Profile Based Access Control Model Using JSON Web Tokens

Mustafa Albayati
albayati.mustafa@outlook.com

Aslan Murjan
aslan.turkman31@gmail.com

Department of Electrical and Information Technology
Lund University

Supervisors:

Ben Smeets
Lund University

Johnny Wahnström
Axis Communications

Examiner: Thomas Johansson

June 21, 2023

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Currently at Axis, a local role-based access control system is used in devices, which forces the user credentials to be directly installed on the individual devices and the limited selection of roles does not allow for fine-grained access rights. This creates an administrative nightmare in a large scale network and leads to elevated privileges. Instead of this approach a profile based access control can be used.

The goal of this thesis work was to design an access control system for profile based access control, utilizing JSON Web Tokens (JWT) for distribution. How profile based access control works was investigated and the possibilities of enforcing dynamic, user defined and distributed profiles were explored in contrast to static access tables. This system allows an admin to create custom access control profiles depending on the use case, instead of being limited by the roles or profiles preinstalled on the device. Open ID Connect was used for user authentication and authorization of profiles.

The system's design was implemented through an ambitious Proof-of-Concept (PoC) that encompassed numerous components with the primary objective of evaluating the feasibility of incorporating the proposed idea into an actual production system. The innovative features of the resulting system design have been condensed and included in a patent application, which was subsequently filed by Axis.

Acknowledgement

We would like to express our sincere gratitude to Axis Communications and the hiring manager Robert Svensson, for providing us with the opportunity to conduct this research and for their support throughout this study. We are particularly grateful to Johnny Wahnström at Axis for supervising this project and providing invaluable guidance and feedback that helped us to achieve our goals.

We would like to thank Ben Smeets for supervising this thesis at the Department of Electrical and Information Technology, at Faculty of Engineering of Lund University. His expert advice, encouragement, and support were instrumental in completing this study successfully.

We would also like to thank Boman Axelsson for the help with the filing of a patent application under European patent application number 22214794.4.

Finally, we would like to express our deepest gratitude to our families for their unwavering support and encouragement throughout our academic career. Without their love and support, we would not have been able to achieve our goals and complete this thesis.

Table of Contents

Definitions

- **ACL**: Access-control list.
- **Forward Proxy:** A server that pass requests from isolated network to a public internet.
- **CGI**: Common Gateway Interface is a middle-ware that passes data between the web-server and CGI applications. These applications are small scripts or compiled programs.
- **IdP**: A system that creates and manages identity information of users.
- **IoT**: Internet of Things are devices that can connect and exchange data with other devices and systems over the internet or other communication networks.
- **OACS**: OACS is the deliberation of OIDC Access Control System, which is the name used for the final solution of this thesis work.
- **OAuth 2.0**: An industry-standard protocol for authorization.
- **JWT**: JSON Web Token.
- **OIDC**: Open ID Connect, identity layer on top of OAuth 2.0 protocol for authentication and authorization.
- **RBAC**: Role-based access control. Access rights are associated with different roles that defines a type of user. These roles are assigned to users to give them a certain level of access.
- **RoT**: Root of Trust, A cryptographic device that can securely store keys and sign data, such as a TPM or Secure Element (SE).
- **TLS**: Transport Layer Security
- **Claim:** A claim is a statement about a user associated with a scope. It can be dynamically using user attributes defined by the identity provider or statically mapped data.
- **Scope:** A grouping of claims.
- **User-agent:** Any external client that communicates with the device on the users behalf.

- **External Service:** Any service that can be accessed from outside the device.
- **OIDC Client:** A registered software that is authorized to communicate with the provider on behalf of the user to handle user authentication and authorization.
- **OIDC Client id:** A unique user (admin) defined name to identify the OIDC client registered on the provider.
- **OIDC Client secret:** A predefined secret for client authentication.
- **OIDC Client key:** A private key used to sign single use authentication tokens instead of a client secret.
- **OACS:** OIDC Access Control System, A library and D-Bus service solution that makes up a profile based access control solution using JSON Web Tokens (JWT).
- **D-Bus:** An interprocess communication protocol.
- **Profile:** A set of features defined as a scope in OAuth 2.0 with the targeted OACS version.
- **Access Token:** Represents authorization to a specific resource.
- **Refresh Token:** Represents authorization to retrieve a new access token from the provider without user interaction. This allows providers to use short lived access tokens without having to involve the user when they expire.
- **Zone:** A group of regionally related devices.

List of Figures

Chapter 1
Introduction

The need for improved access control methodologies increases over time to limit the actions or operations that a legitimate computer system user can perform. Large IoT networks introduce additional challenges that cannot always be satisfied with current solutions. This thesis work is about designing a profile based access control system using open sourced technologies, for a network of devices (in this case Axis devices), using JSON Web Tokens (JWT) to achieve an authorized access to many devices.

Currently a local role-based access control (RBAC) system is enforced by the devices. RBAC is a security model that is designed to restrict system access to authorized users based on their roles within an organization. This means that individuals or groups are only given access to resources that are required for them to perform their specific job functions.

The RBAC model is used to help organizations control access to information and resources in a way that minimizes the risk of unauthorized access or data breaches. This is especially important in organizations where sensitive data or valuable intellectual property is stored, or where regulatory compliance is required.

The RBAC model is based on the principle that access to resources should be granted based on the roles that individuals or groups hold within an organization. This ensures that individuals are only given access to the resources that they need to perform their job functions, and nothing more. But this model also creates an administrative nightmare in a large scale network. RBAC main weakness is inherent in reliance on manual input, and its constant need for maintenance [34]. Instead of this approach a profile based access control can be used.

1.1 Problem and Goal

Currently, the Axis system utilizes a local RBAC mechanism that requires user credentials to be directly installed on individual devices see Section 1. However, the limited number of available roles in this system results in inadequate fine-grained access control. As a result, this mechanism presents an administrative challenge for large-scale networks.

How a profile-based access control using JWT works in such a system will be investigated. The possibilities of enforcing dynamic, user-defined, and distributed profiles will be explored in contrast to static access tables. The system will allow the admin to create custom access control profiles depending on the use case instead of being limited by the roles or profiles preinstalled on the device.

These access control profiles will provide a high level abstraction of what a user is allowed to do on the device. They will be stored in a central server where user authentication will take place before authorization is provided. The signed JWTs that contain the profile will then be verified and handled by the individual device, eliminating the need to manage user credentials and rights on individual devices.

The following questions will be answered:

- How to create and handle the profiles?
  - What rights can be assigned? How to identify them?
  - How can a role be expressed by a profile, in all details?
- Where should the JWTs be verified and the profile enforced?
  - By the server, on entry?
  - Behind the server, by an API?
  - On a lower level, so that it can be used by other protocols that don't use HTTP?
    * Such as WebSocket, MQTT and GraphQL.3. How should the revocation of access rights be handled?

1.2 Example scenarios

The following example scenarios illustrate some of the core issues that this thesis project aims to address.

Scenario 1. Handling user credentials

Background: A user is the operator (an underprivileged user with permission to operate the device) of a network of 200 networked loudspeakers. The user's

password has been compromised or in accordance with the company's policy, the password needs to be replaced.
Problem: As an representation of the user credentials that are installed on the individual devices. The password needs to be changed on all the devices, individually.
Proposed solution: The user credentials are stored in a central server which handles authentication and authorizes signed tokens with access control profiles that can be verified and enforced by the individual device.

Scenario 2. Individual Access Rights

Background: A device provides a set of roles that can be assigned to a user. These roles are not always able to meet the requirements of the user.
Problem: To overcome this limitation, a user will then be assigned a role with higher privileges than what is required.
Proposed solution: With personalized access profiles, a user can be assigned rights to the resources needed to perform their job. For instance, a user that needs resources with higher privileges than what's allowed for an operator doesn't need to be assigned an admin role (the only higher privileged role in this scenario), but can instead be explicitly assigned the required rights.

Scenario 3. Temporarily Access Rights

Background: A technician needs temporary access to one or more devices.
Problem: Currently there is no solution to provide temporary access. A temporary user needs to be added to the individual devices manually and then removed afterward. Which requires administrative work.
Proposed solution: With a centralized authentication solution and the ability to create custom access profiles. It will be possible to create profiles with an expiration time, which can be used to give users temporary access.

1.3 Limitations and boundary conditions

These are some additional conditions applied to the work due to system requirements and future expectations of service deployment.

1.3.1 System

The devices are running on an embedded Linux environment based on the Yocto project. The solution should be suitable to apply to this type of environment with reasonable work. The Yocto project provides a reference distribution called Poky, which contains the OpenEmbedded build system. OpenEmbedded is a build framework for embedded Linux that offers a cross-compile environment for developers to create complete Linux firmware for embedded systems. The necessary changes are expected to be made on the OpenEmbedded Core layer in order to use system daemon (systemd) as the init daemon. This allows for the use

of systemd units that can perform tasks and be managed by the system daemon. Systemd units can be configured to activate on startup or on D-Bus (see Section 2.9) requests. The solution must exhibit a sufficiently low level of performance and memory overhead to be considered an acceptable trade-off.

1.3.2 Open Source

The proposal solution should be based on open-source technologies. Technologies based on proprietary tools should not be included. This is to avoid licence limitations and other limitations put on the customer. Different organizations have different requirements on their infrastructure. Some may prefer using a managed service from service providers such as Microsoft and Google, while others may have no trust in external parties and want to have total control of their system. This level of flexibility will be provided with open source protocols and technologies.

OpenID Connect is used for authentication and authorization, and is a base of the final solution. This will allow the customer to use any arbitrarily OpenID Connect provider from service providers or a self hosted server.

1.3.3 Clients

The solution is expected to work on a diverse set of clients, such as CLI tools, web applications and PC software. It's therefore expected by the proposed solution to operate in a way that is suitable for these kinds of clients.

1.4 Related Work

This section provides an overview of a selection of literature and research related to the topic:

1. The paper titled "Token-Based Access Control" [37] describes how to profile-based access control can be applied to a network of devices by recording the credential information inside blockchain-based smart contracts. A brief description of the general data structure of the access control token is provided.
2. The article titled "A User Profile Based Access Control Model and Architecture" [25] acknowledges the importance of personalization in access control to provide services dependent on the user's need and the lack of adaptability in current access control models. The authors propose an adaptable access control model and the related architecture-based security policies dependent on the user's profile.
3. The paper titled "Profile based access control model in cloud computing environment" paper [26] provides solutions for challenges like high access time, high searching cost, and the problem of data redundancy. Access control models based on profile tokens deliver the solutions for this demand. This article also provides valuable information on handling resource rights

without the resource owner being online (available). The suggested solutions in this paper will be taken into consideration.

4. The article titled "A Revised Model for RBAC" [15] provides information about RBAC. The core features are explained together with the role hierarchy and the dynamic/static resources.

5. The publication titled "JSON Web Token (JWT)" [17] specification from the internet engineering task force.

The related works section of this thesis sheds light on the diverse range of research conducted in the field of access control systems. Upon reflection, it is evident that access control is a critical aspect of security for both physical and digital systems, ensuring that only authorized individuals are granted access to specific resources or areas while preventing unauthorized access.

One of the observations from this section is the emphasis on the need for flexible and adaptable access control systems. Several reviewed studies highlight the limitations of traditional access control systems, which can be inflexible and challenging to update. As technology evolves, access control systems must keep pace, with the ability to adapt to new threats and access requirements.

Overall, the related works section provides a valuable overview of the current state of research in access control systems. This information will serve as a useful reference point for this thesis work research and for the development of practical access control solution.

1.5 Contributions

This thesis explores the potential of utilizing a profile-based access control mechanism employing JSON Web Tokens (JWT) in the context of managing user space services on an IoT device. The current local RBAC mechanism in the Axis system suffers from limitations in fine-grained access control, making it challenging to administer large-scale networks. As an alternative to static access tables, the thesis investigates the feasibility of enforcing dynamic, user-defined, and distributed profiles to overcome this limitation.

The thesis proposes a solution that empowers administrators to create customized access control profiles based on specific use cases. These profiles abstract the high-level user permissions on the device. The profiles are stored in a centralized server, and user authentication takes place before granting authorization. The signed JWTs that encapsulate the profile are then verified and processed by the individual device, obviating the need to manage user credentials and access rights on individual devices.

To implement this solution, the thesis addresses several questions related to the creation and handling of profiles, assignment of rights, expression of roles, and profile verification and enforcement. In particular, the thesis examines where the JWTs should be verified and the profile enforced, as well as the revocation of access rights.

The proposed solution enables more flexible and scalable access control that can better adapt to evolving requirements in large-scale networks. By allowing

administrators to define more precise user access through customized access control profiles, the solution enhances security and reduces administrative overhead. This thesis can serve as a resource for organizations that are interested in adopting a profile-based access control mechanism using JWTs.

1.6 Work division

The contributions to this study were equally distributed between the two authors. Both authors collaborated on designing, researching, and validating the results to ensure quality. To facilitate parallel work, a few tasks were carried out individually. Aslan Murjan was primarily responsible for conducting a comprehensive background study and gathering relevant information related to the problem at hand, While Mustafa Albayati had a primary focus on designing the architecture, implementing the proof of concept, and testing tools that were tailored to the proposed solution. Both authors also participated in evaluating the solution and monitoring the results. Regarding the report, both authors contributed equally to documenting this case study.

Chapter 2
Background Theory

In this section, we present a comprehensive overview of the key theoretical concepts and frameworks that make up the foundation of this thesis.

2.1 Access Profile

An Access Profile is an association between a user and a list of protected objects that the user may fully or partially access. Objects such as protected endpoints, functionality and documents on the file-system. In Chapter 4.4 we review why access profile is useful for our system.

2.2 JSON Web Tokens

JSON Web Tokens or JWT is a standard for transmitting information as JSON object between network entities in an organized and secure way, the technical specifications and organizational notes about JWT are written in RFC 7519 [2].

The information that is sent using JWT is verifiable and immutable because it is digitally signed by a secret or by a public/private key pair.
JWT has an essential role in this project. A JWT token is used in every subsequent request between the servers to authorize the user to access resources and services or exchange permitted data [2].

Every JWT token consists of three parts, the header, the body, and the signature, and all three parts are encoded in base64. The token looks like shown in Figure 2.1.

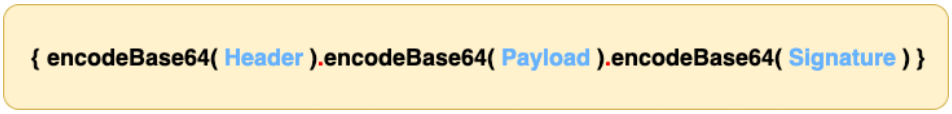

Figure 2.1: JWT Format

- The header contains the token type and an identifier of the signing algorithm.
- The payload consists of claims about the user and additional data.
- The signature is created by digesting and encrypting a dot separated string of the base 64 encoded header concatenated with the base 64 encoded body.

2.3 Identity Provider

An identity provider(IdP) is a server that contains and maintains the identity information of the system's users. It also provides an authentication mechanism for the clients such as mobile and web applications. It is also used to provide an SSO (single-sign-on) service in OIDC system [28]. In Chapter 4.3 we review why an identity provider is useful for our system.

2.4 OAuth 2.0

The OAuth 2.0 Authorization protocol is an Internet Standards Track protocol that enables obtaining controlled access to an HTTP service by third-party applications. The application uses the "Scope" parameter to specify the access rights it needs, such as user information and Home Address. The authorization server responds with the granted access rights (if the granted access rights differ from the requested access rights) using the "Scope" parameter [13].

OAuth 2.0 introduces an authorization layer, a mechanism in which the client requests access to resources controlled by the resource owner and hosted by the resource provider [13].

In this project, the resource server and an authorization server are the same server, but they exist as different endpoints on that server. The authorization grant denotes the client's entitlement to request access or refresh tokens. The resource owner authorizes the authorization process through their user consent. surely

2.4.1 Client Authentication

Client authentication is a mechanism meant to prevent unauthorized OIDC clients from communicating with the provider [3]. Client authentication prevents a malicious third party from consuming the resources of the provider or obtaining potentially sensitive information by pretending to be the client (see Section 2.4.4).

Client Secret

The client secret is a String-type data that serves to authenticate, via a proof of possession, the OIDC client in the authorization server. The secret must be challenging enough to make guessing improbable, and it is recommended that a 256-bit value converted to hexadecimal representation be used as the client's secret.

Mutual TLS authentication

Two-way authentication is the principle that defines the mechanism of two parties authenticating each other. Mutual authentication in TLS is accomplished by using public key cryptography and certifications. TLS is used to transmit sensitive data and to ensure the security of the data [3].

Signed JWT authentication

OpenID Connect (OIDC) clients are authenticated using the JWT Bearer Token, which is a form of token applied for authentication and authorization in web-based systems. This mechanism constitutes the authorization grant type for OAuth 2.0. This method is another client authentication mechanism that utilizes public key infrastructure in addition to mutual TLS authentication. A single use JWT is generated to represent the client with a unique id and signed with the client device's private key. The OIDC provider can then verify the token using the public key and grant access if successful. This approach was preferred over mutual TLS authentication because of the requirements and limitations put on the transport layer. Figure 2.2 describes the process of generating the client access token [17].

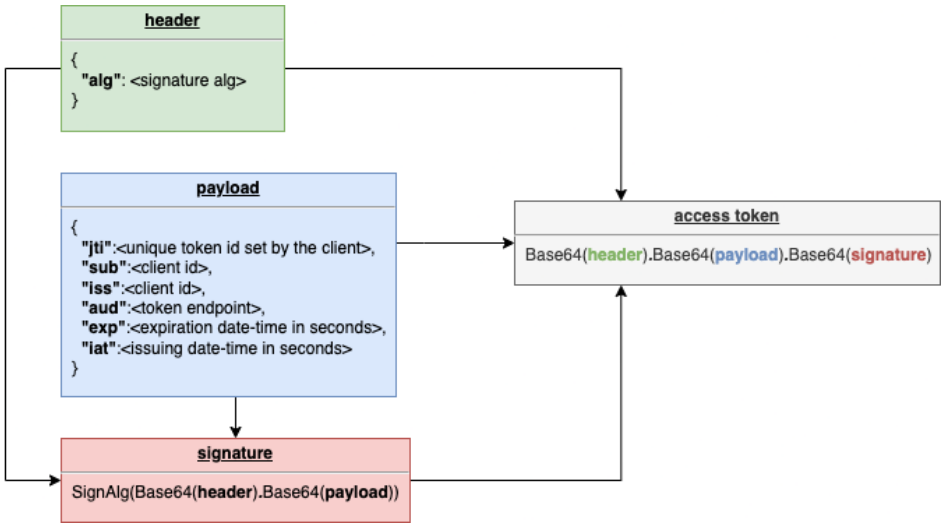

Figure 2.2: Client Access Token

2.4.2 Implicit Flow

The Implicit flow, as a method of obtaining an access token in OAuth 2.0, is an open standard for authorization. It is primarily utilized for client-side applications in which the confidentiality of the client's secret cannot be maintained, and the use of the so-called Authorization Code flow(see Section 2.4.3) is not feasible.

In the Implicit flow, the client application redirects the user's browser to the authorization server. The authorization server subsequently redirects the user's browser back to the client application with an access token located within the URL fragment (the portion following the "#" in the URL). The access token is subsequently extracted from the URL and utilized to gain access to protected resources [12].

Additionally, it is noteworthy that the Implicit flow is currently considered less secure due to the fact that it does not provide the client with a refresh token. Furthermore, the access token is returned within the URL fragment, which is not accessible to the server side and can only be accessed by the client side, thereby limiting its functionality. In Appendix A.2 we review how this flow is used in our system

2.4.3 Authorization Code Flow

Authorization Code Flow is the primary authorization method in OAuth 2.0 for server side applications. Unlike the Implicit Flow, it requires that there is a backend were the client secret can be kept and exchanged securely. It is a redirect based method.

The login process is initiated by the backend with a request to the authorization server. User authenticates using an available authentication option and provides consent to the requested resources. The authorization server redirects the user back to the application together with an authorization code that can only be used once. Unlike the Implicit Flow, the access token isn't retrieved directly. Instead, the authorization code is sent back to the authorization server together with client id and client secret.

If the provided client id, client secret and the authorization code are all valid, the authorization server will respond with an ID token, Access token and an optional Refresh Token. The Refresh Token can be stored in a HTTP only secure token on the client side or in a server session. It can be used to retrieve new access tokens without requiring the user to interfere. This allows for shorter lived access tokens compared to the Implicit Flow [12].

Authorization Code Flow with Proof Key Code Exchange

Proof Key Code Exchange (PKCE) is an extension to Authorization Code Flow which provides an alternative method for client side application instead of the implicit flow. Storing a client secret on the client is still not a propper option, as it can be extracted one way or another. Instead, PKCE provides a way for the client

application to give proof that the authorization code belongs to it.

PKCE introduces three new parameters, "code_challenge", "code_verfier" and "code_challenge_method". The client will request an authorization code, however instead of a client secret, a random code_challenge will be sent together with a code_challenge_method which states the method used to transform the code_verifier to the code_challenge and retrieve an authorization code. The code is then sent to the authorization server together with the code_verifier.

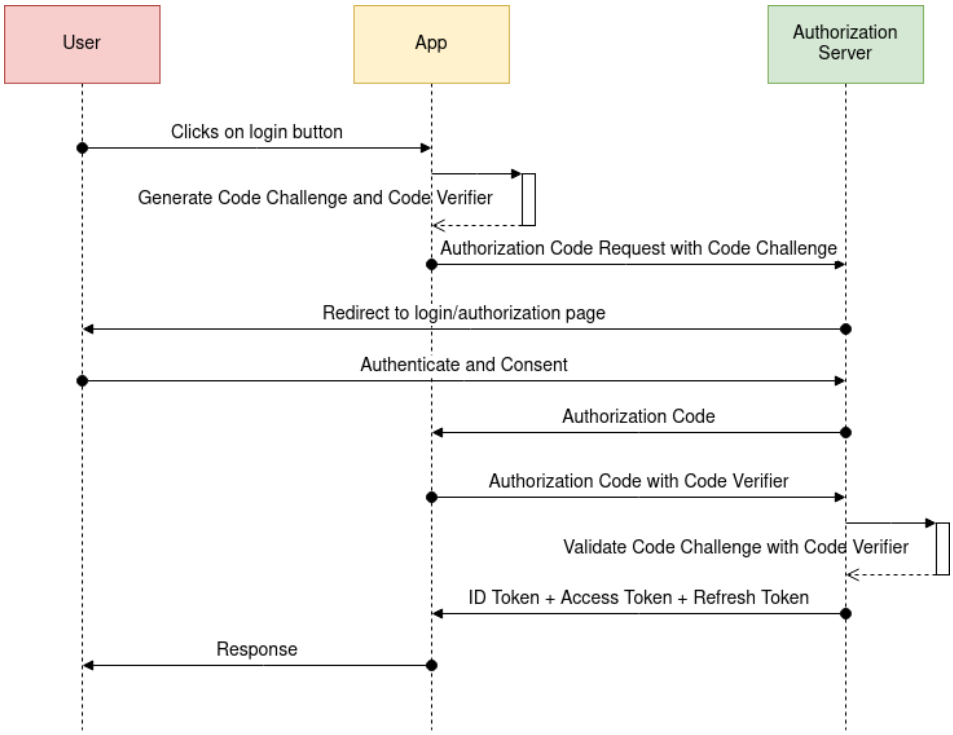

Figure 2.3: Proof Key for Code Exchange

The authorization server can use the code_challenge_method to transform the code_verifier and confirm that the code_challenge provided by the application to retrieve the authorization code belongs to the application now requesting an access token using that authorization code. This method is recommended for single-page web-applications and mobile applications [33].

2.4.4 Client Credential Flow

The Client Credentials flow, as a method of obtaining an access token in OAuth 2.0, is utilized for accessing protected resources on behalf of a client rather than an individual user. It is a machine-to-machine authentication mechanism commonly employed in server-to-server communication, where the client is a server-side application [12].

In the Client Credentials flow, the client initiates a request to the authorization server, providing its client ID and secret (or a client assertion). In exchange, the authorization server issues an access token, which can be utilized to access the protected resources on behalf of the client. In Appendix A.3 we review how this flow is used in our system

2.4.5 Device Code Flow

Device Code Flow is an OAuth 2.0 method meant for input restrained devices and clients, An input-restrained device is a device that has limitations or restrictions on the types of input it can accept for example cameras. This can refer to a variety of devices, such as electronic devices, computer peripherals, or even physical machines.). Unlike redirect based methods like implicit flow and authorization code flow, device code flow is a decoupled method. Rather than directly authenticating the user, the user is asked to visit a link on another device and authorize the device. The authorization server responds to an authorization request with a "user_code", "device_code", "verification_url", "expires_in" (expiration time for the codes) and "interval" (the pooling interval).

The user is instructed to visit the verification link and use the "user_code" in order to be authenticated and authorized access to the device. Meanwhile, the device will be using the "device_code" while pooling the authorization server for the authorized token as instructed by the authorization server. The device will then obtain an access token and an optional refresh token. It can use the refresh token to keep the user logged in for a longer period of time instead of using short-lived access tokens. This reduces the possibility of a leaked access token to be reused for malicious intent [4]. In Appendix A.2 we review how this flow is used in our system

2.4.6 Refresh Token Rotation

Refresh tokens are commonly employed to acquire new access tokens following the expiration of short-lived access tokens, and they typically have a longer lifespan. Native mobile applications often utilize refresh tokens in conjunction with short-lived access tokens to facilitate a seamless user experience, without the need to generate long-lived access tokens [12].

With refresh token rotation enabled, every time an application exchanges a refresh token to get a new access token, a new refresh token is also issued, invalidating the previous one. Thus securing the long-lived refresh token that, if compromised, could provide illegitimate access to resources. As refresh tokens are continually exchanged and invalidated, the threat is reduced [24]. In Appendix A.4 we review how this flow is used in our system

2.5 Open ID Connect

OpenID Connect or OIDC is a protocol that defines how the client can verify the end-user's identity by authenticating the user using an authorization server. It also defines how to obtain the profile information about the end user in the form of JWT [27]. The OIDC is responsible for requesting and receiving information about the authenticated end-user. OpenID Connect is built on top of the OAuth 2.0 protocol, which allows using web authorization codes flow, like Device code flow and client credentials code flow used in this project [12].

2.6 CGI

CGI or Common Gateway Interface is used by the servers to run an external program remotely on other servers. Usually, this file is saved with an extension (.cgi) and is written in C or Perl programming languages. When a client, a web server, or a user requests a URL address that contains a CGI directory, it will be executed by the HTTP server and send back the output to the user [32]. The CGI files are entry points to the system services.

2.7 Apache

The Apache software platform is an open-source initiative that is maintained and developed by an open community. The software is released under Apache License 2.0. The project's version of Apache operates on the Linux distribution and serves as a flexible, highly configurable, and extensible web server, which renders it a compelling option for embedded projects. Additionally, Apache can utilize CGI endpoints for enhanced functionality [10]

2.8 Systemd

System daemon (systemd) is a service and system manager for Linux operating system, designed especially for Linux kernels. This service runs as the first process on boot (PID 1). In other words, it becomes an init system that starts user space services later. It provides features like the parallel startup of system services at boot time, on-demand starting of daemons, and processes tracking using Linux control groups. Systemd offers the concept of "systemd units". These units are services represented by configuration files that are located in Systemd Unit Files Locations and contain all information about the services, sockets, and objects that are associated with init system [30].

2.9 D-Bus

D-Bus or desktop bus is an inter-processor communication protocol (IPC). It is a way of communication between applications to talk to each other or to allow multiple processes that are running on the same host to exchange data in a regulated way [29]. The D-Bus protocol ensures that the processes aren't communicated with unsuitable services by specifying how to exchange the data and what metadata is available to exchange. As well as offered introspection of services and the ability to trace the message traffic [14].

It carries messages as discrete items instead of continuous data streams. Both one-to-one messaging and broadcast to subscribed clients is possible. D-Bus deals with data in binary form and can reject ill-formed messages. Exchange on the bus requires a communication endpoint on one end, that is called an object. Objects

are created by client processes and exist within the context of the client's connection to the bus. A client can create any number of objects.

Objects contain members such as methods and signals specified in interfaces. Methods are specific named actions. They can have optional input and output parameters, and are declared in the interface with the expected data types. Methods are one-to-one, request-response based calls between two clients over the bus. Clients may on the other hand subscribe to signals exposed by another client. When an object emits a signal, all subscribed clients will receive a copy of the signal. The signal may carry parameters like a method invocation, however, there are no replies to signals. An object may contain multiple interfaces [19].

2.10 Virtual Machine

A virtual machine (VM) is a software emulation of a physical computer. It allows multiple operating systems to run on the same physical machine, each with its own virtual resources such as CPU, memory, and storage [11].

2.11 Containerization

The process of bundling together all the resources required to run software on any infrastructure that we use is called containerization. As the Keycloak server would be running locally in a development environment, a containerization solution was approached allowing us to easily deploy the complete server on different environments and perverse the expected behavior.

2.11.1 Docker

An open-source platform that enables developers to construct and refine a system or tools, and execute them within distinct containers. These containers operate as namespace-isolated runtime environments, sharing the use of the host's kernel and providing the dependencies required to run the code [5].

Chapter 3
Approach

In this chapter, we summarize the approach of our thesis project. The thesis work started with a brief introduction to the problem by Axis Communications. The first look at the problem happened via a short description from the supervising engineer at the workplace.

3.1 Literature Study

After the initial problem description, we performed a literature study. First, an initial set of related literature and data was gathered on the subject. This study was used to explore and find research questions from an academic aspect, look in Section 1.1. The resources were evaluated and only reputable publications and information from the official authorities were utilized. Scientific articles and thesis works published by others proved to be a good way to understand the problem better. These papers were found in Google Scholar engine search and LUBsearch database. This study was essential to highlight how similar problems were solved, look in Section 1.4, which helped us determine the project's priorities and formulate the solution.

3.2 Design Phase

The next phase, the design phase(see Figure 3.1), started with a set of preparations. The first step was to identify the best tools to solve the problem based on the limitations and goals of the project. The next step was to define the concept of profile as a data structure, which is the cornerstone of the access token system. The description of the profile can be found in Section 4.4. The proper code flow was then determined to provide confidentiality, authenticity, and integrity to the system.

The last step of preparation was to create a low-level design(see Figure 3.1) that takes the access token and distributes it to all needed services securely.
Following the preparations, a solution design is produced and analysed internally. Complete design solutions were reviewed together with our supervisors.

The final solution was refined through several design iterations after supervisors feedback.

3.3 Iterative Demonstration & Feedback

During this phase, a series of meetings were conducted with LTH department supervisor and experienced engineers from Axis company to gather feedback on the proposed solution and OIDC. Based on the feedback received from these groups, a model modification process was initiated to align the proposed solution with the desired outcome. This iterative step was repeated multiple times until the intended outcome was achieved.

Lastly a proof of concept was produced and presented internally at Axis Communications, before the thesis project was finalized with a thesis paper and final presentation.

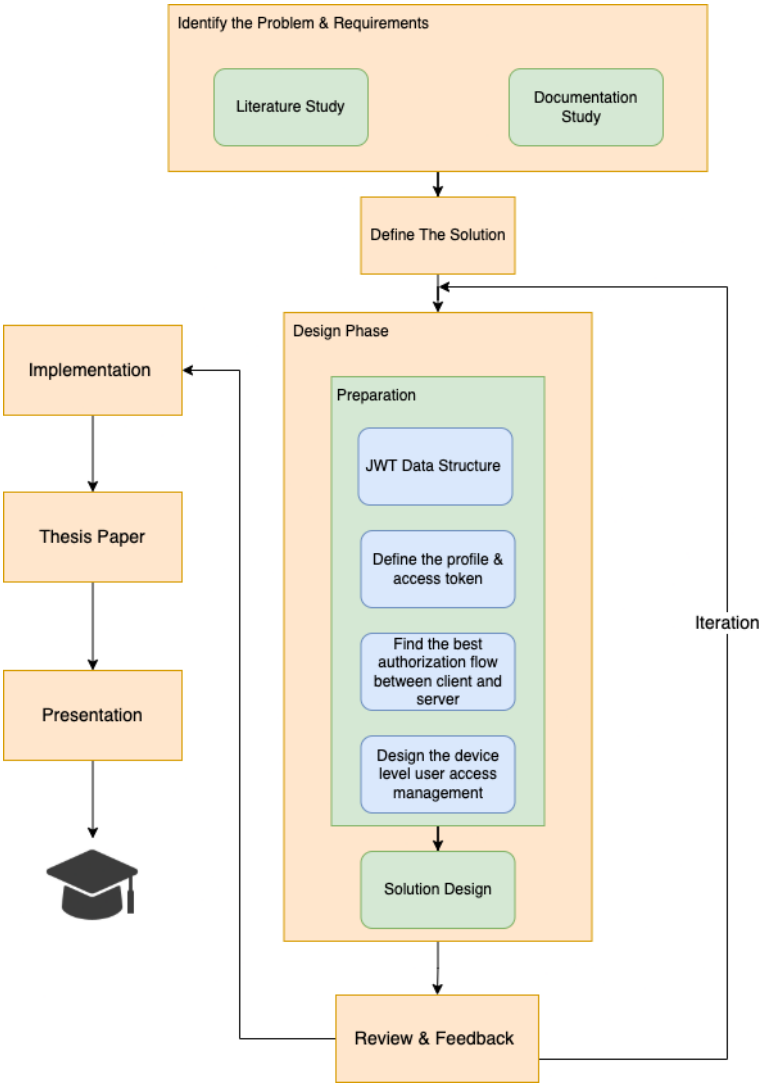

Figure 3.1: Overview Of Project Process.

Chapter 4
Solution

This chapter will detail the design solution of this thesis work. It starts with the network architecture's big picture, then diving into the choice of technologies and SW components, design decisions and code flows used in this thesis, in addition to profile data structure and client access token. This section will also introduce and describe a service ticket that will be used to authorize tasks. The resulting system is named OIDC Access Control System (OACS). The deliberation OACS will be used throughout the remainder of the thesis report.

4.1 Network Architecture

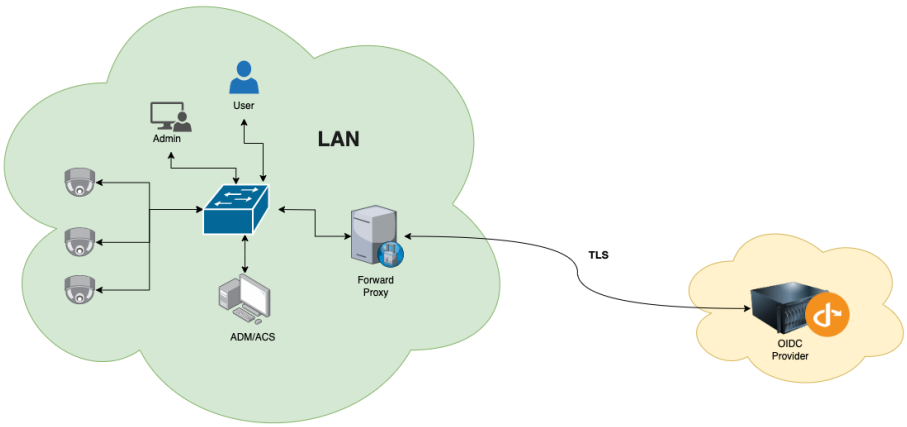

Figure 4.1: Network Architecture

Figure 4.1 depicts the network architecture of the solution proposal. A local network consists of cameras, some kind of management software (Axis Device Manager, Axis Camera Station, ...), a forward proxy to communicate with the OIDC provider and end users. Each camera acts as its own OIDC client using the same OIDC Client id and OIDC Client key. The id and key need to be installed on

the devices during setup before use, otherwise the client authentication against the OIDC provider will fail. The client key needs to be stored in a tamper-resistant device to establish a root of trust. With devices as clients, the tokens will travel less over the local network and also, there will be no central server acting as an OIDC Client back-end, that can become a potential single point of failure or a bottleneck that congests the traffic to the devices. Each device should be configured to protect the external communications with HTTP over TLS in order to prevent tokens from being leaked.

4.2 Proof Of Concept Setup

In this section, we summarize the choices made for the setup of the proof of concept (POC).

4.2.1 Identity and Access Management (IAM)

We needed a solution for an Identity and Access Management tool, in order to configure and experiment with an OpenID Connect 1.0 compatible client. Keycloak was one of such tools that provided us with a certified client with the convience of a built-in identity provider, where mock users can easily be generated and deployed into the ecosystem.

Keycloak is an open source project licensed with Apache License 2.0, written by Red Hat. It is used for "Identity and Access Management". It can be configured to work with industry standard protocols like OAuth 2.0 and OpenID Connect.

Keycloak is deployed as an independent server for authenticating users with the application by saving the users' identity and their roles, permissions, and profiles in a database (in keycloak) in order to send it when needed in the form of JSON tokens [20].

4.2.2 Containerization

Docker was chosen as the containerization solution as Keycloak provided a ready to use Docker image for development purposes as well as production [31].

4.2.3 Device Virtualization

The Yocto project uses an implementation of Quick EMUlator (QEMU) open source emulation software as part of its development tool-set. This utility was used to virtualize an arbitrarily ARM based machine to run a compatible firmware compiled with OpenEmbedded build system. This firmware was built on top of the Poky reference distribution provided by Yocto. The proof of concept was built as a layer and tested in the virtualized environment.

4.2.4 System Architecture

This is the final architecture used for the proof of concept. The Keycloak AIM is acting as an OIDC provider running inside a docker container isolated by the runtime environment. An arbitrarily ARM based device is virtualized by a QEMU virtual machine running the Linux distribution housing the device side of the solution.

A cryptographic device such as TPM (Trusted Platform Module) or SE (Secure Element) was not used in the proof of concept. As the proper use of these

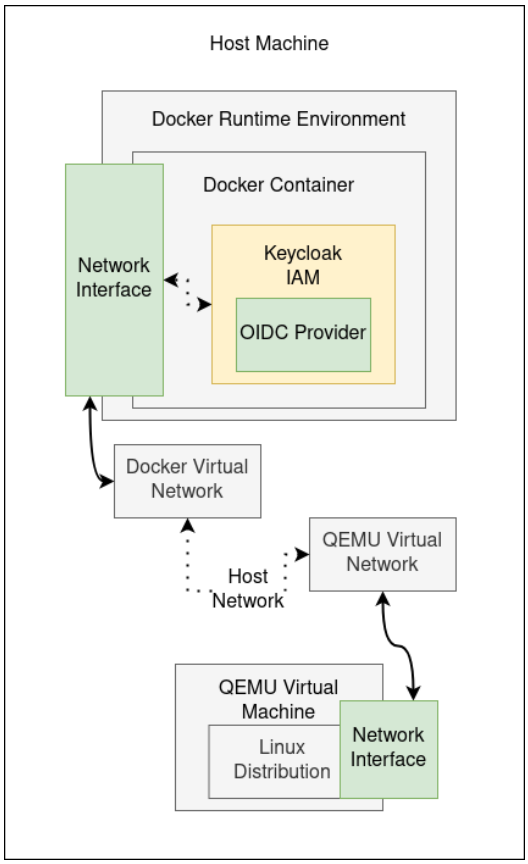

Figure 4.2: POC Architecture

devices are already well known and not interesting for this study. We are assuming that for this solution to work, one of these devices are properly implemented.

The virtual devices and the containers are located on two different virtual networks accessible via the network stack of the host machine. More details about the device side implementation and the role of the provider are explained later in Section 4.3.

4.2.5 Aim of the POC

The aim of the proof of concept is to demonstrate that the central principles of an access control system as described in this chapter are possible to implement and apply in a real life scenario. Already well established technologies such as HTTPS, TPM, and a physical network are excluded as the presence of these technologies won't produce new knowledge. The following acceptance criteria will be used to assess the completeness of the solution.

1. Demonstrate that an arbitrarily user can be authenticated and authorized access to necessary resources through an OIDC provider.
2. Demonstrate that an arbitrarily device service can be utilized by an authenticated user with correct access rights.
3. Demonstrate that the same service is unavailable when the required permission is not provided.
4. Demonstrate the life-cycle of the authorization object(s). When are they created? Where are they used? When are they disposed of?

4.3 OACS Architecture

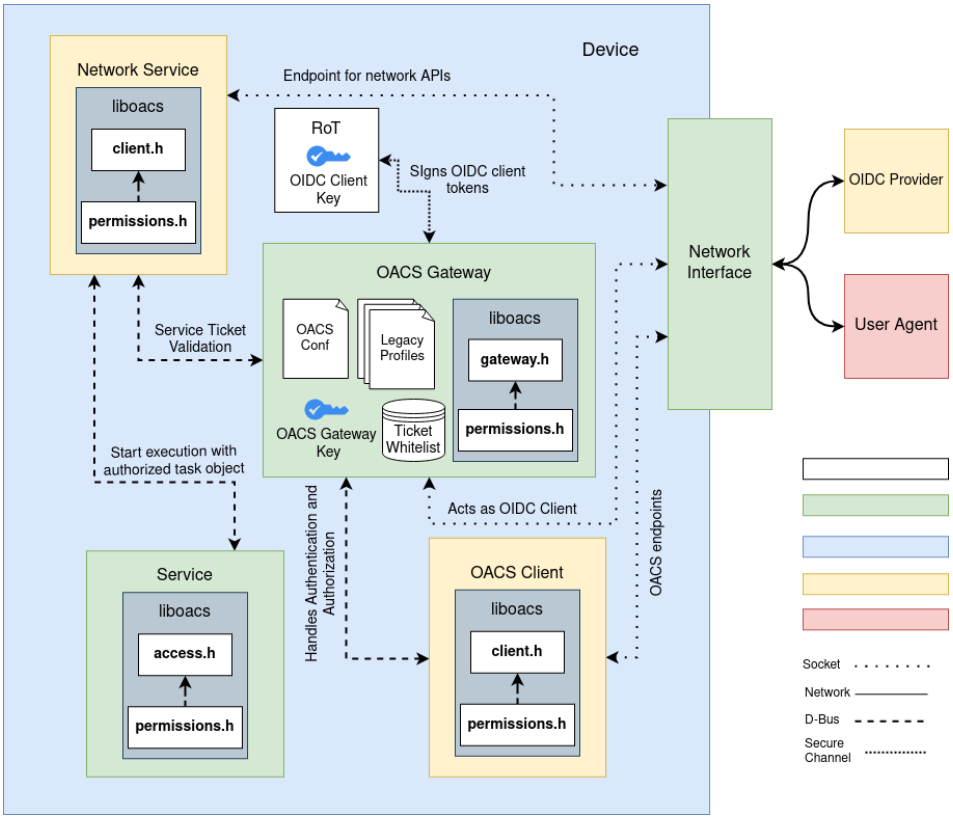

Figure 4.3: OACS Architecture

OACS utilizes four different public endpoints by using the OACS Client, in order to handle login, get refresh token, authorize ticket, and revocation of the refresh token. The proof of concept includes four CGI endpoints for this purpose. These CGI endpoints are resolvers that make use of the liboacs library to communicate with the OACS Gateway over D-Bus to serve requests as shown in Figure 4.3.

The OACS Gateway further communicates with the Root of Trust (RoT) hardware device to sign client authentication tokens as previously defined. The RoT protects the private key used for client authentication. The OACS Gateway communicates directly with the OIDC Provider. All logic and functionalities are provided by the OACS Gateway (see Section 4.10). The CGI endpoints only act as entry points and potentially translate or in other way handle response from the gateway.

The OACS Client will only act as a relying party to establish communication

with the OACS Gateway. The OACS Gateway D-Bus service will only be allowed to be used by the "oacs" group on the host OS. Thus, only external entry points in the system can process user access as intended, reducing the chances of exposure to a rogue service under the control of a malicious user.

The user login is handled through a slightly modified form of OAuth 2.0 Device authorization flow, where the device code (Section 2.4.5) is kept by the requesting user agent, instead of the device. A detailed description of the login flow can be found in the appendix, under Section A.2. Device code flow was chosen for OAuth 2.0 authorization as a decoupled process, unlike the redirect-based authorization code flow as described in Section 2.4.3 or the extended version utilizing Proof Key for Code Exchange (PKCE) mentioned in Section 2.4.3. Authorization code flow with PKCE is the industry standard, however, it's designed for web-clients that can handle an HTTP redirect response as defined by RFC-7231 [9]. Device code flow, on the other hand, is designed for input restrained devices and is therefore favourable for this thesis work, as the solution is meant to be used by restrained clients, such as Command Line Interface (CLI) applications, that would otherwise not be able to handle the OAuth 2.0 authorization.

User logout is handled through standard OAuth 2.0 token revocation. A refresh token is revoked in order to prevent more access tokens from being issued. A detailed description of the token revocation can be found in the appendix, under Section A.4. Observe that Single Sign-Out is not used, as logout in the OACS refers to ending the session with a device and not completely invalidating the users session with the identity provider.

4.3.1 OACS Resources

In this section, we describe the resources that OACS consists of and their roles. Here, resources refer to the entities, data structures, and documents required to build this proof of concept.

OIDC Provider:

A OpenID Connect 1.0 certified service provider.

liboacs:

A C library used by the gateway to handle authentication of users and authorization of profiles. Also used by services to verify access rights and by external services to handle communication with OACS Gateway.

OACS Gateway:

A D-Bus service provides user authentication, profile authorization, and service tickets. Does not have any logic of It is own but uses the gateway header found in the OACS library to perform the tasks.

OACS Conf:

The configuration file used by OACS Gateway to connect to the OIDC Provider and verify the access token.

OACS Gateway key:

A temporary symmetric key generated on runtime to sign service tickets.

Service Ticket:

Data structures required by the gateway to authorize a task. It is a base64 encoded JWT. See Section 4.8 for a more detailed description.

Ticket Whitelist:

The ticket whitelist is a runtime register of tickets issued by the OACS Gateway. Tickets are registered by id in order to keep track of the authorized tickets and prevent reuse.

Task Object:

A JSON data structure that represents the unpacked service ticket after successful validation. This representation is used by services to determine if the action is permitted. See Section 4.9 for a more detailed description.

Legacy Profiles:

These are JSON files that represents the traditional user roles through OACS access profiles

OIDC Client Key:

A persistent private key used to sign tokens for OIDC client authentication. The public key is registered for the client by the OIDC provider.

RoT:

A cryptographic device, such as a TPM or a Secure Element (SE) to securely store the OIDC Client Key and sign client access tokens.

OACS Client:

A set of web APIs that are used to authenticate user and authorize service tickets.

Network Service:

Any network service with external API endpoints that can process requests.

Service:

Any service that provides functionality that can be expressed as described under Section 4.6.

4.3.2 liboacs

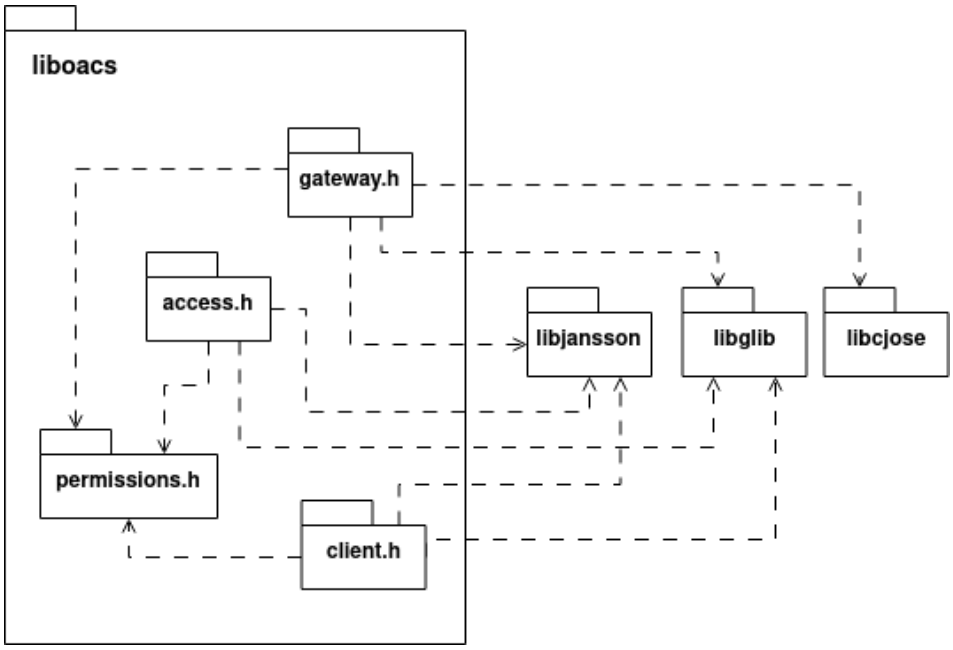

Figure 4.4: liboacs

The library liboacs uses a set of header files. The relationship between these headers is as visualized in Figure 4.4. In addition to the internal dependencies, there are also dependencies for external resources. Such as the "jansson" library that provides functionality for encoding, decoding, and manipulation of JSON data. The "cjose" library provides an implementation of JavaScript Object Signing and Encryption (JOSE) and the "glib" library that is a portable general-purpose utility library with many useful data types and functionality.

- **gateway.h:** Has functions to initiate the internal OIDC Client and set parameters necessary for verification of access tokens. The gateway functions uses OAuth flows through HTTP API calls to handle user authentication and profile authorization. Additional functions exist for verification of access tokens, generation of service ticket and verification of service tickets. See Figure A.1 for a more detailed description of the header.
- **client.h:** An abstraction layer to handle communication with the gateway service. Defines a communication handler function pointer, that is used by

the functions in "client.h" to relay requests to the gateway. The client functions crafts request objects that are then serialized for transfer over any inter process communication protocol using a corresponding communication handler function. See Figure A.2 for a more detailed description of the header.

- **access.h:** Used by services to verify access rights using the Task Object and predefined authorization requirements. See Section 4.11 for more details. See Figure A.3 for a more detailed description of the header.
- **permissions.h:** Definition of the permission type and permission flags used by "gateway.h", "client.h" and "access.h". Services will use these flags to define authorization requirements for their APIs. See Figure A.4 for a more detailed description of the header.

4.4 Data Structures

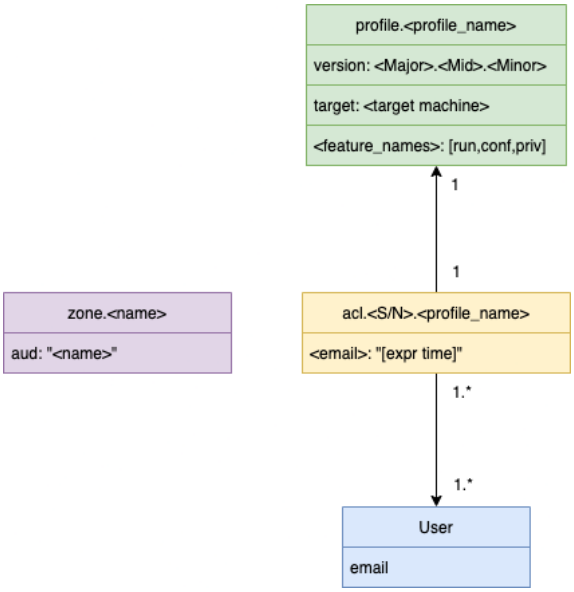

Figure 4.5: Data Structures

The entities in Figure 4.5 are represented by scopes with static claim mappings, in the OIDC provider. They are further defined below.

4.4.1 Profile

A profile is an abstract set of features that the device can perform together with one or more permitted actions assigned to each. These features can be native features provided directly by a service, pseudo features defined by a specific use of another native feature or composite features that are a combination of two or more features.

A native feature in this case could be to play an audio resource in a network speaker or start recording video in a security camera. While a pseudo feature can be for that network speaker to play the sound of a fire siren as a fire_alarm feature. No combination of features should occur. If a procedure would require the use of two or more features, then a new composite feature should be defined. This way, the service can allow access to single features without the concern of all the possible combinations. The profile will also contain the target OACS version. See version management below.

Each feature will have one or more of the following permissions, "run" and

"conf". This permission expresses the action of using the feature of configuring it. The association between a feature and action is called a task. Tasks are authorized to users through profiles. If the user sends a request to perform a task, then the request must include the run permission to use the feature, while "conf" is used to configure it. The "priv" flag can be used in combination with "run" and "conf" for elevated privilege requirements, such as privileged executions or configurations. The permissions are represented as a JSON list of strings.

An optional "target" field exists in the profile to determine the target platform of the profile. As different features exist for cameras and speakers. Or the same feature could have different uses depending on the target platform. Because of this ambiguity, the target field should be used in order to prevent malicious misuse, as a feature on a speaker could potentially result in elevated access on a camera. If features are deemed platform specific or unique enough, then this field can be opt out.

The profile abstraction was chosen as a way of re-visioning access control from the consumers perspective. Access control is traditionally a machine-centered process where the system uses a strict set of highly technical, predefined rules, with limited exposure to the consumer. However, as the system opens up and the consumer becomes a central part of it, the needs of the person must be satisfied. The high level of abstractions described in this section allow for the consumer to fulfill their access control needs without knowledge of the inner workings of the system.

4.4.2 Zones

The zones are definitions regarding a group of devices. The zones utilizes the "aud" claim defined by OAuth 2.0 to target a specific audience. The audience can be a specific application or service when used with cloud services. In our case, the audience claim refers to a zone defined by the system admin. The "aud" option is added to the configuration when setting up the device. The system admin can create a new zone by creating a new zone scope and then configuring a set of devices to accept that zone as the intended audience. When a user then logs in on a device in a particular zone, the device will insert its zone into the authorization request as a scope. After a successful authentication, a refresh token will be authorized with a profile intended for that audience. This refresh token will only be accepted in that zone, isolating different zone from each other.

4.4.3 ACLs

An ACL is included in the access token as a scope, that can be used by the device to determine the profiles a user is authorized to use on that particular device. Defining the ACL as a scope enables for a convenient way of managing and customizing existing profiles as well as creating new ones, at a central server without involving the device itself. To add a profile to a device, such a scope is created using the device's serial number and the profile scopes name, in order to bind the

two of them together.

The entries in these scopes are the "email" of the users, which correspond to the email claim in the default email scope, acting as the user identify, paired with expiration date-times for access to the profile. This way, an elevated access profile, such as "technician" can be assigned for a few hours to days and be inaccessible for the user when expired. Or an empty string can be used for persistent access to the profile. If the wildcard user "*" is used in a profile, any user can use that profile on that device without being explicitly assigned. In that case, any user added to the list will instead get black-listed.

An ACL can also be defined for a zone, in order to assign a profile and authorize users for a group of devices, instead of configuring individual ACLs. For instance a "m_building" zone can be defined to include all the devices in that building. The devices would then be configured with "m_building" as audience during setup. The ACL would be bound to a profile that would then either individually authorize users for all the devices on that building or everyone using the wildcard user.

4.4.4 User

The user is just the regular OIDC user. The only needed entry here is the email claim in the email scope.

4.5 Version Management

The OACS version is used to assess the compatibility of the defined profile, and the feature set device services have been developed for. The version number consists of three parts. A major version, a mid version, and a minor version. The minor version defines bug fixes or performance improvements that do not impact the features. In this case, a profile defined for the same major and mid version as the service will be accepted.

The mid version will define constructive changes. A larger mid version means that new features have been added. Thus, if the profile is defined for a higher mid version, then the service will potentially use features not supported by the service with a lesser mid version, given that the major versions match. The OACS library will then generate and log a warning for debugging purposes. The service could define a white list of mid versions if the feature set used by the service overlaps several mid versions in order to silence unnecessary warnings and false negatives. If the mid version of the profile is less than what's used by the target service, no warnings are generated as the profile uses a subset of the features that are supported by the service. Mid versions should only include constructive changes without removing functionality of features from a previous set of the same major version, for backwards compatibility purposes.

Major versions define breaking changes. It can be a destructive feature change that removes or mutates the behavior of a feature or other breaking changes to how the access control system behaves. The major version should not change if neither of these conditions is met.

4.6 Feature Definitions

Feature sets should be defined for each major-mid version combination, defining the expected behavior of the features. The services should be developed based on these definitions. The developer of a user-agent can then use this API documentation to configure their own services with the required permissions. The external services targeting users should further include these features in their documentation with a target OACS version specification, in order to specify which features are required for API use. As the user interacts with a user-agent, the developer of the user-agent should provide documentation for which features are required for their services. So that the IT admin of an organization can tailor custom profiles.

For instance, a pseudo feature, "fire_alarm", can be defined as mentioned previously. The feature set would include the expected behavior of the feature. Then the developers of the external service would create an API to use this feature, together with documentation describing the required permissions. In this particular case, the API could be called with a "trigger" option requiring "run" permission in order to trigger the "fire_alarm". While the "priv" permission is required to use the "disable" option to disable the alarm. The developer of a user-agent, such as a mobile application, could then provide a service to trigger the fire_alarm using the phone. And would implement the API request with the required feature and permissions according to the documentation. The admin of the organization using these services could then add the required feature and permission to use this user-agent, into an existing profile or create a new one. In this case, the admin chooses to create a "fire_alarm" profile with "fire_alarm" feature and "run" permission, assigned to the wildcard user "*" of dedicated devices. The "conf" and "priv" permissions are added to the admin profile. Thus any user can use the mobile application to trigger the alarm, but only the admin can disable it. The proof of concept will be based on this scenario.

4.7 Example of Use

In this section we give three use-case examples.

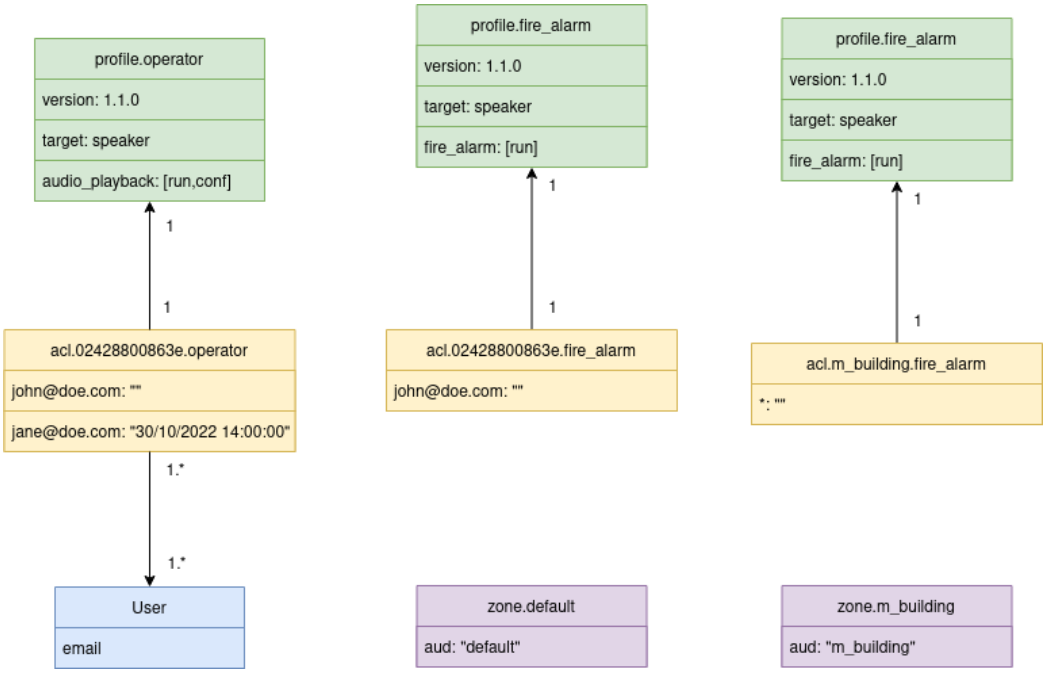

Figure 4.6: Example of Use

The first example demonstrates an operator profile defined after the OACS 1.1.0 feature set definitions (this is just an arbitrarily version) targeting speakers, with an audio_playback feature set to "run" and "conf" permissions. This profile can thus be used to run and or configure an audio playback task. It's however unauthorized to perform privileged elevated tasks, as the "priv" permission is missing. An ACL is also present for this profile defined on a device with "02428800863e" as serial number. A user with email address "john@doe.com" has been given access using an empty string, which means persistent access, while the user "jane@doe.com" has a temporary access until 30th of November. This profile is thus explicitly assigned to that particular device. A default zone is defined and can be used by devices that are configured to accept "default" as the intended audience.

The second example demonstrates a fire alarm profile with the "fire_alarm" pseudo feature set to "run" permission. Together with a ACL for the profile on a device with "02428800863e" as serial number. This profile ACL assigns a user with email address "john@doe.com", persistent access to the "fire_alarm" profile. This user can now use the "fire_alarm" profile to trigger the fire alarm. This example is demonstrated by the proof of concept demo.

The third example is another approach to the second one. In this case, the "fire_alarm" profile is assigned to the entire "m_building" zone without creating an explicit ACL for each device. Together with the wildcard user, any authorized user can start the fire alarm on any device in that zone.

4.8 Service Ticket

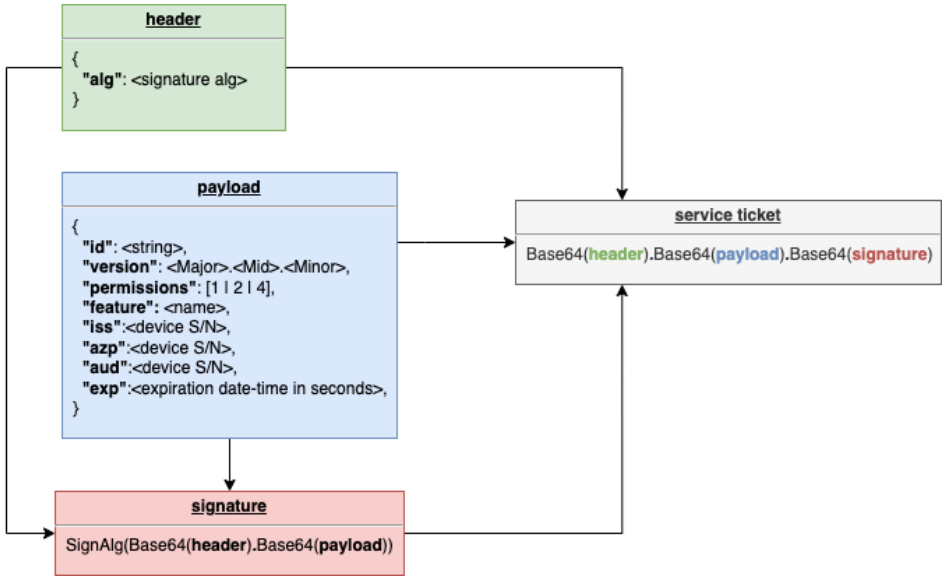

Figure 4.7: Service Ticket

A service ticket is an authorization by the OACS Gateway (see Section 4.10) to perform a task given that the user has the right permissions. The "id" key is a string used to identify the ticket and make sure that the ticket is used once. The "version" key is used to assess the compatibility of the defined profile with device services. "feature" is the request feature defined in the target feature set. The "permissions" key is used to determine the authorized permission for the requested feature. If the user is authorized to perform the task, a new short-lived service ticket is generated, as shown in the chart above.
The ticket will be bound by the serial number of the device that authorized the use, given by the "iss", "azp" and "aud" claims together with an expiration time. The ticket is then signed by the device and added to a ticket whitelist, binding it and limiting its use to a single feature on a single device.

4.9 Task Object

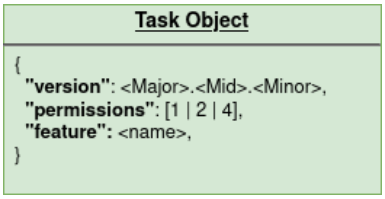

Figure 4.8: Task Object Structure

In Figure 4.8, a task object is utilized to represent the unpacked ticket, which is produced once a ticket is processed and approved. This simplified form of the ticket is utilized by services for access control purposes and is considered to be an unprotected data structure. The transformation of the service ticket to the task object signifies the transition to a trusted environment, wherein the system is entrusted with the responsibility of maintaining the integrity of the task object.

4.10 OACS Gateway

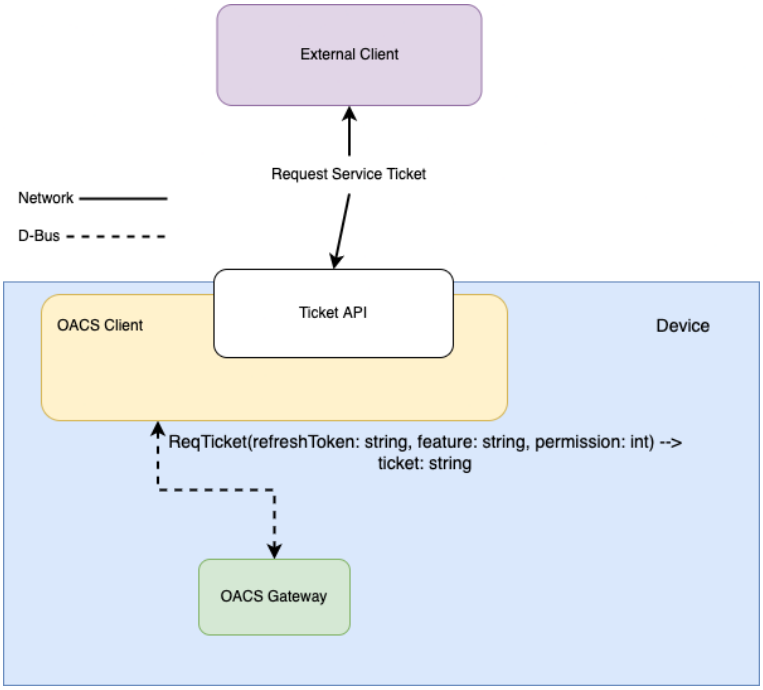

Figure 4.9: Request Service Ticket

The OACS Gateway is a systemd service that starts at system initialization and acts as a D-Bus server. At initialization, the service will read the configuration file (see Section 4.10.1) and generate a temporally symmetric key. This key is used to bind a service ticket to a specific device, limiting the use of an authorized ticket to the device that authorized it. As the keys are randomly generated on each device, a compromised key won't affect other devices. The key is a 256 bit secret that is updated once every hour. This is to ensure perfect forward secrecy. A more frequent update rate could increase the chances of a race condition due to use of tickets generated before the key update.

The user will be able to use the external Ticket API to request a service ticket to use a device feature with authorized permissions, see Section A.3 for a more detailed explanation of the authorization process of a ticket request. A process belonging to the "oacs" group will be allowed to request the ticket from OACS Gateway, on behalf of the user, with the user's refresh token, the requested feature, and permissions for that requested feature. The refresh token will be used to request a new access token with the authorized access profile. The access rights to a profile will be verified by confirming that the user exists in the access control list for the authorized profile, on the target device or the zone that the device belongs to. If the user is authorized to use the requested feature with the expected

permissions, a new service ticket will be issued by the device signed using the symmetric key of OACS Gateway together with a new refresh token using refresh token rotation.

4.10.1 OACS Gateway Configuration

```
{
  "iss": "https://server/realms/axis",
  "aud": "default",
  "azp": "oacs",
  "client_key": "<a key handle or other kind of id>",
  "provider_ca": "/etc/oacs/oidc/ca/provider_ca.pem"
}
```

Listing 1: OACS Gateway configuration

This Listing 1 is the structure of the OACS configuration file. The device admin configures these fields. The "iss" option is the issuer URL pointing to the OIDC provider and the target realm. The "aud" option refers to the intended zone, and "azp" is the authorized party (client id). The "provider_ca" is the CA certificate of the server. These options are necessary to ensure the integrity of the access token. The "client_key" is a reference to the private key in the RoT device. It is loaded and configured by the device admin.

4.11 Ticket Use

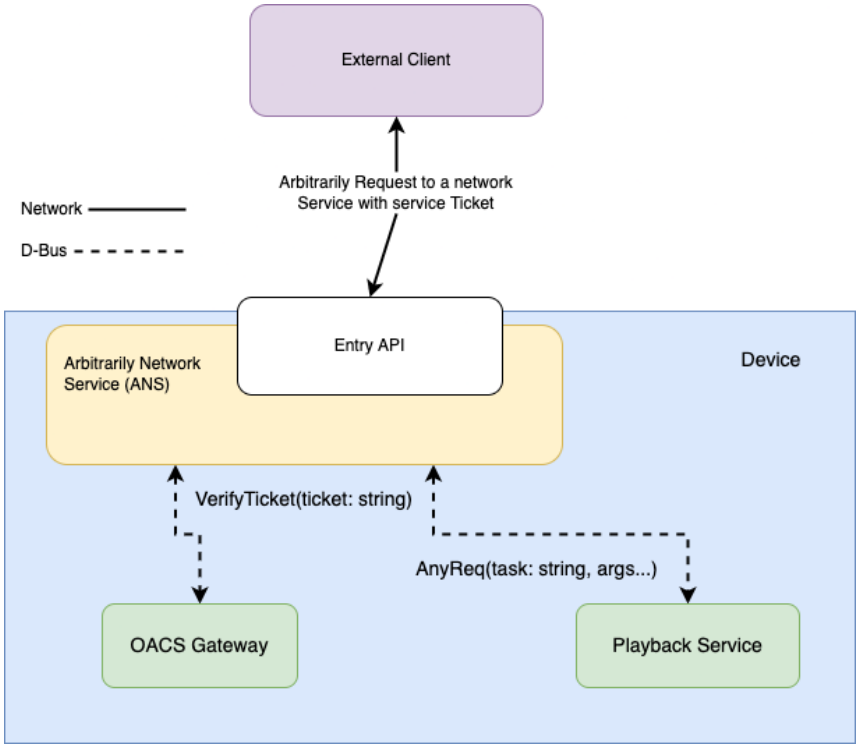

Figure 4.10: Access Authorization Service

As it is shown in Figure 4.10, the process of using a ticket starts when an external service API is called. The service ticket is passed with the request. Once the single use service ticket is validated by the OACS Gateway, it's unpacked to a task object. The task object is forwarded to the next service as a string. Each service does a permission check on the task object, at the beginning of a serving API call, expecting one of a set of approved features and permissions required to use that API. Every OACS feature set version is checked according to Section 4.5.

An entry service can confirm the validity of the ticket by sending it to OACS Gateway over D-Bus. The integrity of the ticket is first confirmed by checking the ticket whitelist after its ticket id. Tickets that are not previously whitelisted by the OACS Gateway are considered unauthorized and immediately invalidated, under the suspicions of ticket reuse. If the ticket is whitelisted, the signature of the token and the expiration time is verified. If the ticket is valid, the gateway will unpack the ticket to a task object and return it with an confirmation of validity. If the ticket is invalid, the authorization will fail and the OACS Gateway will return an error, terminating the API call. If the task object has the right permissions, then the API call will resume, and the task object will be passed to the next service. Otherwise, the API will break out with an error.

4.11.1 Task Authorization (Legacy)

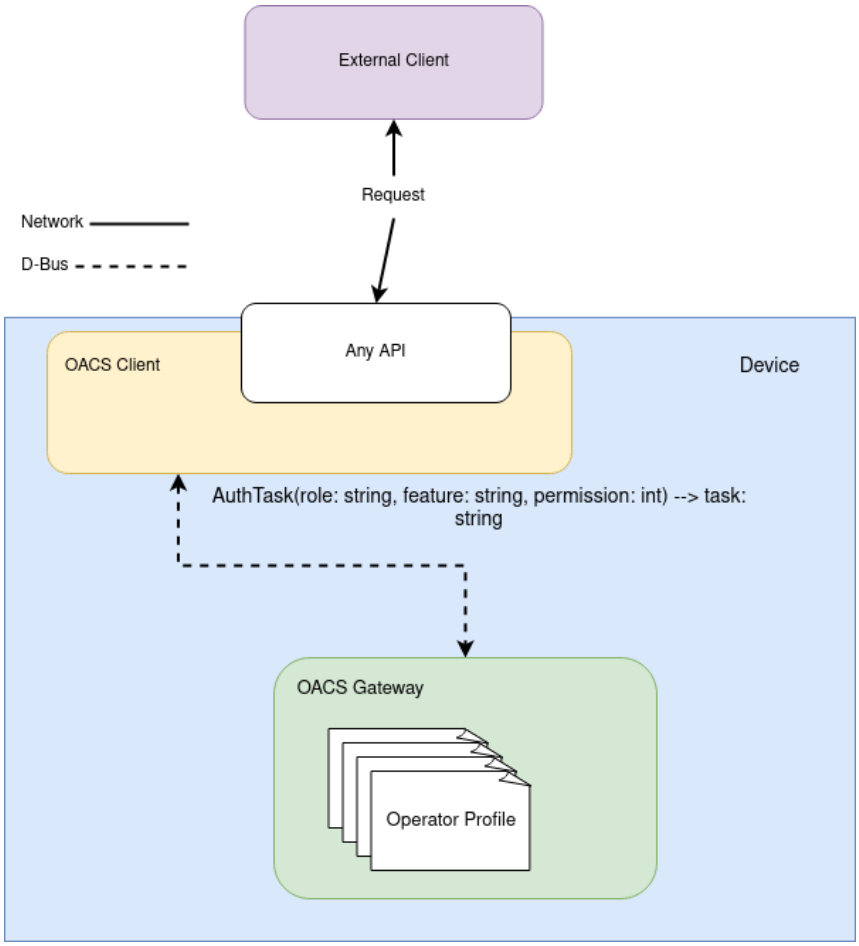

Figure 4.11: Task Authorization

The OACS requires changes to the existing clients in order to handle profiles and service tickets. A OIDC provider is also required, which is not suitable for all environments. The task authorization exists to provide backwards compatibility with existing ecosystems. This will still require some changes to the CGI endpoints to make them aware of the feature based access control system. OACS Gateway is provided with static legacy profiles. These are JSON files that represents the traditional user roles through OACS access profiles.

The user is authenticated through existing means, such as HTTP Basic and Digest authentication mechanisms. A representation of the user credentials exist on the device. Once the user is authenticated and user roles have been determined, the OACS Gateway can be used to authorize a task object.

The request will have to be evaluated at the CGI endpoint in order to deter-

mine the required feature and permission. A task object can then be authorized by requesting authorization for the feature and its permission using the established roles. If a legacy profile exists for the role and the task is authorized by it, a task object is generated and returned. If none of the roles assigned to the user permits the requested task, authorization fails.

Chapter 5
Discussion

In this chapter, the challenges encountered during the work of this thesis work will be discussed. Additionally, limitations that may have impacted the research outcome will be discussed as well. Furthermore, the security evaluation of the system will be expounded upon using the attack tree analysis methodology. The design of the system will also be assessed from a security perspective, highlighting potential vulnerabilities and strengths. Subsequently, the advantages and disadvantages of the proposed solution will be analyzed. Finally, suggestions for future research endeavors will be proposed.

The proposed solution provided a definition of the mechanism underlying the OACS system. This mechanism is primarily composed of two main components, namely the functions assigned to the services, and the access permissions granted to each user through profiles. Additionally, the user-device and the device-profile relationship play a key role in enabling the assignment of employees to specific permissions on designated devices.

Furthermore, the proposed solution underlines that the process of assigning permissions has become exceptionally specific, thanks to the ability to pinpoint target devices and grant specific types of permissions, such as the authorization to run or configure a specific service functionality, with or without elevated privileges as defined under Section 4.4.1.

It is noteworthy that access rights are not allocated directly to individual users; rather, they are defined within profiles, which are then assigned to users. As a result, ensuring that access rights are accurately and appropriately granted to individuals has become a matter of assigning the correct profiles to the appropriate users, alongside the opportunity to create new profiles as the need arises. In order to achieve the desired goal of these access rights, service functionalities, so called features were paired with two types of actions expressed through permissions, as explained in Section 4.4.1. The "run" permission allows the feature to be used, the conf (configuration) permission allows for the feature to be configured, and the priv flag added together with one or both of these permission allows for elevated privileges when a task is consumed.

The utilization of a locally deployed OpenID Connect 1.0 certified provider enhanced the convenience of development and experimentation to a significant

extent.

The ability to construct and execute a virtual machine image based on open embedded core facilitated the development of a proof of concept without being constrained by the proprietary technologies and solutions employed by theAxis OS. As a result, we were able to realize the required modifications without consideration for their effect on the remainder of the system. This approach was implemented as a meta-layer, which was readily portable into and usable by theAxis OS.

5.1 Limitations

No additional features aside from what's defined within the scope of OIDC 1.0 were utilized. As this solution aimed to be compatible with all OIDC providers, the system, therefore, did not used a server-side access solution through the Keycloak IAM. Access control to the ACL, profile and zone scopes had to be moved to application level.

The proposed solution relies exclusively on the information and data expressed within the access profiles and ACLs to deduce the privileges that have been delegated to the users. Therefore, systems' admins must prepare these profiles in the form shown in the design chapter. The main way to do this job is the traditional method, i.e., manually creating it and storing it on the provider side in a way that suits the design of the provider, knowing that there may be automatic ways or tools to do this task and may help reduce a large amount of work required for the creation of all the profiles in the system. Still, such automation tools arenot discussed or mentioned in the project because it is outside the scope of the study.

The profile access control lists (ACLs) of a device comprises a list of users who have been granted access to the device, as expressed by the access profile. Users are identified through their email addresses in the ACLs. This is due to administrative advantages. As email addresses are unique and predictable. The ACLs can be generated even before the user account is created. Removing a hard dependency on the user entity. Unlike using abstract identities such as Universal Unique IDs (UUIDs) [22], emails are easier for the human administrator to manage.

The user's service ticket is converted by the OACS Gateway to a task object on entry, with the name of the associated feature in addition to the permissions granted to the user. See Section 4.9. At this stage, it was necessary to choose between two different designs to determine how the ticket would be dealt with. The first choice was to maintain the integrity of the service ticket between services, enforcing integrity controls at each step, and the second was to reduce the overhead and assume the system as a trusted environment in which a unprotected task object can flow within. The present design is primarily aimed at embedded systems with limited resources. As a result, it is imperative to efficiently manage the available resources.

In order to ensure the integrity of the service ticket, the signature control will

always provide a high level of trust within the system. However, this procedure places an increased computational load on the device's resources, especially on the CPU. The ticket could be verified through the use of public-private key pair. Where the ticket is signed by the OACS Gateway during authorization and then validated by each service using the associated private key. There are two issues with this approach. The first one is the computation overhead. Asymmetric encryption has significantly higher computational load than the symmetric approach proposed in this paper. The other issue is with the decentralised nature of public key infrastructure. If the system is deemed untrustworthy, then an attacker could forge a service ticket and then change the public key used for signature validation. Thus, rendering this approach unreasonable. Either way, trust in the system needs to exist for the solution to work.

Another way would be to keep the symmetric key as proposed by this paper, where the key is kept secure within the process of the OACS Gateway service. Each service would then be required to ask the OACS Gateway for ticket validation, where OACS Gateway acts as a trusted third party while the communication is protected by the D-Bus session. However, this approach introduces overhead in the form of D-Bus method calls which are deemed inappropriate by the supervising engineer at Axis Communications.

Hence, it was necessary to opt for the second solution, which assumes that the environment where the object will reside is reliable, specifically between services, and that it is responsible for maintaining the object's integrity [35].

Clients are being installed and configured inside the device instead of installing one on a separate server within the local network. All devices contain the same client, that is, the same key and the same client identity. Doing so, removed the dependency of a third-party computer and the requirement network setup. Utilization of a third party API back-end to handle interactions with the OIDC provider would create a single point of failure. Server downtime would cause all the devices to be unusable. Such a server would also become a bottleneck on the network, potentially causing congestion and scale-ability issues. On the other hand, the use of a OIDC client in all devices may result in increased overhead. It requires the capability of running and installing the OIDC client in the devices containing the client. Therefore, the system may require high-cost devices to run efficiently.

5.2 The security evaluation

The objective of this section is to analyze and interpret the resulting findings. The section is structured in the following manner: we start with a presentation, examples of threat models are presented, followed by an evaluation of the design.

5.2.1 Threat modelling

This section aims to provide an account of potential cyber-attacks that have the chance to compromise our proposed solution, according to specific circumstances and conditions. In order to analyze the security of the system, the attack tree

methodology is utilized, see Figures 6.1 and 6.2. This approach is commonly used to describe a system's security by testing different types of attacks and structuring them in a tree graph format. The attack tree begins with a root that represents the ultimate goal of the attack, with various threats branching out from it that can lead to the achievement of this goal [7].

We start by describing potential security weaknesses in a system that could lead to preventing the user from having access control.

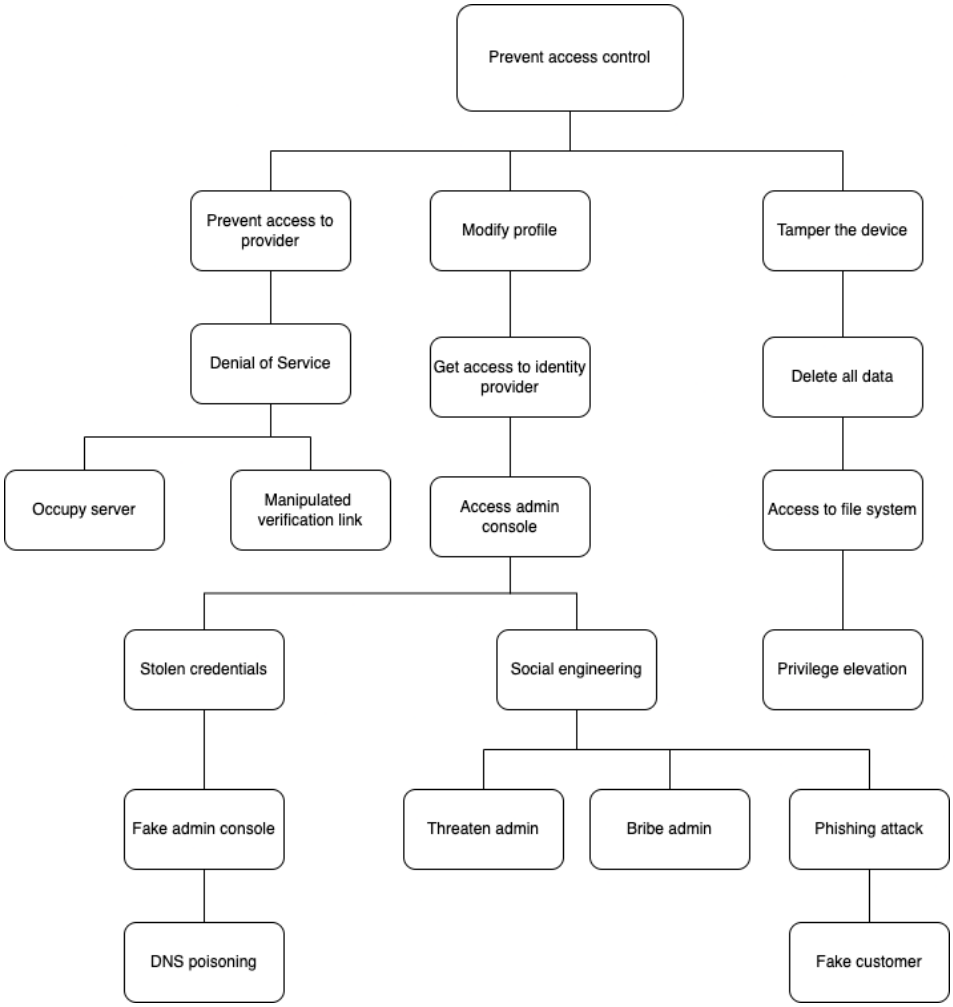

Figure 5.1: Attack Tree for Prevent Access Control

The following enumeration aims to identify the different ways in which each attack scenario depicted in the attack tree model (see Figure 5.1) could potentially undermine the user's access control:

1. **Occupy server:**
   An access control to the system can be hindered by a denial-of-service attack aimed at occupying the server resources and preventing access to the Identity Provider (IdP).
   To prevent this attack, the IdP must have a protection mechanism to detect and block malicious traffic coming from multiple sources.
2. **Manipulated verification link:**
   If a denial of server attack is being carried out to manipulate the verification link and prevent access to the IdP, it can lead to a prevention of access control to the system.
   To prevent this attack the solution in (Occupy server)is recommended and using a multi-factor authentication method to increase the security of the verification process.
3. **DNS poisoning:**
   If a fake admin console is created with stolen administrator credentials by running a DNS poisoning attack on the domain to make changes on profiles in the identity provider, access control to the system can be prevented.
   To prevent this attack, it is recommended to update and patch the DNS servers to prevent them from being vulnerable to attacks, and it is important to regularly monitor and audit administrator access and activity to detect any unauthorized access or changes to profiles.
4. **Threaten admin:**
   This attack can block access control to the system through a social engineering attack that threatens the administrator to take control of the admin console and make changes to the profiles in the identity provider.
   To prevent this attack, a security awareness training and testing can help ensure that employees remain vigilant and informed about the latest security threats and best practices.
5. **Bribe admin:**
   This attack can block access control to the system through the use of a social engineering attack to bribe the admin to take control over the admin console and make changes to profiles in the identity provider.
   To prevent this attack the same recommendation is needed in Threaten admin.
6. **Fake customer:**
   This attack can block access control to the system by using a phishing attack while imitating a customer to manipulate the system administrator into making changes to profiles in the identity provider.
   To prevent this attack the same recommendation is needed in Threaten admin.

7. **Privilege elevation:**

   This attack can block access control to the system if a privilege elevation is exploited to gain access to the file system and delete essential data through tampering with the device.

   In order to prevent this attack, it is recommended that the device has a robust encryption method to protect the important data. Additionally, our solution provides software security measures by restricting access privileges to specific tasks and only authorizing designated users to access them.

Defense mechanisms against such attacks or threats have not been implemented in the proof of concept due to time constraints and its non-essential nature to complete the proof. Nevertheless, it is strongly recommended to consider incorporating appropriate defense mechanisms to prevent such attacks or threats in future real implementations. Its aim is to highlight the weaknesses that accompany this solution.

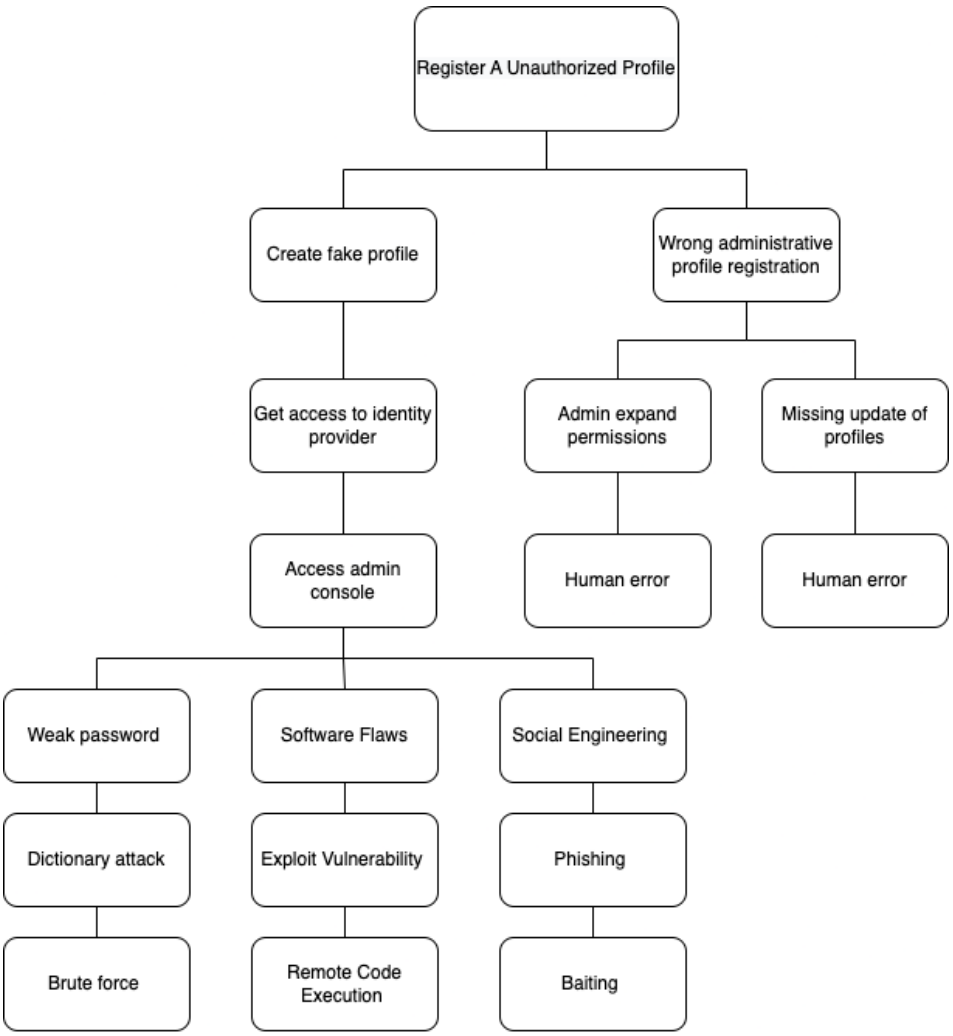

Figure 5.2: Attack Tree for Register Unauthorized Profile

The following enumeration aims to identify the different ways in which each attack scenario depicted in the attack tree(see Figure 5.2) model could potentially undermine the registration of an authorized profile:

1. **Human error:**

   This security weakness can result in the registration of an unauthorized profile if a human error expands the permissions incorrectly during the registration of a profile.

   Implementing proper training and awareness programs to educate users about the importance of accurate profile registration and the potential consequences of human errors can help to prevent this weakness.

2. **Brute Force :**

   This attack may result in the registration of an unauthorized profile, as a brute force attack, or similarly a dictionary attack can be used to break a weak password and gain access to the admin console.

   To prevent this attack, it is recommended to use strong passwords and enable multi-factor authentication on the admin console.

3. **Remote Code Execution:**

   This attack may result in the registration of an unauthorized profile. A remote code execution can occur by exploiting an existing vulnerability in the system to gain access to the admin console, leading to a failure to update the profiles and permissions, which results in incorrect registration of a profile.

   To prevent this attack, it is important to regularly update the system with the latest security patches and fixes to address any known vulnerabilities

4. **Baiting:**

   This attack can potentially result in the unauthorized registration of a profile. A baiting attack is executed through a phishing attempt that utilizes social engineering techniques to gain access to the admin console. This can lead to a failure to update the profiles and permissions, ultimately resulting in incorrect registration of a profile.

   The same recommendation in Human error weakness (number 1) can help to fix this vulnerability.

Defense mechanisms against the above attacks or threats have not been implemented in the proof of concept due to time constraints and its non-essential nature to complete the proof. Nevertheless, it is strongly recommended to consider incorporating appropriate defense mechanisms to prevent such attacks or threats in future real implementations. Its aim is to highlight the weaknesses that accompany this solution.

5.2.2 Design

This section aims to discuss how we solved some problems related to some dependencies like the Idp, OIDC client, signatures, and tokens.

Single OIDC Client

All devices equipped with the OACS gateway service utilize a singular OIDC client with an identical configuration and an asymmetric elliptic curve key (prime256v1). Such a mechanism may heighten the probability of key disclosure. To address this issue, our solution mandates that each device possesses a trusted platform root and stores the keys in a secure manner.

Single point of failure

Our solution depends on our IdP to authenticate the user in the login phase. Additionally, the authorization server it utilizes to request a refresh token and revoke the refresh token. The generation of service tickets is also dependent on the authorization server for the access tokens. The dependency on external entities leads to a single point of failure problem (SPOF), and it makes the system prone to provider downtime. Inaccessibility to any of these entities will prevent access to the provider will lead to system failure, such as a denial of service attack (DDoS).

Integrity of provider

Another problem related to the dependency on IdP is the integrity of the provider. A compromised server attack could take place on the provider side, which would compromise it's integrity. The integrity of the provider is crucial to the proposed solution.

Trust in the system

The secure environment inside the system is built on the trust between the gateway, system services, and the network services. The gateway acts as a validation zone that checks the validity of a token or ticket before letting it through. The remaining parts of the system need to be trusted with preserving the integrity of the task object.

Signatures

Two types of signatures are used in the system: the Elliptic Curve Digital Signature Algorithm (ECDSA) based ES256, algorithm using the secp256k1 curve defined in Standards for Efficient Cryptography (SEC), and the Hash-based Message Authentication Code using SHA-256 (HS256). ES256 signatures are used to sign the tokens from the provider and the client access token generated in OACS Gateway. ES256 signatures for current industry standard that is recommended as best practice. ECDSA provides the same level of security for much smaller

key sizes. This is an attractive feature in resources-constrained execution environments [16]. Unlike ES256, HS256 is a symmetric key-based algorithm. It's not possible for another entity to verify the signature without having prior knowledge of the secret. As the same secret is used to sign the token as to verify, this method is not suitable when signature needs to be verified elsewhere. As the secret needs to be shared and can be used to forge tokens. HS256 is used as a message authentication code between the gateway and the network services to verify the authenticity of service tickets. The secret is thus kept in one place in the system and not shared with any other party. Together with the temporary nature of the secret and the speed of the algorithm, HS256 was deemed suitable for this purpose without compromising on security. HS256 is recommended to be used alongside ES256 in order to provide a more robust authentication mechanism for the system, as each algorithm provides complementary security properties [18].

Leaked refresh token

The refresh token is one of the essential parts of the system, and it is used by the OACS gateway to request access tokens from the provider. In other words, it substitutes the session between the client and provider. This will allow unauthorized users to use the system by compromising the refresh token. Since the refresh token is an essential component of the solution, security best practices were applied according to RFC-6819 [23].

One of the best practices is token rotation. Refresh tokens should be rotated frequently to reduce the risk of them being stolen or compromised. This can be achieved by revoking and issuing new refresh tokens periodically.

Another best practice is token revocation. In case of a suspected or confirmed breach, it is important to have the ability to revoke refresh tokens immediately. This can help prevent further unauthorized access and mitigate the damage caused by a potential breach.

The system addresses this issue through the use of refresh token rotation, where a new refresh token is issued together with each access token that is retrieved seamlessly through refresh token flow. This mechanism provides automatic reuse detection and prevention, as the previous refresh tokens are invalidated. When a refresh token is reused, the provider can detect this transgression allowing for appropriate measures to take place.

5.3 Comparison to RBAC

The proposed solution and RBAC mechanism serve the same primary purpose. Which is to manage user access rights. Instead of RBAC, this report presents a dynamic solution where the system admin has the option to customize access profiles or create completely new ones as the need arises. RBAC on the other hand provides static access roles which limits administrative options.

One of the essential problems that OACS solves is the massive manual administrative work that is required when assigning user permissions. The proper role needs to be selected or created carefully to avoid elevated privileges, which adds to the complexity of the workflow. This phenomenon is called "role explosion" and is a consequence of complex access control requirements associated with today's dynamic workforce [6] [21]. A user will have to be assigned a more privileged role when the need arises, which can involve changes to several devices, as the roles need to be assigned on individual devices. This is not a sustainable solution nor is it always possible as the rest of the system needs to be modified accordingly. These roles would then have to be removed from the user, as it would go against the least privilege principle and give the user higher privilege than necessary. A left over role could also be exploited to gain higher privileges than what's appropriate at the moment.

OACS addresses the Role Explosion issue by defining user-manageable custom profiles. These profiles represent set of device features as defined in Section 4.4 that together with the "run" and "conf" permissions, alongside the optional "priv" flag, composes a task that can be allowed for a user on a device by association with the profile it is defined in. Thus, new profiles can be created after user requirements and assigned to devices or zones and the users authorized to use it for limited or unlimited time. The system admin can easily modify an access profile after necessity or create completely new profiles.

5.4 Advantages of the solution

5.4.1 Customized access profiles

Contrary to the RBAC system, that needs to create specific roles for users to access specific data, the solution customizes access rights for each user in a specific way. For example, a profile can be specified to only grant access to a single feature on a device, thereby restricting access control to device resources. Profiles such as the technician profile mentioned by scenario 3 in introduction. A customised profile can satisfy the users needs without over assigning them privileges as with RBAC. This eliminates the problem of role explosion (see Section 4.4) [8].

5.4.2 Temporary access to profiles.

According to the profile data structure (see Section 4.4), the profile contains an "exp" key that determines the lifetime for each user to use this profile, this makes permissions that are given to the user temporary if necessary. This ensures that old authorizations are forgotten in the system, which could otherwise lead to unauthorized permission for the same user that no longer needs this profile.
One more benefit for temporary access is that the stolen token may have a limited impact on the system's integrity.

5.4.3 Centralized authentication.

The users have no registered accounts on the devices, but they must verify their identity by validating their credentials against the IdP, a service that provides user authentication [1]. In this case, the user must complete an authentication process against the IdP to obtain a refresh token.

5.4.4 Single sign on

The proposed solution employs the use of Single Sign-On(SSO) [28], which is a term commonly used to describe a scenario where a user utilizes identical login credentials to access various domains or devices. To authenticate with the system, the user must login once using the verification link provided on one of the devices. Subsequently, upon obtaining a valid refresh token, the user gains access to all devices registered on the system. However, it should be noted that the user's profile must first be authorized through Access Control Lists (ACLs), as previously explained.

5.4.5 Task Authorization

By only granting access to specific tasks through temporary, single-use, service tickets, the risk of unauthorized access to sensitive data or functionality is reduced. Single-task authorization allows for fine-grained control over who can perform actions which makes it easier to delegate tasks and manage access rights. Single-task authorization makes it easier to track and audit who and when performed a specific action, which can be useful for compliance and incident response. By granting access to specific tasks, the user interface can be tailored to the user's rights, making it more user-friendly and less cluttered. Single-task authorization allows for more efficient use of resources and cost reduction by only providing access to what is needed. Better risk management, by limiting the access to certain actions, the risk of malicious actions can be reduced, which leads to better risk management.

5.4.6 Flexible privileges

Admin can assign necessary privileges through profile. New features can always be added on the run, or new profiles with additional features can be created and

injected into the system.

5.5 Disadvantages of the solution.

5.5.1 Network connection

Our system requires a network connection at all times. The system needs to be connected to the external entities to request the necessary data, such as user authentication, access tokens or refresh tokens. If the connection is lost for any reason, the system will no longer be available to the user, but the devices will keep functioning locally on their own.

5.5.2 Manual Register

The other access control systems that are in use today, including the updated RBAC model, require significant administrative work, such as registering users, organizing them into groups, and granting the necessary permissions to each user. This work involves a lot of time and effort on the part of the access administrator [34] [36]. The solution also requires this kind of administrative work because the IdP must contain user accounts and their data, profiles needs to be created and each user must be manually associated with profiles through ACLs. However, this is an initial setup work. The most significant cost is the generation of the access profiles. However, it can be easily automated using tools and predefined templates. Once created, these profiles are available on demand and don't require any additional work on the device side. User-device-profile bindings can also be simplified through the use of tools that conveniently visualise the relationships. Credential administration is also improved through the use of single global identities instead of localized instances on each device. The resulting system scales better than the other localized systems, due to the centralised nature of the data.

5.5.3 One Client

All devices are OIDC clients and need to be configured with the necessary information and a client key. This configuration creates significant labour especially with in a large network with significant amount of devices. The same client ID and client key are used in all devices, which creates a potential opportunity for key leaks to the public.

5.5.4 Synchronization Problem

The proof of concept revealed that refresh token rotation can cause race conditions in the system, when parallel connections are used to request service tickets. When more then one request arrive to the servers request queue, only the first

one will be served, as the refresh token is updated, the other requests will be invalidated.

One simple solution to this problem is to synchronize all requests and wait for consequent responses. However, this could have severe performance issues.

The other more reliable solution is to reuse the access token until it expires. When requesting a service ticket, the user agent would first check if the access token has expired. If expired, it will halt all requests until a new access token can be retrieved from the device, using the refresh token. The new access token would then be used against the device, instead of the refresh token, to authorize service tickets. As the access token isn't rotated like the refresh token, parallel connections won't be an issue. It would also reduce the overhead imposed by constantly requesting a new access token from the provider over the internet. The network round-trip time is the most significant time cost in access control process. The short lived nature of the access token is also suitable, as the time-frame of malicious use is smaller, in the case of a leaked access token. As the refresh token is rotated when the access token expires, refresh token reuse is still detectable. The proposed modification would however open up for a time-frame equal to the life-time of the access token, during which a refresh token can be reused without any knowledge. The time-frame can be shrunk further by probing the OAuth 2.0 introspection endpoint for the validity of the refresh token in shorter intervals, in order to respond earlier to a refresh token reuse.

5.6 Possibilities and Future work

The system is based on an idea that is still in the early stages of development, and there are numerous prospects for future improvement. One area for further development could be the authentication mechanism used to enhance the trust and security of the data provider's login process. A possible approach for achieving this could be the implementation of a two-factor authentication protocol, which would provide an additional layer of security and confirm the user's identity in a more robust manner.

It is also possible to research or develop tools that help to solve the problem of manual registration of user data and profiles. These tools will greatly help reduce the burden on admins and will contribute to reducing the area of human error when registration of data.

5.7 Reflection on This Work

In this project, we set out to answer some interesting questions and find a solution to the problem presented by Axis company. We aimed to improve upon the existing access control system based on roles by making privileged access easier

and more flexible.

Our approach involved collecting data from peer-reviewed sources and testing various solutions to arrive at the best possible outcome. We received valuable feedback from our supervisors during the iterative process, which helped us improve the solution.

Upon reflection, we have identified both advantages and disadvantages of the solution. On the one hand, the solution provides a good approach to distributing roles in the system, with the added benefits of using OIDC protocol for customized access control, temporary access to resources, centralized authorization, and flexibility of distributing privileges. On the other hand, there are some limitations to the solution, such as having only one client for all devices and no offline mode for the system.

Overall, we believe that this project has helped us to develop our critical thinking and analytical skills, and we have learned a great deal about access control systems and their applications. We also recognize the potential for further research in this area, particularly in addressing the limitations of the current solution.

When we look back at the process of developing and submitting a patent application for the solution presented in this thesis, we realize how important it is to protect our intellectual property. Our solution is a new and innovative way to solve the problem we set out to address, and the patent application we submitted to the European Patent Office with the help of an Axis patent engineer is a way to protect the unique aspects of our solution.

Looking back on this experience, we are grateful for the chance to engage in the innovation process and develop skills in research, entrepreneurship, and intellectual property protection. We are excited to see how the patent application for our solution will contribute to ongoing research and development in the field, and how it could potentially have a positive impact on society.

Bibliography

[1] amazon.com. *Creating OpenID Connect (OIDC) identity providers*. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers_create_oidc.html. (accessed: 15.09.2022).

[2] auth0.com. *JWT.IO - JSON Web Tokens Introduction*. URL: https://jwt.io/introduction. (accessed: 14.07.2022).

[3] B. Campbell et al. *OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens*. RFC 8705. RFC Editor, Feb. 2020.

[4] W. Denniss et al. *OAuth 2.0 Device Authorization Grant*. RFC 8628. RFC Editor, Aug. 2019.

[5] Docker. *Docker overview*. URL: https://docs.docker.com/get-started/overview/. (accessed: 6.10.2022).

[6] A. Elliott and S. Knight. "Role Explosion: Acknowledging the Problem." In: *Software Engineering research and practice*. Citeseer. 2010, pp. 349–355.

[7] G. Falco, A. Viswanathan, and A. Santangelo. "Cubesat security attack tree analysis". In: *2021 IEEE 8th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*. IEEE. 2021, pp. 68–76.

[8] A. Fatima et al. "Towards Attribute-Centric Access Control: an ABAC versus RBAC argument". In: *Security and Communication Networks* 9.16 (2016), pp. 3152–3166. (accessed: 15.09.2022).

[9] Roy T. Fielding and Julian Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231. June 2014. DOI: 10.17487/RFC7231. URL: https://www.rfc-editor.org/info/rfc7231.

[10] The Apache Software Foundation. *The Apache Software Foundation celebrates 15 years of open source innovation and community leadership*. Nov. 2014. URL: https://www.globenewswire.com/news-release/2014/11/19/684497/10108887/en/The-Apache-Software-Foundation-Celebrates-15-Years-of-Open-Source-Innovation-and-Community-Leadership.html.

[11] Robert P. Goldberg. "Survey of virtual machine research". In: *Computer* 7.6 (1974), pp. 34–45. DOI: 10.1109/MC.1974.6323581.

[12] D. Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. http://www.rfc-editor.org/rfc/rfc6749.txt. RFC Editor, Oct. 2012. URL: http://www.rfc-editor.org/rfc/rfc6749.txt.

[13] D. Hardt, A. Parecki, and T. Lodderstedt. *The OAuth 2.1 Authorization Framework*. URL: https://www.ietf.org/archive/id/draft-ietf-oauth-v2-1-04.html. (accessed: 2.10.2022).

[14] P. Havoc et al. *D-Bus Specification*. Tech. rep. freedesktop_org, Revision 0.40 2022-10-05.

[15] W Jansen. "A Revised Model for Role-Based Access Control". In: *NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, Gaithersburg, MD* 5 (1998), pp. 171–181. DOI: https://doi.org/10.6028/NIST.IR.6192. (accessed: 01.06.2022).

[16] M. Jones. *JSON Web Algorithms (JWA)*. RFC 7518. RFC Editor, May 2015.

[17] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Token (JWT)*. RFC 7519. http://www.rfc-editor.org/rfc/rfc7519.txt. RFC Editor, May 2015. URL: http://www.rfc-editor.org/rfc/rfc7519.txt.

[18] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Token (JWT)*. RFC 7519. http://www.rfc-editor.org/rfc/rfc7519.txt. RFC Editor, May 2015. URL: http://www.rfc-editor.org/rfc/rfc7519.txt.

[19] T. Kaskinen. *Introduction to D-Bus*. URL: https://www.freedesktop.org/wiki/IntroductionToDBus/. (accessed: 13.07.2022).

[20] keycloak_org. *Server Administration Guide*. URL: https://www.keycloak.org/docs/latest/server_admin/. (accessed: 06.10.2022).

[21] D Richard Kuhn, Edward J Coyne, Timothy R Weil, et al. "Adding attributes to role-based access control". In: *Computer* 43.6 (2010), pp. 79–81. (accessed: 15.09.2022).

[22] Paul J. Leach, Michael Mealling, and Rich Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. RFC 4122. http://www.rfc-editor.org/rfc/rfc4122.txt. RFC Editor, July 2005. URL: http://www.rfc-editor.org/rfc/rfc4122.txt.

[23] T. Lodderstedt, M. McGloin, and P. Hunt. *OAuth 2.0 Threat Model and Security Considerations*. RFC 6819. RFC Editor, Jan. 2013.

[24] T. Lodderstedt et al. *OAuth 2.0 Security Best Current Practice*. Internet-Draft draft-ietf-oauth-security-topics-21. Work in Progress. Internet Engineering Task Force, Sept. 2022. 56 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-oauth-security-topics/21/.

[25] M. Zerkouk; A. M'Hamed; B. Messabih. "A User Profile Based Access Control Model and Architecture". In: *International journal of Computer Networks Communications* 5 (2013-1-31), pp. 171–181. DOI: 10.5121/ijcnc.2013.5112. (accessed: 01.06.2022).

[26] S. Namasudra, S. Nath, and A. Majumder. "Profile based access control model in cloud computing environment". In: *2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE)*. 2014, pp. 1–5. DOI: 10.1109/ICGCCEE.2014.6921420. (accessed: 01.06.2022).

[27] openid.net. *Welcome to OpenID Connect*. URL: https://openid.net/specs/openid-connect-core-1_0.html. (accessed: 14.07.2022).

[28] V. Radha and D. Hitha Reddy. "A Survey on Single Sign-On Techniques". In: *Procedia Technology* 4 (2012). 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012, pp. 134–139. ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2012.05.019. URL: https://www.sciencedirect.com/science/article/pii/S2212017312002988.

[29] J. Rayhawk. *What is D-Bus?* URL: https://www.freedesktop.org/wiki/Software/dbus/. (accessed: 12.07.2022).

[30] redhat.com. *Chapter 10. Managing Services with systemd*. URL: https://access.redhat.com/documentation/en-us/red%5C_hat%5C_enterprise%5C_linux/7/html/system%5C_administrators%5C_guide/chap-managing%5C_services%5C_with%5C_systemd. (accessed: 02.08.2022).

[31] D. Reis et al. “Developing Docker and Docker-Compose Specifications: A Developers’ Survey”. In: *IEEE Access* 10 (2022), pp. 2318–2329. DOI: 10.1109/ACCESS.2021.3137671.

[32] D. Robinson and K. Coar. *The Common Gateway Interface (CGI) Version 1.1*. RFC 3875. http://www.rfc-editor.org/rfc/rfc3875.txt. RFC Editor, Oct. 2004. URL: http://www.rfc-editor.org/rfc/rfc3875.txt.

[33] N. Sakimura, J. Bradley, and N. Agarwal. *Proof Key for Code Exchange by OAuth Public Clients*. RFC 7636. RFC Editor, Sept. 2015.

[34] Ravi S. Sandhu. “Role-based Access Control11Portions of this chapter have been published earlier in Sandhu et al. (1996), Sandhu (1996), Sandhu and Bhamidipati (1997), Sandhu et al. (1997) and Sandhu and Feinstein (1994).” In: ed. by Marvin V. Zelkowitz. Vol. 46. Advances in Computers. Elsevier, 1998, pp. 237–286. DOI: https://doi.org/10.1016/S0065-2458(08)60206-5. URL: https://www.sciencedirect.com/science/article/pii/%5C%5CS0065245808602065.

[35] Aneta Vulgarakis and Cristina Seceleanu. “Embedded Systems Resources: Views on Modeling and Analysis”. In: *Mälardalen University* (). (accessed: 27.09.2022).

[36] E. Yuan and J. Tong. “Attributed based access control (ABAC) for Web services”. In: *IEEE International Conference on Web Services (ICWS’05)*. 2005, p. 569. DOI: 10.1109/ICWS.2005.25.

[37] G. Gan; E. Chen; Z. Zhou; Y. Zhu. “Token-Based Access Control”. In: *IEEE Access* 322.8 (2020), pp. 54189–54199. DOI: 10.1109/ACCESS.2020.2979746. (accessed: 01.06.2022).

Appendix A

Appendix

A.1 liboacs

Bellow are the different header files that together with the external dependencies as described in Section 4.3.2, composes the OACS library, defining the core functionalities of the access control system.

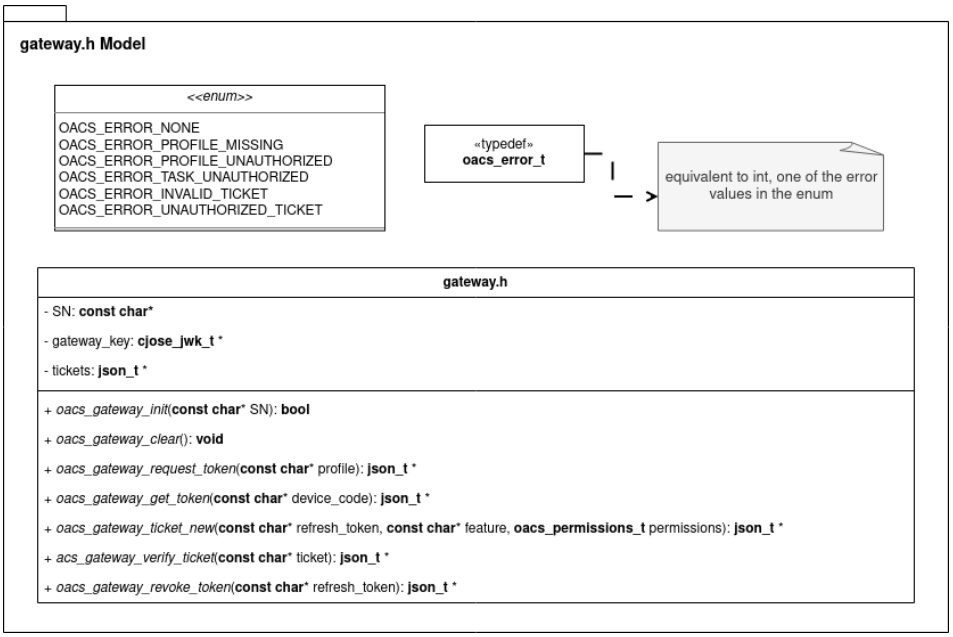

Figure A.1: liboacs/gateway.h

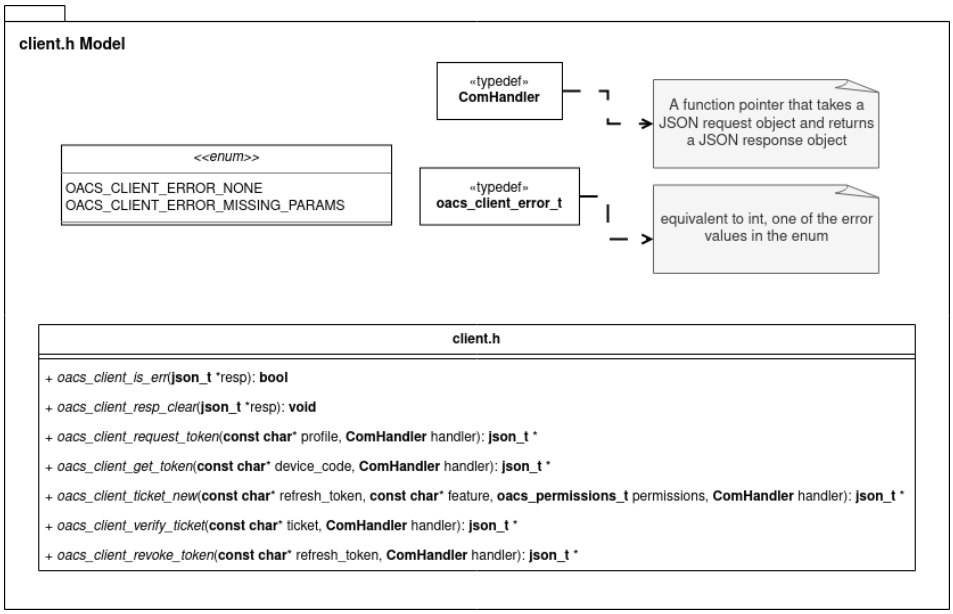

Figure A.2: liboacs/client.h

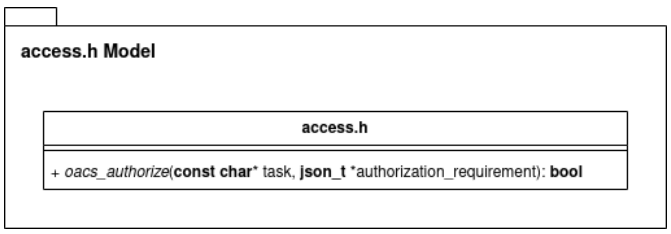

Figure A.3: liboacs/access.h

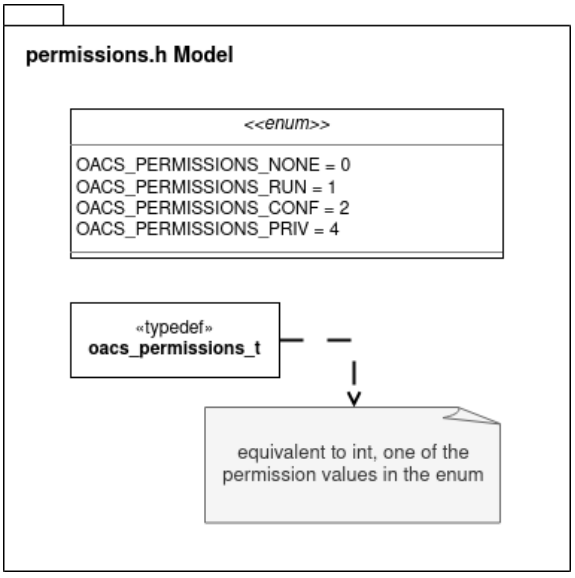

Figure A.4: liboacs/permissions.h

A.2 Login Flow

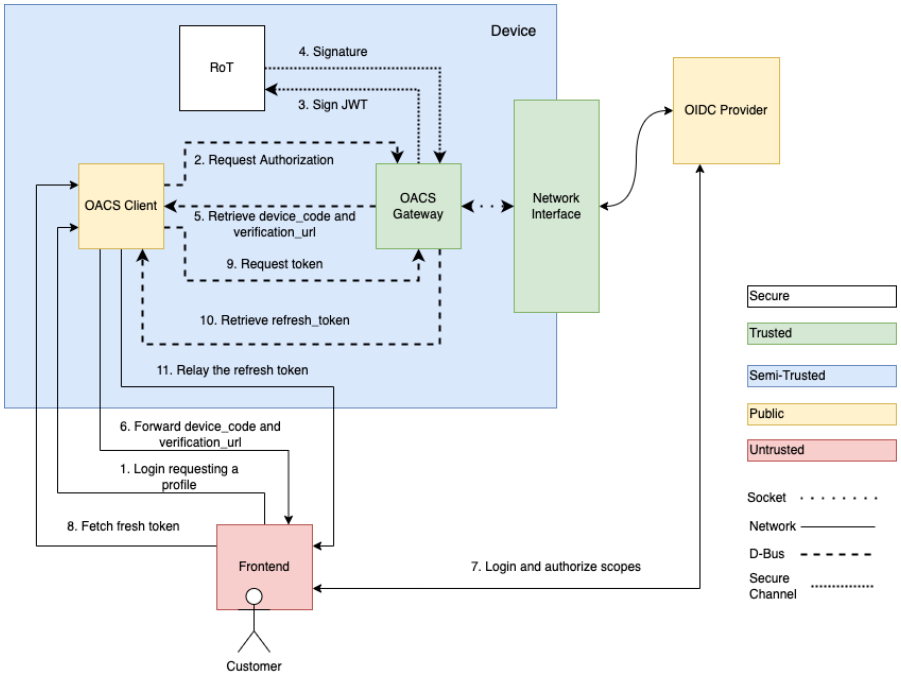

Figure A.5: Login Flow

The entire login flow shown in Figure A.5 has the following steps, for more details see the Appendixes A.6 and A.7 :

1. **Login requesting a profile**
   The front-end requests to login with a specific profile from the Authorization endpoint.
2. **Request Authorization**
   OACS Client requests the OACS Gateway to initiate the device code flow procedure by sending an authorization request to the device authorization endpoint with the requested profile, the devices corresponding profile ACL, the configured zone and the profile ACL of that zone, as scopes.
3. **Sign JWT**
   The client authentication tokens header and payload are constructed by the OACS Gateway and defined as previously described. A base64 encoded representation from the header and payload of the client authentication token is generated, which is then signed by the RoT device.
4. **Signature**
   The signature is returned and the signed JWT is assembled. This token can now be used to authenticate the internal OIDC client of the OACS Gateway, towards the OIDC provider.

5. **Retrieve device_code and verification_url**
   If the client is successfully authenticated, the OIDC provider's response will contain a device_code and verification_url. The verification_url has an embedded user_code which is paired with the device_code connects to the same session.
6. **Forward device_code and verification_url**
   The verification_url is forwarded to the front-end for authentication of user together with the device_code which will later be used to retrieve the issued refresh token.
7. **Login and authorize scopes**
   Front-end is used to convey the verification_url to the user, which can be opened on any browser. This provides single-sign-on capabilities for headless IoT devices, as the user is kept logged in via cookies set on the device used for authentication. The user_code and device_code are only valid for a short time period. After authentication, user consent is given for the requested scopes.
8. **Fetch fresh token**
   After successful user authentication the front-end will request a refresh token authorizing the requested profile, using the device_code.
9. **Request token**
   The access_token and refresh_token are requested from the OIDC provider, by the device, using the device_code provided by the front-end. Repeat steps 3 and 4 to generate the client authentication token.
10. **Retrieve the refresh_token**
    A access_token and refresh_token is retrieved from the OIDC provider. The user access right to the requested profile is confirmed by checking the device and zone ACLs for the profile. If the user has a valid entry in at least one of them, then the profile is permitted and the refresh_token is returned. Otherwise an error message is returned.
11. **Relay the refresh token**
    The response is relayed to the user-agent.

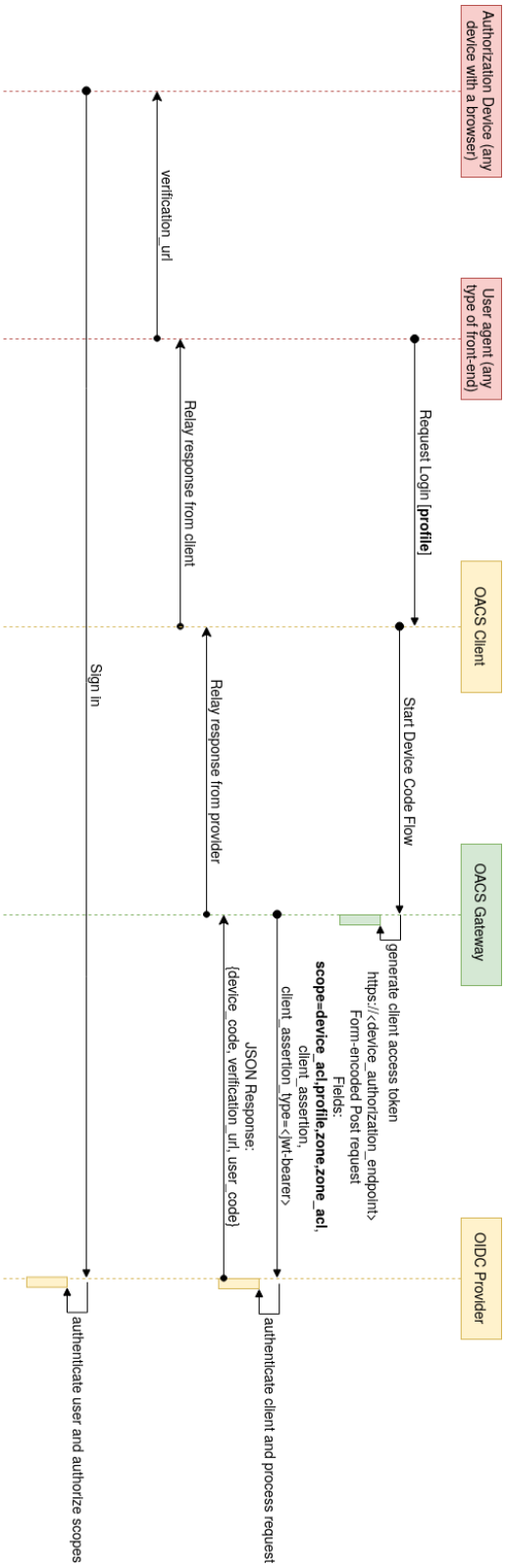

Figure A.6: Login Process 1

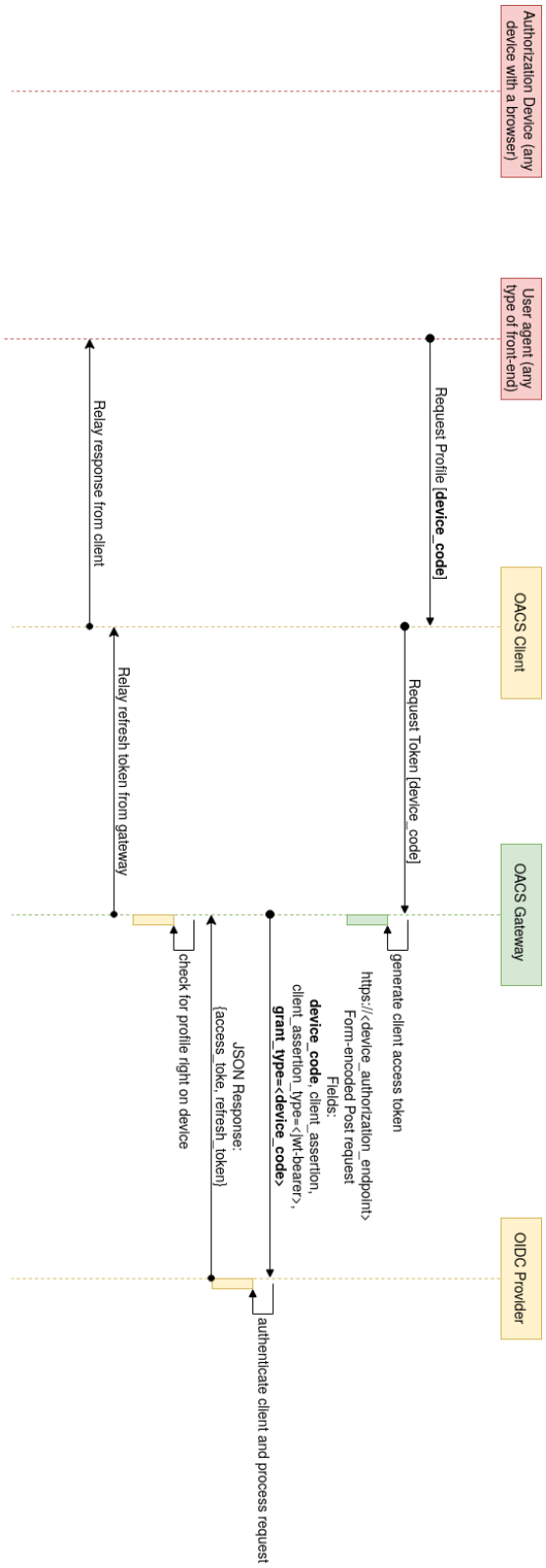

Figure A.7: Login Process 2

A.3 Ticket Authorization

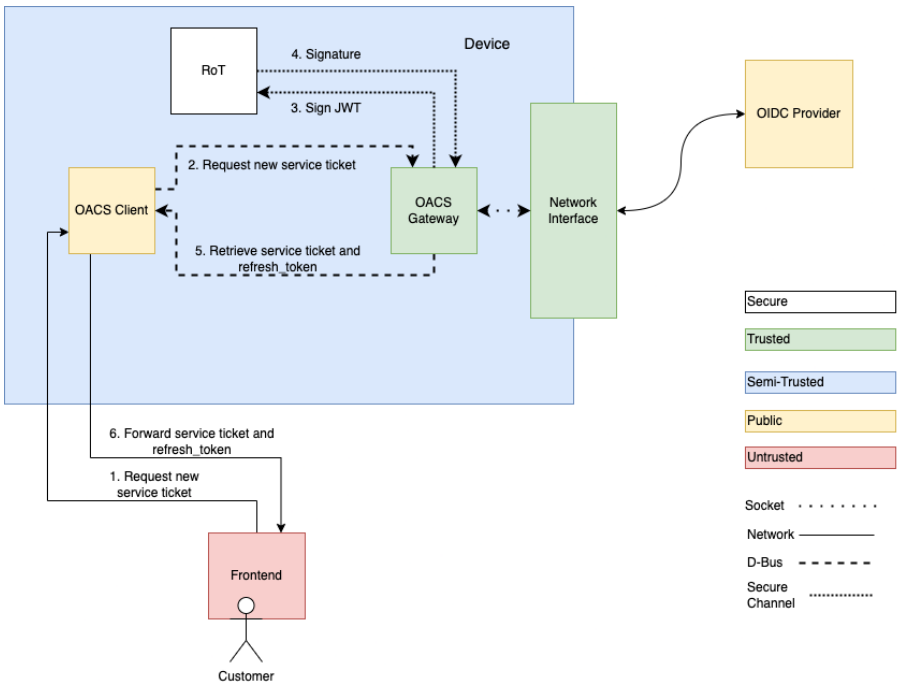

Figure A.8: Ticket Authorization

The entire ticket authorization flow shown in Figure A.8 has the following steps, for more details see the Appendix Figures A.9 and A.10:

1. **Request new service ticket**
   The front-end requests a new ticket for a specific feature and permission, using it's refresh token.
2. **Request new service ticket**
   OACS Client requests the OACS Gateway to issue the service ticket. First, the profile and ACLs needs to be retrieved.
3. **Sign JWT**
   The OACS Gateway generates a base64 encoded representation from the client authentication tokens header and payload, which is then signed by the RoT device.
4. **Signature**
   The signature is returned and the signed JWT is assembled. This token can now be used to authenticate the client towards the OIDC provider.
5. **Retrieve service ticket and refresh_token**
   If the refresh token was originally intended for the target device (authorized by user during login), then it will contain the profile ACL of the device. In that case, a new access token is requested from the OIDC Provider

and the feature permissions are verified. If authorized, the new service ticket is signed and returned together with a new refresh token using refresh token rotation.

If the devices isn't the originally intended target, OACS Gateway will use Client Credential Flow to fetch the profile ACL of the target device. If the user have the right to use the authorized profile on the target device, the process precedes as in the previous case and the ticket authorization is performed. If the user isn't explicitly blacklisted on the device, the zone profile ACL is checked as a fallback. If successful, a new service ticket is generated and issued.

The id of the newly generated service ticket is added to a ticket whitelist to keep track of the issued tickets and prevent reuse.

6. **Forward service ticket and refresh_token**
The ticket and refresh token is forwarded to the user-agent.

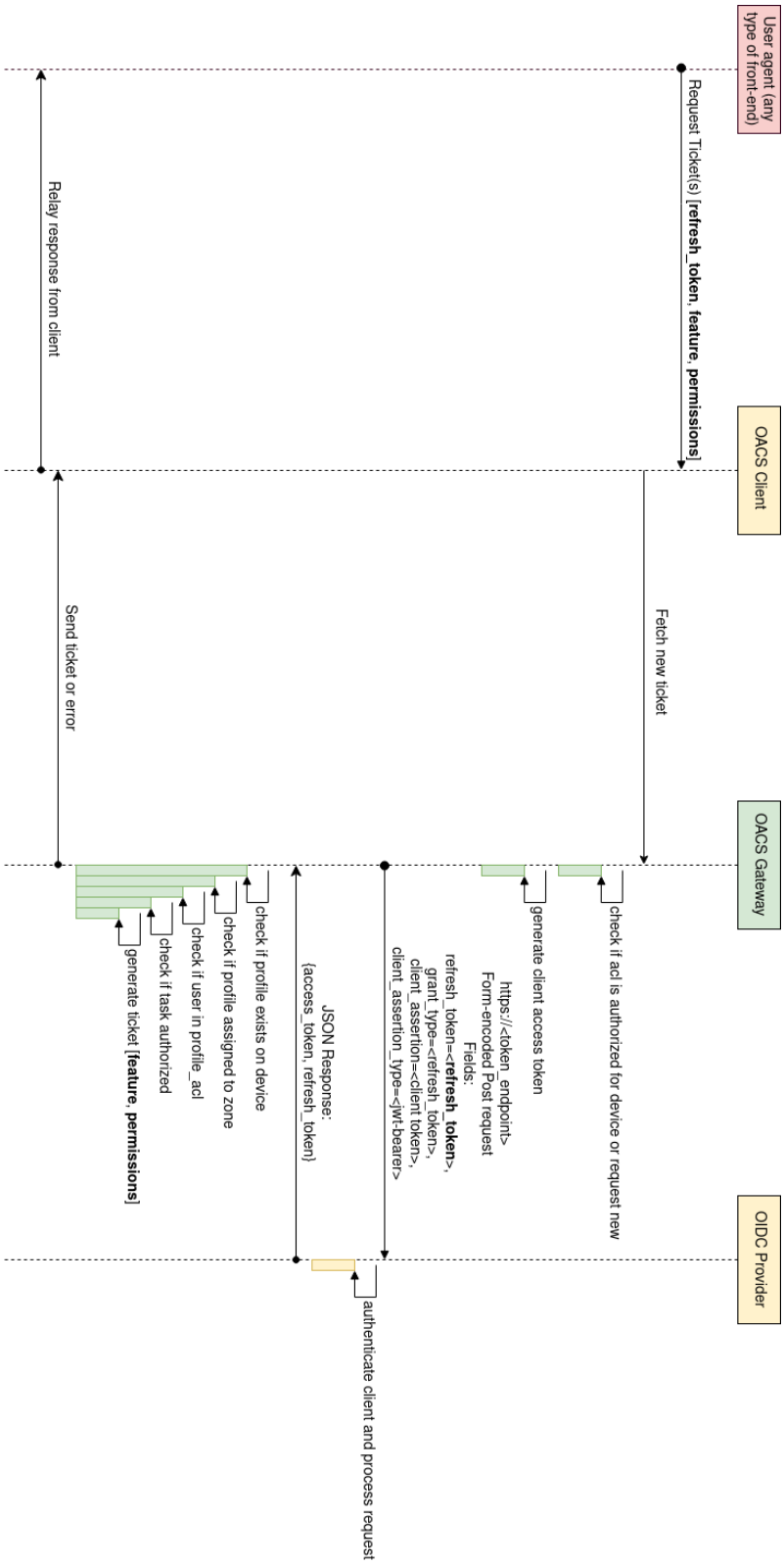

Figure A.9: Ticket Authorization Sequence - Correct device

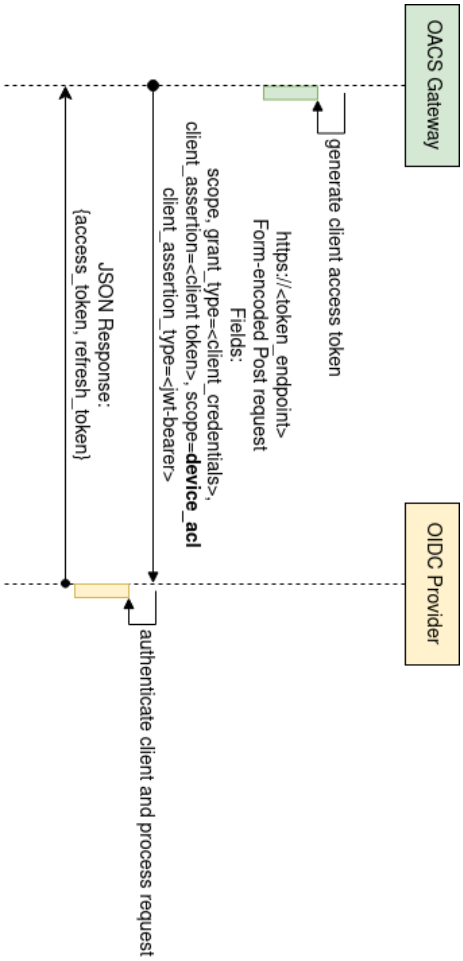

Figure A.10: Ticket Authorization Sequence - Wrong device

A.4 Token Revocation Flow

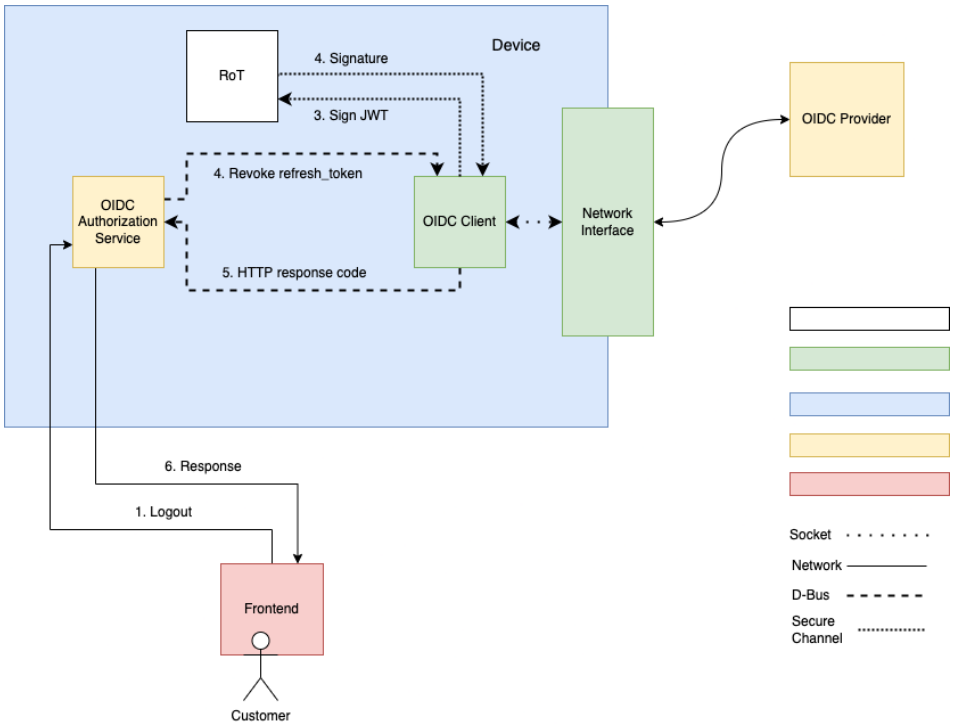

Figure A.11: Token Revocation

The entire token revocation flow shown in Figure A.11 has the following steps, for more details see the Appendix Figure A.12:

1. **Logout**
   The front-end requests to log out.

2. **Revoke refresh_token**
   OACS Client requests the OACS Gateway to initiated the token revocation flow using the refresh token provided by the front-end.

3. **Sign JWT**
   The OACS Gateway generates a base64 encoded representation from the client authentication tokens header and payload, which is then signed by the RoT device.

4. **Signature**
   The signature is returned and the signed JWT is assembled. This token can now be used to authenticate the client towards the OIDC provider.

5. **HTTP response code**
   The OIDC provider will respond with a HTTP response code 200 for success and any other response is considered a failure and the token is not revoked.

6. **Response**
   An arbitrarily response message is returned to inform about the result. The now revoked token can not be used to get a new access_token and thus the user is considered to be logged out.

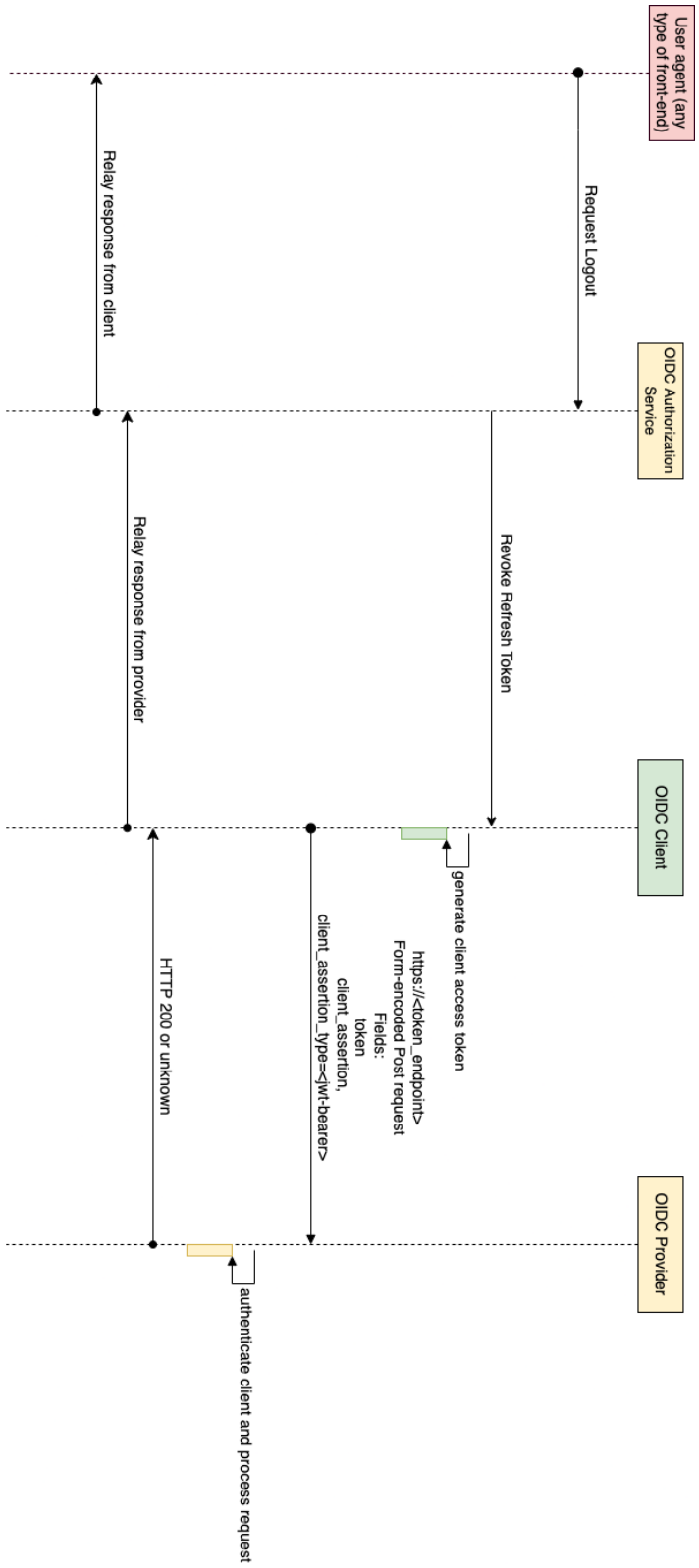

Figure A.12: Revoke token sequence