possibility of having over-privileged users. This can happen if access control is done poorly or if it is managed poorly. For example, DAC has a good possibility of having over-privileged users, but with RBAC or MAC, this is much more unlikely. The risk from over-privileged users comes from the potential of compromised credentials, in which case the perpetrators could gain access to a much more significant part of the resources.[60]

Privacy Policy Violations

Nowadays it is important for all systems, like integration platforms, to comply with all of the different privacy policy regulations, such as GDPR in the EU, and HIPAA in the USA. Violations of these regulations could lead to legal penalties and possible loss of trust from the users. Proper authorization and authentication are integral to complying with their policies. For example with authorization the integration platform can make sure that only the right users can access sensitive data.[62][63]

3.5.2 Security Threats

Cross-Site Request Forgery

Cross-Site Request Forgery or CSRF is an exploit, that malicious users could exploit to perform unauthorized actions in the integration platforms. They could for example modify the permissions and data of a user, or start integrations. There are many ways to prevent CSRF, for example, PKCE can be implemented into OAuth 2.0 authorization code flow to prevent such attacks. Also, confirmation of users' credentials while performing certain actions can prevent this exploit.[40]

Phishing and Social Engineering

There are many ways phishing and social engineering attacks can pose a security threat to an integration platform. Usually, the goal of these kinds of attacks is to gain access to user credentials. Without proper precautions, people with malicious intent could access the resources on the platform.

Credential theft can also happen in the form of API key hijacking or theft. API keys usually have authorization attached to them, so the wrong party getting access to an API key can cause data breaches and unauthorized access. In integration platforms, API keys can often easily be changed and regenerated, so the effects of API key leaking can be lessened with proper API key management and monitoring.

To lessen the effects of credential theft, proper authorization and authentication can be added. For example, adding two-factor authentication upon login makes credential theft much more unlikely. Proper authorization monitoring can also help to identify compromised users and revoke their permissions before too much damage is caused. Using OAuth 2.0 reduces the risk significantly also because credentials do not have to be shared between different systems, so they can be kept more secure.[58]

Man-in-the-Middle Attacks

Man-in-the-middle attacks are data interception attacks, where the attackers interrupt the communication between two systems and steal or alter the transmitted data. There are many ways to prevent these kinds of attacks, for example by limiting the permissions for transmitting data, and when data needs to be transmitted it should be end-to-end encrypted. This encryption can be done with for example TLS. Using API keys can also help mitigate these attacks. The keys need to be rotated regularly and there needs to be a secure way to manage the keys.

Third-party Apps

Integration platforms often use third-party apps and systems for many different purposes. For example, many integrations utilize different APIs, which need to be trusted and secure. Authorization and authentication are important when dealing with APIs and other third-party systems, especially when these apps can connect to the integration. Like any other client, APIs and third-party apps need to have permissions for example to access integrations or resources in the platform. Many different authorization models can be used to give and handle these permissions, for example, API keys and OAuth 2.0. If the authorization is done correctly and the third-party application is trustworthy and secure, there is little security risk for the integration platform.[60]

4 Frends Integration Platform

This chapter gives background information about the Frends Integration Platform. It focuses more on the information relevant to authorization and the thesis in general, so it is not meant to be a comprehensive document of the whole architecture of Frends. The chapter also includes a section about integration development in Frends, and how authorization is done within processes.

Frends Integration Platform is a product owned by Frends Technology Oy. Frends Integration Platform or Frends was originally developed for connecting gas stations in Finland and exchanging data, mainly gas price information, between the stations and a central IBM Mainframe. The Frends name actually comes from this original use case: Front End Dialing System or Frends.[64]

4.1 Architecture

Frends is a low-code hybrid integration platform. It focuses on flexibility, with a clean DevOps experience. Being a hybrid integration platform, Frends enables the use of services and systems located both in the cloud, as well as on-premise data centers. The integration flows between the different services and systems, are made using a BPMN-based visual GUI. In the background, Frends is written in C#.[64]

4.2 Agent

In Frends *Agents* refer to the remote integration runtimes, which execute the integration code generated by the platform. They can be deployed to any on-cloud or on-premise infrastructure. All of the Agents have three distinct architectural layers, which give the agents the ability to provide API Gateway functionality, expose APIs, and run integrations. These layers are:

- **Control Layer**, which uses Azure Service Bus queues to communicate with the central management portal. These queues are used for example when integration flows are deployed, agent configuration is modified, or integration flows are started manually.
- **Trigger Layer**, which is used for example when starting a scheduled integration or exposing an API Endpoint. Its main function is to trigger integration flows themselves.
- **Execution Layer** works after the trigger layer has triggered an integration flow. It handles the actual execution of the integration flows.

When multiple agents share the same configuration, it is called an agent group. This is used for example for a high-availability installation. When a *Process* is deployed to any of the agents in the agent group, the same version deploys to all of the agents in the group. Agent groups can contain one or more agents.

4.3 Environment

Environments are sort of containers for grouping agents used in the same role. Usually, there are three different environments, which are the Development environment, the Testing environment, and the Production environment. Environments are also used often in authorization, by limiting users' access to certain environments. For example, developers can be denied access to the production environment, or denied permission to create and manage APIs within the platform. Environments also contain other security settings, for example, API keys, which can be used in different APIs and processes. Usually, they are environment-specific and require specific permissions to access.

Environment Variables

Environment variables are optional static environment-specific variables within the platform. They are defined in the Frends Control Panel and are stored securely in the database. They are used to store static information related to specific processes, for example, usernames and passwords and other connection information for systems, such as connection strings to a database. The variables being environment-specific means that for example different connection addresses can be used for the testing environment than the production environment, or an archival task can be turned off for the testing environment.

4.4 Process

In Frends, actual integrations are built-in Processes. There are two different kinds of processes, main processes and *Subprocesses*. Regular main processes are used to create the actual functionality for the integration flow in question. There are a few mandatory elements which every process has to contain for it to function, which are called a trigger and a return, so essentially a start and an end. There are a lot of different elements processes can include, such as Tasks, Code sections, Scopes, and Decisions. All of these different elements have to be connected to each other with a connect tool. In the background the processes actually compile into C#code. A

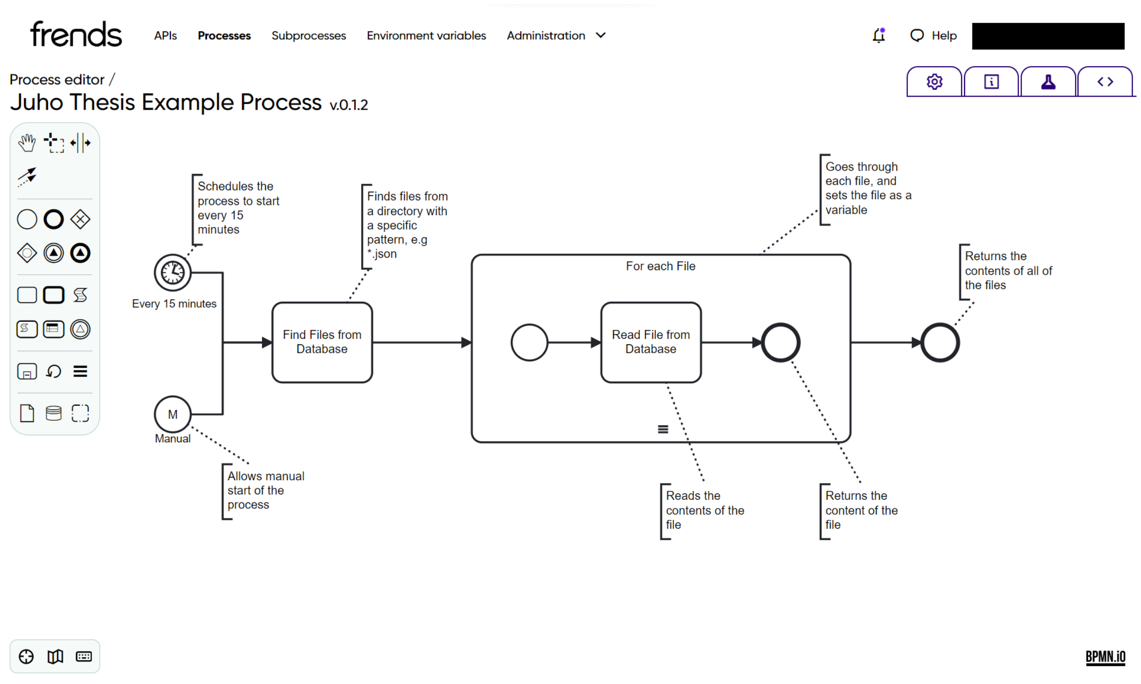
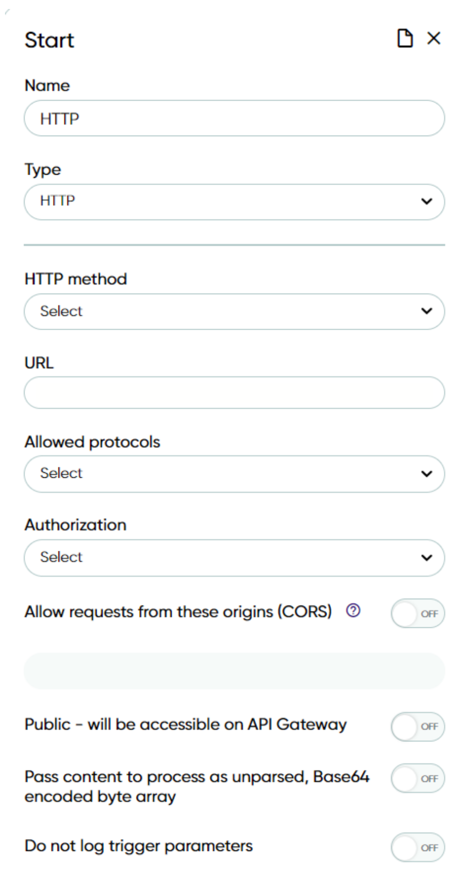

frends
APIs
Processes
Subprocesses
Environment variables
Administration
Help
[redacted]
Process editor /
Juho Thesis Example Process v.0.1.2
Schedules the process to start every 15 minutes
Every 15 minutes
Finds files from a directory with a specific pattern, e.g *.json
Find Files from Database
For each File
Goes through each file, and sets the file as a variable
Read File from Database
Returns the contents of all of the files
M
Manual
Allows manual start of the process
Reads the contents of the file
Returns the content of the file
BPMN.io

Figure 4.1: An example of a simple scheduled file reading Process in Frends

process also works as a visual documentation of what the integration actually does, as shown in Figure 4.1. Authorization in processes is usually done in the Triggers, but different kinds of authorization can also be implemented through the use of tasks. [64]

4.4.1 Subprocess and Remote Subprocess

Subprocesses can be used to house smaller parts or functionalities of processes, which can be called by other processes. Subprocesses are also able to call other subprocesses, which enables a process hierarchy housed inside a regular process. Subprocesses are usually created for functionalities which are usable as is by other processes as well. Subprocesses can also be located in a different agent group compared to the main process, in which case they need to be called remotely. This is useful in cases where certain resources are only available in certain environments. [64]


Start

Name: HTTP

Type: HTTP

HTTP method: Select

URL:

Allowed protocols: Select

Authorization: Select

Allow requests from these origins (CORS): OFF

Public - will be accessible on API Gateway: OFF

Pass content to process as unparsed, Base64 encoded byte array: OFF

Do not log trigger parameters: OFF

Figure 4.2: HTTP trigger option in Frends 5.6

4.4.2 Triggers

Triggers are an essential component of any process in Frends. Triggers are used to start the process. There are multiple different types of Triggers available in Frends 5.6: Manual, File, Schedule, Conditional, HTTP, API, Queue, Service Bus, and RabbitMQ. Each has different configurable options, as shown for the HTTP trigger in Figure 4.2. Any process can contain multiple different triggers, such as a schedule trigger and an HTTP trigger.

Authorization in triggers

The authorization in triggers checks that the system or the user triggering the process has the right permissions to do so. For example, when using an API trigger, it checks if the API key given by the caller matches with the one specified in the trigger. If authorization fails an error will be thrown and the process won't start.

For most triggers authorization is not necessary, because they either can not be started from outside of the environment, such as a manual trigger, or they start automatically based on a specific rule, such as the schedule and the conditional trigger. Some triggers have the option for authentication, but the authentication is for the system the trigger is connecting to, such as the file system, which the file trigger reads might need authentication, or the service bus, queue or RabbitMq might need to authenticate to read the queue they are connecting to.

The only triggers, which have their own authorization method options are HTTP trigger and API trigger. They can enable authorization, in which case the caller needs to have proper authorization to start the process. HTTP trigger can use basic authorization, API key or certificate authorization to authenticate the caller and to check for the permissions. API trigger is connected to an API deployed in the platform and uses the same authorization that is set up for the API. This can be for example any of the OAuth 2.0 flows or an API key.[64]

4.4.3 Tasks

Tasks are the sort of elements or building blocks used to create Processes. They enable the individual functionalities of Frends. Tasks work sort of like microservices. For example, a task can be used to read a file from a database, make an HTTP request, or make an SQL query. Often they can be customized by using different parameters and various connectors. For example, a file write task might have the option to overwrite a file if a file with the same name already exists. There are a lot of ready-made tasks available from Frends, which cover most of the common functionalities. But in case the common tasks are not enough, there are also community-made tasks, which cover more specific functionalities. It is also possible for developers to make their own custom tasks, which can possibly later on be added to the list of community-made tasks.[64]

4.5 Integration development in Frends

4.5.1 Creating integrations

Integration development in the Frends Integration Platform is done in the web browser. The platform supports all of the major web browsers in the market. A new process or modifications to an existing process can only be saved after the process passes validation. The validation checks that there are not any syntax-related problems within the process. Whenever a modification to a process is saved, the patch number of the process increases automatically. If the developer wants to increase the minor or major versions of, they can do it manually in the process settings. There is also the possibility to save drafts in between actual patches. Drafts are saved, but they are not validated so they can not be deployed.

One of the main things, which needs to be taken into consideration when creating a new process, is who is going to use the process. This can determine what kind of authorization, if any, needs to be implemented, and if multiple different types of authorization need to be used. Also, the systems the process might need to connect to need to be identified. Sometimes third-party systems or APIs need to be connected with specific authorization and authentication methods in use.

4.5.2 Deploying

Usually, there are three different environments where integration processes can be deployed, development, testing and production. After modifications have been made to a process, and the modifications are saved, the platforms run the validation and if successful the process can be deployed. If the validation fails, the process can not be saved, and therefore can not be deployed. This protects the agents from running code with syntax-related problems.

When deploying a new version of a process to an agent, it updates the same version to all of the agents in the same agent group automatically. A certain specified amount of older versions of processes are stored and if needed they can also be deployed to an agent replacing the version already on the agent. To see which version of a process is on which agent, the change log can be used. It shows the agents the process is deployed to, along with showing when, who, and what modifications have been made to the process in which version. It also gives the ability to see the difference between the current and another version of the process.

4.5.3 Authorization inside Processes

There are many ways of doing authorization inside Frends processes, because of the flexibility of the platform, it allows the developer to do essentially anything within it. The authorization can be done in the trigger, or it can be done separately inside the process. Some triggers can use authentication and authorization methods like API keys or OAuth 2.0 flows, in which case if the authorization passes, the process starts. Inside the process authorization can be done by for example checking if the call came from a whitelisted IP address, or if a specific parameter was provided. When creating a new process in a Frends environment, everyone with viewing permissions to the environment gains access to the specific process, and everyone with editing access can make modifications to it.

If the Process calls other APIs which need authorization, it can also be done inside the process in several different ways. The process can for example call a token API to get an access token, which can then be used to call the actual API. The process can also for example forward the authorization information from the trigger or from a file to another system. Also often for authentication and authorization, environment variables are used to store usernames, passwords, API keys, and other static variables.

4.5.4 Testing

There are a few ways to test processes within Frends. For testing individual tasks in processes, Frends has support for unit testing, but this feature is often disabled for testing and production environments, and the development environment often does not have the same permissions to access resources as testing or production. To test complete integration flows, integrations are usually deployed to the testing environment. There end-to-end testing can be completed while checking that each task works correctly and for example, data transformations and data flow between systems work as expected.

5 Authorization Privacy and Security in Frends

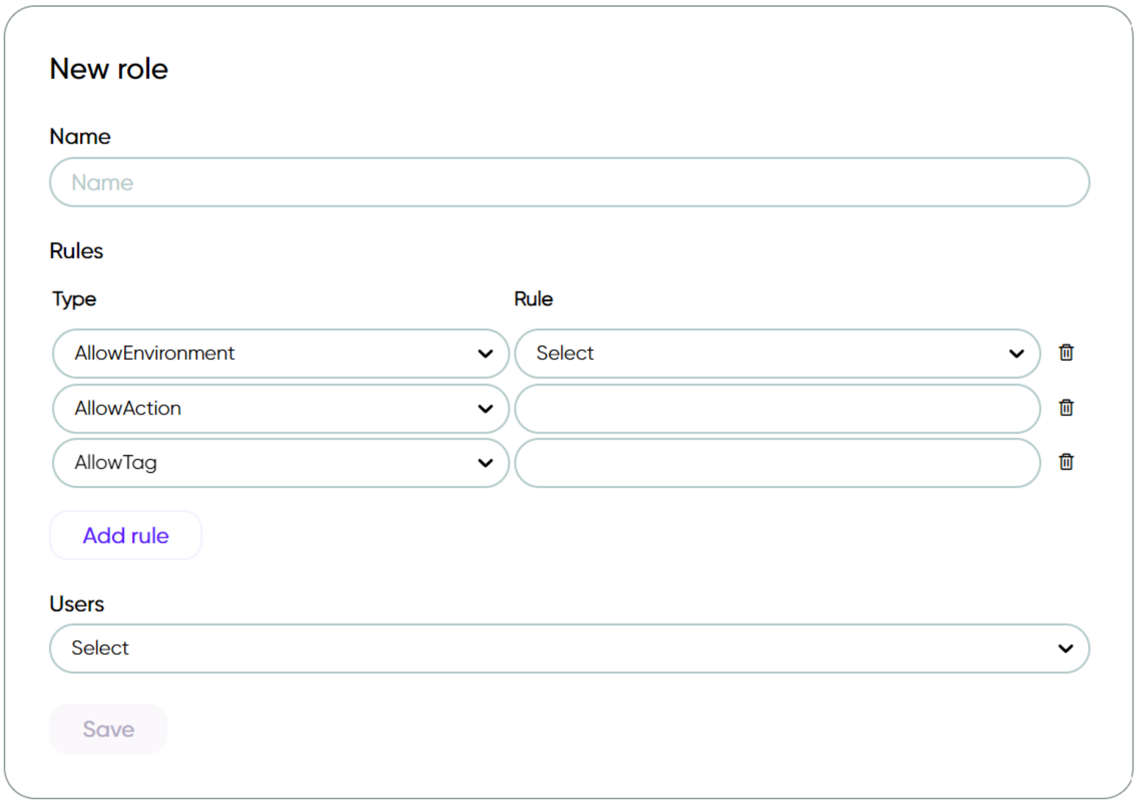
This chapter focuses on how Frends manages the different privacy and security threats authorization is designed to handle in APIs and integration platforms. These threats are introduced in the Chapter 3. The threats addressed in this chapter are: unauthorized access, data leakage, over-privileged users, cross-site request forgery, phishing and social engineering, man-in-the-middle attacks, and third-party apps. Integration platforms and APIs face a lot more threats than covered in this thesis, so the list is not meant to be comprehensive, but it contains some of the most common and biggest threats. Through the list, this chapter aims to answer the RQ3.

Frends Integration Platform does comprehensive privacy and security testing, so this chapter covers only a small part of what Frends does to prevent threats. The threats have been divided into two categories, privacy and security threats, but many of the threats have aspects relating to both.

5.1 Privacy threats

5.1.1 Unauthorized attacks

There are a few ways which help Frends prevent unauthorized access to resources. The most important way is the use of RBAC. Frends has its own User Manage-


New role

Name

Name

Rules

Type	Rule
AllowEnvironment	Select
AllowAction	
AllowTag	

Figure 5.1: Creating a new role in Frends

ment window, which is accessible to administrators of the environment. This window allows the management of different roles, management of individual users, and OpenID authentication applications. Roles can have many different rules associated with them, for example allowing or denying access to certain environments or allowing or denying an action. New roles can be easily set up as shown in Figure 5.1, and users can have multiple different roles. Role and user management help with making sure users have only the permissions required for their work.

For APIs, Frends also has the API Access Policies, API keys and API rulesets, which all keep unauthorized systems from calling the API. API Access Policies are often used with OAuth 2.0 authorization flows. It enables to choose which token issuer is allowed, and it allows the setting of different rules to deny or allow requests based on whether the defined claims match in the access token given by the caller.

5.1.2 Data Leakage

To prevent data leakage, Frends employs a variety of methods, for example, logging for certain tasks that could leak data can be turned off, and access to the production environment can be restricted with RBAC. Data stored in the platform is also encrypted with industry-standard encryption algorithms, making it readable only from the Frends environment. Frends also has continuous monitoring within the platform for signs of data breaches or unusual activities.

5.1.3 Over-Privileged users

In Frends the threat of over-privileged users can also be managed from the User Management window. It requires the administrators to frequently check the different roles in the environment and make sure users have the roles they need. With roles specific actions can be denied from users, which can help in managing over-privileged users.

5.2 Security threats

5.2.1 Cross-Site Request Forgery

Cross-Site Request Forgery is quite simple to prevent in Frends. Relevant requests can have CSRF tokens added to them, which are tied to the specific session and have to be validated before the requested action is executed, for example using PKCE with authorization code flow. When handling cookies in certain requests in Frends, *SameSite* attribute can be added, which ensures that cookies are not sent along with cross-site requests. For incoming requests, checking *Referer* and *Origin* headers can help in verifying that the requests are coming from the correct source.

5.2.2 Phishing and Social Engineering

Frends handles the effects of phishing and social engineering in the platform well. Frends environments are set up to handle authentication through OpenID. Multiple different applications through which the authentication process goes through can be set up for any Frends environment. Many OpenID authentication apps also require two-factor authentication to give additional security and prevent unauthorized users from getting access to the environment by only gaining access to an authorized user's login credentials.

5.2.3 Man-in-the-Middle attacks

There are many ways Frends can prevent Man-in-the-Middle attacks. For example, Frends also uses Transport Layer Security or TLS, and Secure Sockets Layers or SSL protocols to encrypt data during transmission. Frends also includes the possibility to encrypt the data even further if needed. Frends also prioritizes the use of secure communication protocols such as HTTPS over the use of HTTP.

5.2.4 Third-party Apps

Frends Integration Platform can implement a lot of third-party apps through the use of tasks, which can also be made by the community or custom-made. Native tasks in Frends are tested to be secure, and only authorized users can make changes to them. Community-made tasks can be made by anyone, but they still are monitored so they do not pose a threat to the platform. Also only authorized users can manage and import tasks in an environment.

6 OAuth 2.0 workflows in Frends

This chapter introduces the most common OAuth 2.0 authorization flows, which are relevant to the Frends Integration Platform, and how they can be implemented into an API published in Frends. Currently, in Frends only Implicit flow has documentation for how it can be implemented, but there is not much knowledge on how client credential flow and authorization code flow can be implemented with Frends. The goal of this chapter is to find out and test how implicit flow, client credential flow and authorization flow can be used for authorization in an API published in Frends. After that, based on the solutions, a list of best practices is created to further help with OAuth 2.0 authorization in Frends. At the end, there are improvement suggestions for the Frends Integration Platform, which could help with OAuth 2.0 flows. With the implementation solutions and the improvement suggestions, this chapter aims to answer the RQ4.

For the different OAuth 2.0 flows the basis for the configuration in the identity and access manager side is based on the Microsoft Identity Platform documentation[29]. And the specific configurations that make the identity manager work well with Frends and the set up on the Frends environment side are based on own experimentation and testing.

6.1 Client credentials flow

6.1.1 Core concept

Client credential flow is an OAuth 2.0 authorization flow. It is very useful especially when dealing with system-to-system authorization rather than user authorization, because it does not require a user to log in. Client credential flow works by exchanging the system credentials for an access token in an identity and access manager, so it does not have a separate authorization phase like authorization code flow. In Frends there is great demand for an implementation solution for this flow.[33]

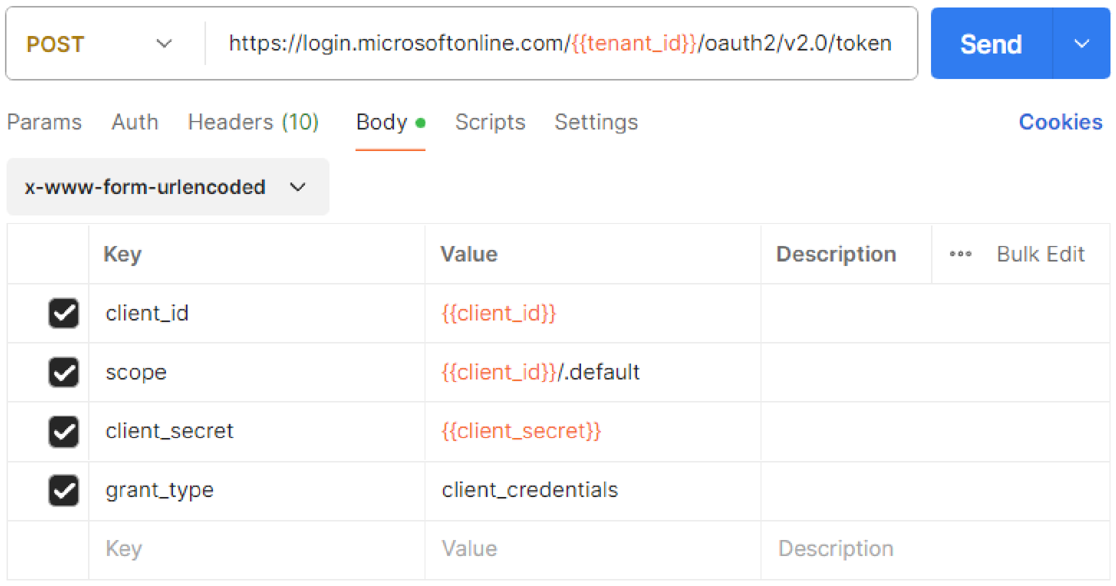
6.1.2 Implementation solution for Frends

Setting up Identity management

Currently, there is no documented way to set up a client credential flow for a Frends API, but in reality, it is quite simple. Like all of the different flows, first, an external identity and access manager needs to be used. In this example setup Microsoft Entra ID is used, because it is one of the most used identity managers in Europe, and it is used often with Frends Integration Platform.

First, an application needs to be set up in Entra ID, and it needs to be accessible at least by users in the same organization, but the *multiple organizations* -option can be used also, depending on who is going to be using the flow. Then a client secret needs to be generated. Entra ID supports the creation of multiple secrets, so different secrets can be created for different clients. For most applications, it is also recommended to set up different scopes for the application, which can be used as scopes later.

Then the access token can be retrieved with a simple HTTP POST request to the */oauth2/v2.0/token* address. As shown in Figure 6.1 The request needs to contain the client ID and client secret from the Entra ID application, as well as a scope



```
https://login.microsoftonline.com/{{tenant_id}}/oauth2/v2.0/token
```

	Key	Value	Description
✓	client_id	{{client_id}}	
✓	scope	{{client_id}}/.default	
✓	client_secret	{{client_secret}}	
✓	grant_type	client_credentials	
	Key	Value	Description

Figure 6.1: Client credential access token request in Postman

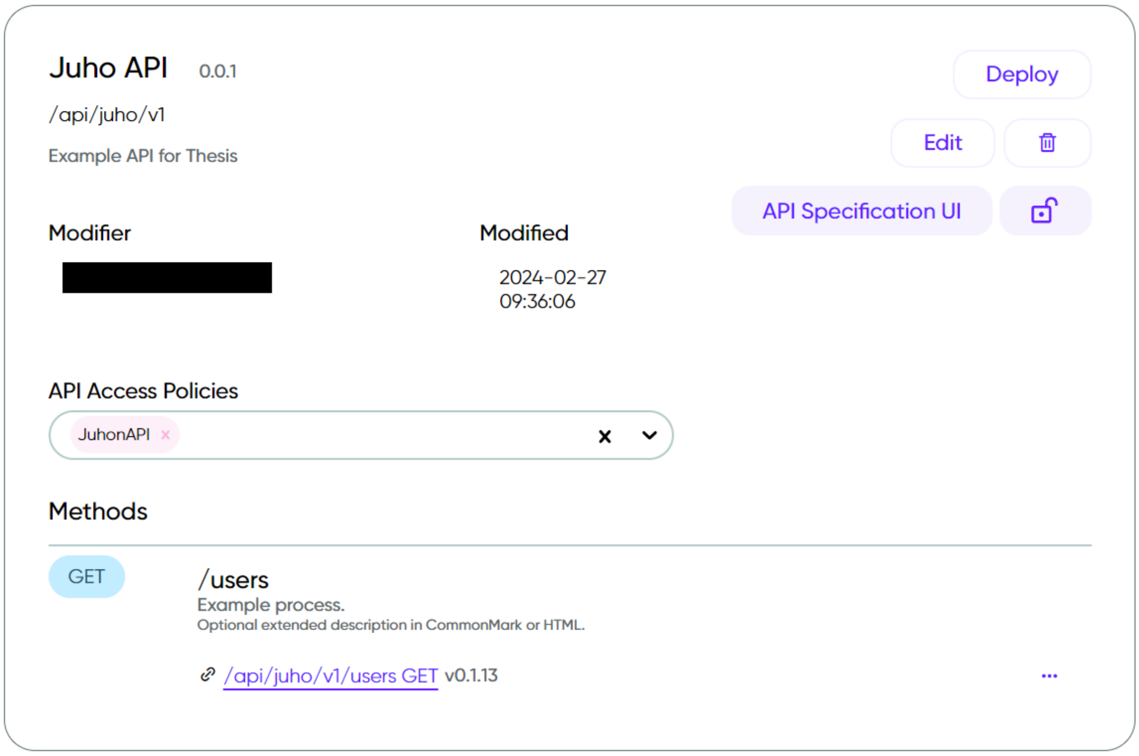
which the request is for. The access token received is a JSON Web Token or JWT, which contains information about the request and the requester, which can be seen when decoding the JWT.

Creating a Frends API with OAuth 2.0 authorization

After getting the access token, setting up any of the different OAuth 2.0 flows works basically the same. From the access token, values for the audience and issuer can be found. These are then used to set up a new OAuth application in Frends with the *OAuth Application setup* tool. In the advanced settings of the tool name, role and scope claim types can be assigned, if needed for the application. The tool also allows the use of Signing certificate details, if automatically fetched singing details can not be used, but for simpler use cases this is not necessary.

After setting up the new OAuth Application, a new API Access Policy needs to be created. There the issuer from the previously created OAuth Application needs to be chosen from the list of available issuers. After the set-up of the API Access Policy is done, the authorization can be added to an API as shown in Figure 6.2.

The API authorization then works by adding the access token to a bearer token in the requests Authorization header.



Juho API 0.0.1

/api/juho/v1

Example API for Thesis

Modifier: [redacted]

Modified: 2024-02-27 09:36:06

API Access Policies

JuhonAPI ×

Methods

GET /users

Example process.

Optional extended description in CommonMark or HTML.

/api/juho/v1/users GET v0.1.13

Figure 6.2: Frends API using OAuth 2.0 flows

6.2 Authorization code flow

6.2.1 Core concept

Authorization code flow is an OAuth 2.0 flow, which is used when authorizing and authenticating users. It works by exchanging an authorization token for an access token, so there is a separate authentication step before getting the access token. In Frends Integration Platform it would be used usually instead of Implicit flow because it is much more secure. This flow also supports PKCE, which is recommended to be used always.[32]

6.2.2 Implementation solution for Frends

The authorization code flow is a bit more complicated to use than the other flows, but for authorization for single users, it is the best method. Setting up an authorization code flow also requires the use of an identity manager like Entra ID. There the major difference in setup for client credentials flow is the use of a redirect URL. This redirect is often an OpenID authentication application, which can be for example provided by the Frends integration platform. All Frends platforms should provide a redirect URL. Otherwise, the setup is the same, so the client secret needs to be generated and tenant ID and client ID are needed from the identity provider application.

After the Entra ID application is properly set up, an authorization token needs to be retrieved. This can be done by calling the */oauth2/v2.0/authorize* address of the identity provider with these query parameters:

- **client id**: can be retrieved from the Entra ID application
- **response type**: code
- **redirect url**: must be the same as in the Entra ID application
- **response mode**: query
- **scope**: a list of scopes that the user consents to
- **state**: an identification number for the specific request
- **prompt**: login

After using this request in a browser and logging in, the authorization token can be retrieved from the response URL. This token is then used to retrieve the access token the same way client credential flow is used, except by adding the redirect URI and code parameters. This access token contains more identifying information than client credential flow, for example, the username of the authenticated user. Setting up the flow for Frends is then the same process as for every other flow.

6.2.3 PKCE implementation

When using authorization code flow it is also recommended to use PKCE with it. It can be used by generating a SHA 256 code verifier and a code challenge as shown. These codes should be unique for each authorization request, so using a tool like Postman to generate them is recommended. To add them to an authorization request the code challenge and the code challenge method, like S256, need to be added as parameters for the authorization token request and the code verifier needs to be added to the access token request. If the code challenge and verifier do not match the request will be denied.

6.3 Implicit flow

6.3.1 Core concepts

Implicit flow is still one of the most common authorization methods used in Frends and it is also the easiest to implement and use out of the different OAuth 2.0 flows. It works by returning the access token immediately upon authorization request without the need for an extra authorization code request. According to the OAuth 2.0 specification, implicit flow is not recommended to be used anymore in most systems, because it returns the access token directly with an http redirect, which is not secure. So it is recommended to replace Implicit flow implementations with Authorization code flow. Implicit flow also does not have support for PKCE, which also makes it a less secure option compared to authorization code flow.[30] Implicit flow is still in use in some APIs deployed in Frends, where the client system can be fully trusted, but it is recommended to switch them to use authorization code flow.

6.3.2 Implementation solution for Frends

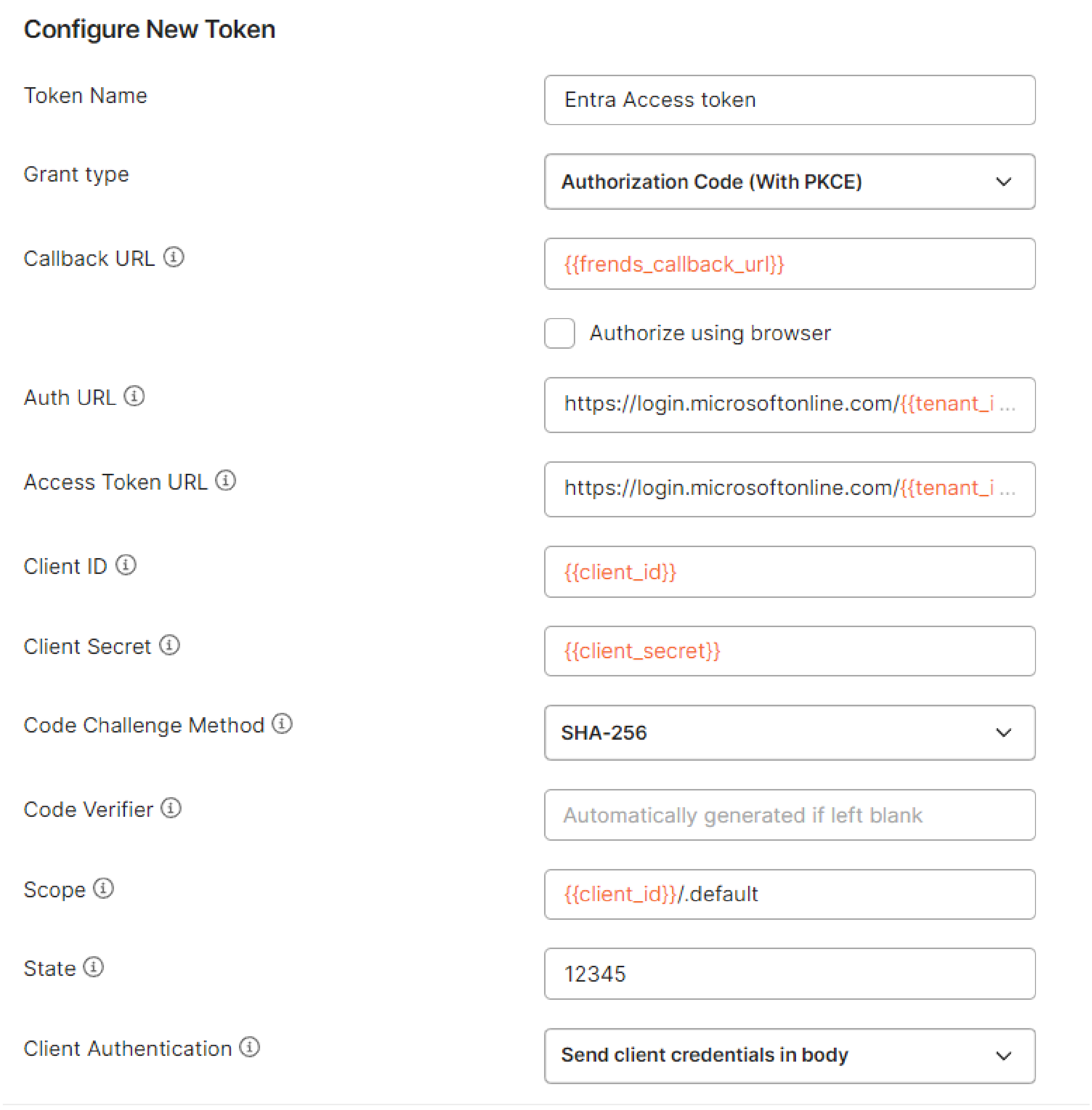
Implicit flow is the only OAuth 2.0 flow, which has proper documentation for Frends. The documentation is a little outdated, for example, it talks about Azure AD, which nowadays is Entra ID, but the functionality stays the same, and following the documentation results in a working Implicit flow implementation.

Like every other OAuth 2.0 flow, implicit flow needs to use an external identity and access manager solution, and Entra ID is chosen for this thesis. To use the service for OAuth 2.0 authorization, an application needs to be created in the service. The application is set up much the same way as the authorization code, except an additional option needs to be configured for the application to return an ID token instead of an authorization code. If both are enabled, it is called a hybrid flow, and both authorization flows can be used for the same application.

To get the access token for implicit flow, it is done the same way as getting the authorization code for the authorization code flow. The only difference is that the response type needs to be *id token*, and the response mode should be *fragment* for the hybrid flow to work. This request prompts the user to log in and instead of returning the authorization code, it returns the access token without needing to do a separate token request.

6.4 Best practices for OAuth 2.0 Flows in Frends

Often when testing APIs in Frends, the Postman API platform[65] is used to call the different HTTP requests required. For authorization requests, Postman has a simple built-in authorization management solution, which simplifies the token retrieving greatly as shown in Figure 6.3. It offers the ability to use any of the OAuth 2.0 flows used in this Thesis and even more. Authorization code flow with PKCE benefits the most from this, because all of the requests can be automated easily, and it offers the ability to log in directly.



Configure New Token

Token Name	Entra Access token
Grant type	Authorization Code (With PKCE)
Callback URL	{{frends_callback_url}}
	☐ Authorize using browser
Auth URL	https://login.microsoftonline.com/{{tenant_i ...
Access Token URL	https://login.microsoftonline.com/{{tenant_i ...
Client ID	{{client_id}}
Client Secret	{{client_secret}}
Code Challenge Method	SHA-256
Code Verifier	Automatically generated if left blank
Scope	{{client_id}}/.default
State	12345
Client Authentication	Send client credentials in body

Figure 6.3: New access token configuration in Postman using the Authorization Code flow with PKCE

When setting up APIs in Frends, proper OAuth 2.0 flows should be used according to the purpose of the API. Often it is best to use Authorization Code flow with PKCE if the API is being used by users. In this case, the users calling the API need to be given proper authorization in the Frends environment. Authorization code flow should not be used without PKCE, because it adds an additional layer of security without increasing complexity too much. It is recommended to reconfigure all systems still using implicit flow to use authorization code flow. If the API is designed for system-to-system communication, Client credential flow is recommended. It does not require callers to authenticate themselves, rather it uses the client secret for authentication, which can easily be revoked if needed.

Frends has its own ways of storing secrets, authorization credentials, and refresh tokens securely, but the caller of the API needs to have a secure way to store them too. Because refresh tokens can be used to get a new access token without needing to authenticate again, it is essential to keep them secure. Client secrets as well as any other secrets are inherently to be kept secure. Even though a single Entra ID application can be used for multiple clients and APIs, there should be a separate client secret for each, to be able to revoke access for certain clients easily.

6.5 Authorization flow improvement suggestions for Frends

If a single Entra ID application is used for different OAuth 2.0 flows, in Frends there is currently no way to natively limit which OAuth 2.0 flows can be used to get the access token for a specific API or process. It can be limited inside the process by limiting which attributes and values the access token needs to contain for the process to start. It would be beneficial to have a way to limit it without needing to do it in the process, for example through the OpenAPI configuration.

In the current version of Frends, when authorizing through the swagger UI implicit and authorization code flow works well, but client credentials flow results in an error and a failed authorization. This is not a significant problem since multiple different flows can be used for a single API, but having a working client credentials flow in Swagger UI would help the testing of APIs deployed in Frends.

7 Discussion

7.1 Main Contributions

The main contribution of this thesis is the research on how different OAuth 2.0 authorization flows can be implemented into APIs deployed in Frends and how the Frends Integration Platform works with OAuth 2.0. Before the research, only Implicit flow was widely in use and only a few somewhat working solutions had been done for client credential flow. However no guide or documentation to getting a working client credential or authorization code flow had been made, even though getting a comprehensive guide has been in high demand recently. The improvement suggestions for the authorization flows will also be forwarded to the development team working on the Frends Integration Platform.

7.2 Practical Applications

The research on OAuth 2.0 authorization flows in Frends will be used for making new authorization flow documentation for the Frends Integration Platform. The documentation was not completed for the thesis, due to Frends Integration Platform getting a new release soon version 5.8, which will change the API Access Policy window completely. The documentation will be done for the new version, once the new version is released, and the changes to the implementations are done.

7.3 Limitations

One of the biggest limitations with this thesis was the lack of peer-reviewed scientific papers on the topics covered. There are a lot of papers about authorization, but most of them do not cover the topics relevant to APIs and integration platforms. Also, not a lot of outside research has been done on how the different integration platforms handle authorization, so most of the research had to be done based on their own public documentation.

8 Conclusion

8.1 Summary of Findings

This thesis had four different research questions. RQ1 and RQ2 focus more on integration platforms in general, while RQ3 and RQ4 focus on Frends Integration Platforms and its authorization for APIs. The research questions for the thesis were:

- **RQ1**: What are the biggest differences in authorization in different integration platforms?
- **RQ2**: What are the typical privacy and security threats that authorization mechanisms are designed to prevent in integration platforms?
- **RQ3**: How does Frends Integration Platform prevent the typical privacy and security threats related to authorization and APIs?
- **RQ4**: How different OAuth 2.0 authorization flows can currently be implemented in the Frends Integration Platform and could they be improved somehow?

RQ1 was answered in **Chapter 3**. There authorization in IBM App Connect, MuleSoft's AnyPoint Platform, Boomi Enterprise platform, and Frends Integration Platform were compared. The comparison was divided into the comparison of integration and component authorization, permission management and API management. It was found that the different integration platforms have a lot of similarities in authorization, but they achieve the result with various different implementations. There were also some unique differences in each integration platform, which were not present in the others.

Also in **Chapter 3 RQ2** was answered. OWASP Top Ten security risks for APIs[59] and various other resources were used to identify the most typical privacy and security threats that authorization handles in APIs and integration platforms. The threats identified were authorized access, data leakage, over-privileged users, privacy policy violations, cross-site request forgery, phishing and social engineering, and man-in-the-middle attacks.

Chapter 5 addressed the **RQ3**. There some of the threats introduced for **RQ2** were evaluated in the Frends Integration Platform. It was found that Frends has a way to prevent all of the threats introduced in various different ways. Frends also can prevent many other privacy and security threats that were not covered in this thesis.

RQ4 was covered in **Chapter 6**. There the methods for how to implement client credential flow, authorization code flow, and implicit flow into an API deployed in Frends were explained. Also, a few improvement suggestions for the authorization flows in Frends were found.

8.2 Future Research

The world of integration platforms and APIs evolves constantly, so it is important that the authorization methods being used evolve with it. Ongoing research and development of authorization methods and protection against emerging privacy and security threats is essential to ensure the secure and efficient operation of interconnected systems.

8.3 Final Thoughts

In conclusion, this thesis has shed light on the nuanced field of authorization methods in integration platforms and APIs, with a particular focus on the Frends Integration Platform. By addressing the research questions, the thesis has contributed to a better understanding of privacy and security measures, and authorization flows that are important for protecting resources in different systems. The knowledge gained from this research highlights the importance of robust authorization mechanisms in integration platforms and APIs deployed in them. As integration platforms continue to evolve, so will authorization with it.

References

[1] OAuth, *OAuth 2.0 — OAuth*. [Online]. Available: https://oauth.net/2/ (visited on 02/02/2024).

[2] Oasis. "FrontPage - SAML wiki". (), [Online]. Available: https://wiki.oasis-open.org/security/FrontPage (visited on 06/19/2024).

[3] Swagger. "API keys". (), [Online]. Available: https://swagger.io/docs/specification/authentication/api-keys/ (visited on 02/25/2024).

[4] A. Jøsang, "A Consistent Definition of Authorization", en, in *Security and Trust Management*, G. Livraga and C. Mitchell, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2017, pp. 134–144, ISBN: 978-3-319-68063-7. DOI: 10.1007/978-3-319-68063-7_9.

[5] NetSuite, *What Is Authorization? Definition & Examples*, en, Apr. 2022. [Online]. Available: https://www.netsuite.com/portal/resource/articles/erp/authorization.shtml (visited on 02/02/2024).

[6] Microsoft. "What is authentication? definition and methods | microsoft security". (), [Online]. Available: https://www.microsoft.com/en-us/security/business/security-101/what-is-authentication (visited on 02/25/2024).

[7] Auth0. "Authentication vs. authorization", Auth0 Docs. (), [Online]. Available: https://auth0.com/docs/ (visited on 02/25/2024).

[8] J. Saltzer and M. Schroeder, ”The protection of information in computer systems”, *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, Sep. 1975, Conference Name: Proceedings of the IEEE, ISSN: 1558-2256. DOI: 10.1109/PROC.1975.9939. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1451869 (visited on 02/16/2024).

[9] N. C. S. Center, *Guide to Understanding Discretionary Access Control in Trusted Systems*, en. DIANE Publishing, Sep. 1995, Google-Books-ID: frz12ta9rQgC, ISBN: 978-0-7881-2234-7.

[10] B. Jayant.D, U. Swapnaja A, A. Sulabha S, and M. Dattatray G, ”Analysis of DAC MAC RBAC access control based models for security”, *IJCA*, vol. 104, no. 5, pp. 6–13, Oct. 18, 2014, ISSN: 09758887. DOI: 10.5120/18196-9115. [Online]. Available: http://research.ijcaonline.org/volume104/number5/pxc3899115.pdf (visited on 02/17/2024).

[11] S. Osborn, ”Mandatory access control and role-based access control revisited”, en, in *Proceedings of the second ACM workshop on Role-based access control - RBAC '97*, Fairfax, Virginia, United States: ACM Press, 1997, pp. 31–40, ISBN: 978-0-89791-985-2. DOI: 10.1145/266741.266751. [Online]. Available: http://portal.acm.org/citation.cfm?doid=266741.266751 (visited on 02/14/2024).

[12] E. System. ”Mandatory access control vs discretionary access control: Which to choose?” (), [Online]. Available: https://www.linkedin.com/pulse/mandatory-access-control-vs-discretionary-which-choose-ekran-system (visited on 04/22/2024).

[13] J. A. Goguen and J. Meseguer, ”Security policies and security models”, in *1982 IEEE Symposium on Security and Privacy*, ISSN: 1540-7993, Apr. 1982,

pp. 11–11. DOI: 10.1109/SP.1982.10014. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6234468 (visited on 02/17/2024).

[14] L. Xu, H. Zhang, X. Du, and C. Wang, ”Research on mandatory access control model for application system”, in *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, vol. 2, 2009, pp. 159–163. DOI: 10.1109/NSWCTC.2009.322.

[15] M. Moyer and M. Abamad, ”Generalized role-based access control”, in *Proceedings 21st International Conference on Distributed Computing Systems*, Apr. 2001, pp. 391–398. DOI: 10.1109/ICDSC.2001.918969. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/918969 (visited on 05/17/2024).

[16] Auth0. ”Role-based access control”, Auth0 Docs. (), [Online]. Available: https://auth0.com/docs/ (visited on 05/17/2024).

[17] RedHat. ”What is role-based access control (RBAC)?” (), [Online]. Available: https://www.redhat.com/en/topics/security/what-is-role-based-access-control (visited on 05/17/2024).

[18] A. D. Brucker and B. Wolff, ”A verification approach for applied system security”, *International Journal on Software Tools for Technology (STTT)*, vol. 7, pp. 233–247, 2005, Publisher: Springer-Verlag. DOI: 10.1007/s10009-004-0176-3. [Online]. Available: https://brucker.ch/bibliography/abstract/brucker.ea-verification-2005.en.html (visited on 05/28/2024).

[19] Google. ”API security best practices | google maps platform”, Google for Developers. (), [Online]. Available: https://developers.google.com/maps/api-security-best-practices (visited on 02/19/2024).

[20] M. B. Jones, J. Bradley, and N. Sakimura, ”JSON Web Token (JWT)”, Internet Engineering Task Force, Request for Comments RFC 7519, May 2015,

Num Pages: 30. DOI: 10.17487/RFC7519. [Online]. Available: https://datatracker.ietf.org/doc/rfc7519 (visited on 03/08/2024).

[21] Auth0, *JSON Web Tokens*. [Online]. Available: https://auth0.com/docs/ (visited on 03/08/2024).

[22] N. Hong, M. Kim, M.-S. Jun, and J. Kang, "A Study on a JWT-Based User Authentication and API Assessment Scheme Using IMEI in a Smart Home Environment", en, *Sustainability*, vol. 9, no. 7, p. 1099, Jul. 2017, Number: 7 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2071-1050. DOI: 10.3390/su9071099. [Online]. Available: https://www.mdpi.com/2071-1050/9/7/1099 (visited on 03/08/2024).

[23] CloudFlare. "What is SAML? | how SAML authentication works". (), [Online]. Available: https://www.cloudflare.com/learning/access-management/what-is-saml/ (visited on 04/21/2024).

[24] Auth0. "What is SAML and how does SAML authentication work", Auth0 - Blog. (), [Online]. Available: https://auth0.com/blog/how-saml-authentication-works/ (visited on 04/21/2024).

[25] S. XML.org. "Advantages of SAML | SAML XML.org". (), [Online]. Available: https://saml.xml.org/advantages-saml (visited on 04/23/2024).

[26] D. Hardt, "The OAuth 2.0 Authorization Framework", Internet Engineering Task Force, Request for Comments RFC 6749, Oct. 2012, Num Pages: 76. DOI: 10.17487/RFC6749. [Online]. Available: https://datatracker.ietf.org/doc/rfc6749 (visited on 02/02/2024).

[27] FusionAuth, *What's new in OAuth 2.1?*, en. [Online]. Available: https://fusionauth.io/blog/whats-new-in-oauth-2-1 (visited on 02/07/2024).

[28] Auth0. "Auth0: Secure access for everyone. but not just anyone.", Auth0. (), [Online]. Available: https://auth0.com/ (visited on 06/19/2024).

[29] Microsoft. ”Microsoft identity platform documentation - microsoft identity platform”. (), [Online]. Available: `https://learn.microsoft.com/en-us/entra/identity-platform/` (visited on 06/14/2024).

[30] OAuth. ”OAuth 2.0 implicit grant type”. (), [Online]. Available: `https://oauth.net/2/grant-types/implicit/` (visited on 06/14/2024).

[31] M. Darwish and A. Ouda, ”Evaluation of an OAuth 2.0 protocol implementation for web server applications”, in *2015 International Conference and Workshop on Computing and Communication (IEMCON)*, Vancouver, BC, Canada: IEEE, Oct. 2015, pp. 1–4, ISBN: 978-1-4799-6908-1. DOI: `10.1109/IEMCON.2015.7344461`. [Online]. Available: `http://ieeexplore.ieee.org/document/7344461/` (visited on 02/21/2024).

[32] OAuth. ”OAuth 2.0 authorization code grant type”. (), [Online]. Available: `https://oauth.net/2/grant-types/authorization-code/` (visited on 06/14/2024).

[33] OAuth. ”OAuth 2.0 client credentials grant type”. (), [Online]. Available: `https://oauth.net/2/grant-types/client-credentials/` (visited on 02/21/2024).

[34] Auth0. ”OpenID connect protocol”, Auth0 Docs. (), [Online]. Available: `https://auth0.com/docs/` (visited on 02/21/2024).

[35] Microsoft. ”What is OpenID connect (OIDC)? | microsoft security”. (), [Online]. Available: `https://www.microsoft.com/en-us/security/business/security-101/what-is-openid-connect-oidc` (visited on 02/21/2024).

[36] C. Bihis, *Mastering OAuth 2. 0: Create Powerful Applications to Interact with Popular Service Providers Such As Facebook, Google, Twitter, and More by Leveraging the OAuth 2. 0 Authorization Framework*. Birmingham, UNITED KINGDOM: Packt Publishing, Limited, 2015, ISBN: 978-1-78439-230-7. [On-

line]. Available: http://ebookcentral.proquest.com/lib/kutu/detail.action?docID=4191352 (visited on 04/23/2024).

[37] M. Argyriou, N. Dragoni, and A. Spognardi, "Security flows in OAuth 2.0 framework: A case study", in *Computer Safety, Reliability, and Security*, S. Tonetta, E. Schoitsch, and F. Bitsch, Eds., Cham: Springer International Publishing, 2017, pp. 396–406, ISBN: 978-3-319-66284-8. DOI: 10.1007/978-3-319-66284-8_33.

[38] ProQuest. "Security evaluation of the OAuth 2.0 framework - ProQuest". (), [Online]. Available: https://www-proquest-com.ezproxy.utu.fi/docview/2093311288?parentSessionId=LotSVF6YPkgz8Q4U7Mi29Cu%2F0l3l7zjCEYm%2BB1NzJPo%3D&pq-origsite=primo&accountid=14774&sourcetype=Scholarly%20Journals (visited on 04/23/2024).

[39] G. D. Maayan. "Should you use OAuth 2.0? pros and cons", Digital Information World. (Nov. 21, 2023), [Online]. Available: https://www.digitalinformationworld.com/2023/11/should-you-use-oauth-20-pros-and-cons.html (visited on 02/19/2024).

[40] N. Sakimura, J. Bradley, and N. Agarwal, "Proof key for code exchange by OAuth public clients", Internet Engineering Task Force, Request for Comments RFC 7636, Sep. 2015, Num Pages: 20. DOI: 10.17487/RFC7636. [Online]. Available: https://datatracker.ietf.org/doc/rfc7636 (visited on 04/06/2024).

[41] OAuth. "Defining scopes", OAuth 2.0 Simplified. (Aug. 17, 2016), [Online]. Available: https://www.oauth.com/oauth2-servers/scope/defining-scopes/ (visited on 05/24/2024).

[42] G. Cloud. "Working with OAuth2 scopes | apigee", Google Cloud. (), [Online]. Available: https://cloud.google.com/apigee/docs/api-platform/security/oauth/working-scopes (visited on 05/24/2024).

[43] Curity. "What are claims and how are they used? | curity". (), [Online]. Available: https://curity.io/resources/learn/what-are-claims-and-how-they-are-used/ (visited on 05/24/2024).

[44] T. Lodderstedt, J. Bradley, A. Labunets, and D. Fett, "OAuth 2.0 Security Best Current Practice", Internet Engineering Task Force, Internet Draft draft-ietf-oauth-security-topics-14, Feb. 2020, Num Pages: 46. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-oauth-security-topics-14 (visited on 02/07/2024).

[45] OAuth, *OAuth 2.1.* [Online]. Available: https://oauth.net/2.1/ (visited on 02/07/2024).

[46] FusionAuth. "Differences between OAuth 2 and OAuth 2.1", FusionAuth. (), [Online]. Available: https://fusionauth.io/articles/oauth/differences-between-oauth-2-oauth-2-1 (visited on 02/21/2024).

[47] IBM. "IBM app connect - application integration software". (), [Online]. Available: https://www.ibm.com/products/app-connect (visited on 04/07/2024).

[48] MuleSoft. "MuleSoft | automate anything. empower everyone.", MuleSoft. (), [Online]. Available: https://www.mulesoft.com (visited on 04/07/2024).

[49] Workato. "The modern leader in automation", Workato. (), [Online]. Available: https://www.workato.com/?utm_source=cpc&utm_medium=adwords&utm_campaign=workato_brand_nordics&utm_content=workato%20specific%20keywords&utm_term=workato&utm_term=search&matchtype=e&device=c&_bt=695546658639&_bn=g&_bk=workato&_bm=e&_bg=147136588790&gad_

source=1&gclid=cj0kcqjwimmwbhdmarisabeq7xqeh70fe6xco9xwbqr0wt2qmolqwc9aylb
wcb (visited on 04/07/2024).

[50] Boomi. "Platform", Boomi. (), [Online]. Available: https://boomi.com/platform/ (visited on 04/07/2024).

[51] IBM. "IBM documentation". (Feb. 12, 2024), [Online]. Available: https://www.ibm.com/docs/en/app-connect/12.0?topic=security-message-flow-overview (visited on 03/24/2024).

[52] IBM, *IBM Documentation*, en-US, Feb. 2024. [Online]. Available: https://www.ibm.com/docs/en/app-connect/12.0?topic=overview-authorization (visited on 03/15/2024).

[53] IBM. "IBM documentation". (Feb. 12, 2024), [Online]. Available: https://www.ibm.com/docs/en/app-connect/12.0?topic=security-enabling-administration (visited on 03/24/2024).

[54] MuleSoft. "MuleSoft documentation | MuleSoft documentation". (), [Online]. Available: https://docs.mulesoft.com/general/ (visited on 05/20/2024).

[55] Spring. "Spring security", Spring Security. (), [Online]. Available: https://spring.io/projects/spring-security (visited on 05/05/2024).

[56] MuleSoft. "Connected apps | MuleSoft documentation". (), [Online]. Available: https://docs.mulesoft.com/access-management/connected-apps-overview (visited on 05/05/2024).

[57] Boomi. "Boomi documentation". (), [Online]. Available: https://help.boomi.com/ (visited on 05/18/2024).

[58] OWASP. "OWASP top ten | OWASP foundation". (), [Online]. Available: https://owasp.org/www-project-top-ten/ (visited on 06/05/2024).

[59] OWASP. ”OWASP top 10 API security risks – 2023 - OWASP API security top 10”. (), [Online]. Available: `https://owasp.org/API-Security/editions/2023/en/0x11-t10/` (visited on 06/05/2024).

[60] A. Reshko. ”How to prevent data breach risks in integration and third-party apps - FortySeven”. Section: IT. (May 29, 2023), [Online]. Available: `https://fortyseven47.com/blog/how-to-prevent-data-breach-risks-in-integration-and-third-party-apps/` (visited on 06/05/2024).

[61] S. Emmerling. ”How to prevent data leakage in the cloud”, Normalyze. (Feb. 22, 2024), [Online]. Available: `https://normalyze.ai/blog/how-to-prevent-data-leakage-in-the-cloud/` (visited on 06/05/2024).

[62] G. eu. ”General data protection regulation (GDPR) compliance guidelines”, GDPR.eu. (), [Online]. Available: `https://gdpr.eu/` (visited on 06/05/2024).

[63] O. f. C. Rights (OCR). ”The security rule”. Last Modified: 2022-10-20T13:34:12-0400. (Sep. 10, 2009), [Online]. Available: `https://www.hhs.gov/hipaa/for-professionals/security/index.html` (visited on 06/05/2024).

[64] Frends, *Frends Docs - Integration Platform Documentation*, en. [Online]. Available: `https://docs.frends.com/en/` (visited on 02/28/2024).

[65] Postman. ”Postman API platform | sign up for free”, Postman. (), [Online]. Available: `https://www.postman.com` (visited on 06/10/2024).