IMPLEMENTATION AND PERFORMANCE ANALYSIS OF CONTEXT-AWARE ROLE-BASED ACCESS CONTROLS FOR CLOUD-BASED IOT PLATFORM

**A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of**

MASTER OF SCIENCE

in Computer Engineering

**by
Mete Mertkan DÖŞEMECİ**

**September 2019
İZMİR**

We approve the thesis of **Mete Mertkan DÖŞEMECİ**

Examining Committee Members:

Assoc. Prof. Dr. Tolga AYAV
Department of Computer Engineering, İzmir Institute of Technology

Dr. Nesli ERDOĞMUŞ
Department of Computer Engineering, İzmir Institute of Technology

Assoc. Prof. Dr. Ahmet Tuncay ERCAN
Department of Computer Engineering, Yaşar University

27 September 2019

Assoc. Prof. Dr. Tolga AYAV
Supervisor, Department of Computer Engineering
İzmir Institute of Technology

Assoc. Prof. Dr. Tolga AYAV
Head of the Department of
Computer Engineering

Prof. Dr. Aysun SOFUOGLU
Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGMENTS

I would like to express my special thanks of gratitude to my teacher Prof. Dr. Yusuf Murat ERTEN who game me the opportunity to do this project.

I would also like to thank Dr. Emrah TOMUR who helped me a lot for this project in a limited time frame.

ABSTRACT

IMPLEMENTATION AND PERFORMANCE ANALYSIS OF CONTEXT-AWARE ROLE-BASED ACCESS CONTROLS FOR CLOUD-BASED IOT PLATFORM

IoT has received substantial attention in both industry and the scholarly world in the recent years. The main idea is to interconnect the physical world with the digital world. Sensors read physical world and present processible data. This data needs to be secured. Currently, most of the cloud based IoT platforms use some sort of Role-Based Access Control (RBAC) , which is one of the approaches to control access to the devices, hence the data. In some cases RBAC is insufficient for fulfilling constantly changing requirements of IoT. ABAC (Attribute Based Access Control) can be flexible enough for fulfilling. However ABAC requires higher level of maintenance. We wanted to implement a access control method that uses both RBAC's and ABAC's advantages. We called it OBAC(Operation Based Access Control). Authorization is being implemented in a plug and play manner. We implemented that way because; It is designed for cloud platforms and we wanted to switch between access control methods easily. The results of the experiment shows that proposed access control(OBAC) had minimum latency and management steps across other access control methods.

ÖZET

BULUT TABANLI NESNELERİN İNTERNETİ PLATFORMU İÇİN BAĞLAM-DUYARLI ROL TABANLI ERİŞİM DENETİM YÖNTEMLERİ UYGULAMASI VE PERFORMANS KARŞILAŞTIRMASI

IoT, son yıllarda hem endüstride hem de bilim dünyasında büyük ilgi gördü. Ana fikir, fiziksel dünyayı dijital dünya ile birleştirmektir. Sensörler fiziksel dünyayı okur ve işlenebilir veri sunar. Bu verinin güvenceye alınması gerekiyor. Şu anda, bulut tabanlı IoT platformlarının çoğu, cihazlara erişimi kontrol etmek için kullanılan yaklaşımlardan biri olan Rol Tabanlı Erişim Kontrolü (RBAC) kullanıyor. Bazı durumlarda RBAC, IoT'nin sürekli değişen şartlarını yerine getirmede yetersizdir. ABAC (Özellik Tabanlı Erişim Kontrolü), doldurma için yeterli olabilir. Ancak ABAC daha yüksek düzeyde bakım gerektirir. Hem RBAC'i hem de ABAC'in avantajlarını kullanan bir erişim kontrol yöntemi uygulamak istedik. Buna OBAC (Operasyon Bazlı Erişim Kontrolü) adını verdik. Yetkilendirme, takma ve oynatma tarzındadır. Bu şekilde uyguladık çünkü; Bulut platformları için tasarlandı ve erişim kontrol yöntemleri arasında kolayca geçiş yapmak istedik. Deneyin sonuçları, önerilen erişim kontrolünün (OBAC) diğer erişim kontrol yöntemleri arasında minimum gecikme ve yönetim adımlarına sahip olduğunu göstermektedir.

To my lovely fiancé

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

IoT	Internet of Things
AWS	Amazon Web Services
GCP	Google Cloud Platform
RBAC	Role Based Access Control
ABAC	Attribute Based Access Control
CA-RBAC	Context Aware - Role Based Access Control
UCON	Usage Control Model
SA	Subject Attribute
OA	Object Attribute
CR	Context Rule
MCR	Multiple Context Rules
AAT	Attribute Assignment Table
SAOA	Subject Attribute Operation Assignment
OAOA	Object Attribute Operation Assignment

CHAPTER 1

INTRODUCTION

The advancement of the internet towards IoT will immensely affect the lives of residents in a positive way. Smart devices in the house making everyday life easier. New applications can be created by the mix of the physical and digital world. Mobiles, wearable sensors, and smart gadgets with improved capacities to enable them act independently can be utilized to support new applications for human services, such as transportation and healthcare, as well as to improve business efficiency and upgrade security, ultimately benefiting the quality of life of residents.

One of the significant aspects of the IoT applications is context awareness. Context awareneess is having ability to understand and adjust itself according to changing environmental conditions. These days, environmental conditions changes very fast and unpredictable way therefore IoT devices such as, smart homes, smart vehicles, smart cities should adapt themselves those changing conditions in order to provide comfortable living spaces for people. In any case, one of the most noteworthy concerns about these applications are the protection and security requirements. 2018 Owasp IoT Top 10 vulnerabilities' 3rd entry : (The OWASP IoT Security Team, 2018): ***"Insecure web, backend API, cloud, or mobile interfaces in the ecosystem outside of the device that allows compromise of the device or its related components. Common issues include a lack of authentication/authorization, lacking or weak encryption, and a lack of input and output filtering. (The OWASP IoT Security Team, 2018)".*** also 8th entry indicates : (The OWASP IoT Security Team, 2018): ***"Lack of security support on devices deployed in production, including asset management, update management, secure decommissioning, systems monitoring, and response capabilities. (The OWASP IoT Security Team, 2018)".***

Security is a basic requirement for the IoT. The quantity of associated gadgets is expanding exponentially. This has offered ascend to an alluring and new attack surface. Access control is a fundamental part of security solutions for IoT according to Owasp's 3rd and 8th entries. Major cloud suppliers, for example, Amazon Web Services (AWS) , Microsoft Azure, Google Cloud Platform (GCP), use customized version of RBAC with predefined roles and permissions for their access control requirements in the cloud. However, a formal access control demonstrating genuine cloud based IoT stages is still lacking. Due to large number of devices in the IoT environment RBAC suffers role explosion problem. Role is a way of grouping permissions for each device and role explosion problem is having tremendous amount of roles. This makes the whole system unmanageable. Adding a new role usually requires system level update. On the other hand, in ABAC (Attribute based access control) , roles are replaced with the subject attributes which introduces higher granularity. Like RBAC, ABAC is also works best with a centralized architecture, making system level updates more difficult.

1.1. Motivation

As previously stated in the introduction, the ever-increasing appropriation of IoT in our daily lives raises serious questions on privacy and security. In IoT environment, it is expected to contain billions of devices interacting with each other through networks. Privacy becomes prerequisite in those kind of environments since they may contain sensitive information. Traditional access controls such as RBAC and ABAC may not be the perfect approach when one considers context data.

Our motivation was using the advantages of the RBAC and ABAC in context aware environment. Implement such code base and compare their performance. To focus on implementation and comparing we had to mock context data. Integrating authorization into implementation is not a ideal choice for us since we are going switch between access control methods. Plug and play concept allows to switch between access control methods easily. Authorization is designed to be a step in the request pipeline. This way we can use in a plug and play fashion.

1.2. Aim of Thesis

The objective of the thesis is to implement OBAC in context aware environment and compare it with access control models RBAC and ABAC with context information to specifically meet the requirements of IoT applications. Such as time of the day, location of the requestor, emergency alarm status.

1.3. Outline of Thesis

The rest of the thesis is organized as follows: In chapter 2 we took a look at related studies. In chapter 3, the concept of context awareness has been given in section 3.1 and all compared access controls in section 3.2. In chapter 4 we explained the experiment. In chapter 5 we present the test results and In chapter 6 we present the possibilities to take the work further.

CHAPTER 2

RELATED WORK

For this work two concepts were the key. One of them is the usage of context in access control, and the other one is to combine advantages of RBAC and ABAC together. When both concepts merged we can obtain flexible, fine-grained, easy to manage and scalable access control model.

Utilizing context awareness in access control guarantees a fine-grained and increasingly powerful access control system. In this manner, a large number of the specialists proposed various works that coordinate context awareness with access control, however the proposed models in these studies focused on the general computing applications that include basic situations with less number of communicating components (one-to-many), for example, smart home, smart office and so on. The term IoT, as we use it here, refers to many-to-many kind of communication with various individuals from application (inhabitant, process, service, device) through the cloud infrastructure and social networks. For such situations, access control for IoT applications ought to be dynamic and more fine-grained requiring the use of context information wisely.

RBAC is the most embraced access control model by business environment, since it has the benefit of simple organization through role utilization.On the other hand, predefining all interactions between things isn't a simple procedure while considering IoT environment with complex communications. Moreover, role creation stage is additionally tedious since the environment has such a large number of subjects with various rights, which cause the making of unnecessary number of roles prompting role-explosion issue. For this reasons, it is accepted that the use of RBAC model alone is deficient to guarantee the security of IoT applications. ABAC is a more advantageous access control model for use in situations with dynamic communication, since it provides flexibility with attribute usage. In spite of this feature, the policy survey as far as organization is unwieldy since any gathering capacity like role isn't utilized in this model. Thus, it is required to produce another bound together access control model that joins the adaptability of ABAC effortlessly of organization. These two methodologies are evaluated independently beneath.

2.1. Context-Aware Access Control Models

(Covington et al., 2001) has broadened the general role based access control approach by including environmental roles that catch the security relevant data from the environment for fine-grained access control. Context data is characterized as a role in their study. Roles activated with the context information from the environment. At the point when an individual access to a specific asset, the member's subject role decide the permitted assets and the related environment roles express the imperatives. In other words, the permissions are associated with both subject roles and environmental roles. For instance; in a context aware home, the individual assigned to the babysitter role can reach the intercom service if the weekday and working hour roles are dynamic. However, this methodology will build the quantity of roles by allocating role to the environment conditions like the day of the month, time, specific area in the house and so on. This is unfortunate for IoT applications that as of now contain numerous components.Execution of this proposed model is performed by (Covington et al., 2002).

(Al-Muhtadi et al., 2003) displayed an authentication mechanism and context-aware access control model that is called Cerberus. The authentication mechanism part composed of two modules; one included the communication protocols like SESAME, username-password, Kerberos etc. and the other contained the authentication devices such as fingerprint scanner, PDA, smart badge etc. Separation of these two modules gave adaptable authentication mechanism. The context data procured by a pretty basically planned context infrastructure which utilizes first request predicates and boolean algebra. The access control part of this project is controlled by the inference engine. The system is offered a dynamic access control that suggests the consistent checking of context. If there should be an occurrence of identification of any infringement in context data, access to a specific gadget or asset is ended. This study differs from ours in where it works. The designed system is not for the IoT applications as it is dawned on proposed scenario.

Another work that consolidates RBAC with context awareness, produced for general computing applications is exhibited by (Kulkarni and Tripathi, 2008). In this study the context data is likewise utilized for conceding a user to a role and to keep being an individual from that role. For a patient information system given in the paper, the worldly requirements can be utilized to concede a nurse to the 'NurseOnDuty' role just inside the working hours. At the point when the imperatives are never again hold, the nurse's participation in NurseOnDuty role is ended. Likewise the roles are dynamically created in the application design period and activated only during related application's execution time. As a matter of fact the context data is utilized for various tasks, for example, role permission assignment, permission activation and resource access. As referenced over, this study is produced for general computing applications that don't utilize the connections of numerous things like individual, process, devices and so forth. This CA-RBAC model isn't appropriate for IoT applications that contain numerous components, as unique role creation for each application will prompt excess of roles.

The most similar study to ours is the ConUCON security module that was developed for the Web of Thing applications by (Bai et al., 2014). ConUCON make use of the context awareness to provide security and privacy in usage control. The main difference of this work is that it is based on UCON (Usage Control Model) instead of RBAC. In RBAC we can use role feature for grouping the users but in ConUCON every subject managed individually

(Abdella et al., 2016) used combination of context awareness and RBAC in Android based mobile applications. The context data is associated with the permissions. To gain an access to a resource, defined context information should be ensured. Their model does not offer a dynamic access control and not suitable for IoT environment.

2.2. RBAC with ABAC Extension

According to the NIST, the first integrated access control model initiative called RABAC (Role-Centric Attribute Based Access Control) is proposed by (Jin et al., 2012). In this model, the subjects are assigned to roles and permissions are grouped by roles as the base of RBAC and the subjects and objects are associated with attributes. To be able to constrain the permissions according to the subject and object's attributes, they added a new module, permission filtering policy (PFP). Proposed model is good at addressing the role-explosion problem since it uses constraints/filters on permissions, however they did not use environmental attributes like time of day or location. We believe that, it can be crucial in IoT systems.

Another access control model proposed by (Rajpoot et al., 2015) is AERBAC. AERBAC is content based context aware access control. They used conditions associated with permissions in order to overcome the role explosion problem. Also, they addressed the role-permission explosion problem by using object attributes in permissions.In their model single permission can include an object group which have a few objects with common attributes instead of a unique object. This approach reduces the number of related permissions with roles. However, as any increment in object attributes cause to exponential grow in number of object groups, attribute based object grouping can cause another managing problem in an environments having huge amount of objects.

(Hasiba et al., 2017) likewise proposed a model that extends properties of ABAC and RBAC by joining role and attribute ideas in a role driven way. As in RBAC, subjects are assigned to roles and roles have permission rules that limited with constraints. The rules are separated into two; private and shared object rules. This methodology empowers limiting every user to access just his/her own information without requiring separate role or rule. This model addressed both the role-explosion and role-permission explosion problems.

(Qi et al., 2015) proposed a model that is created the roles based on static attributes and formed the rules considering dynamic attributes. Therefore, they utilized attributes in both user to role assignment and role to permission assignment. Thus, number of roles in RBAC and number of rules in ABAC are significantly reduced.

As can be seen from the reviewed papers, attribute based and context-aware RBAC approaches are similar in terms of limiting permissions. However, none of the papers in this section are addressing IoT applications.

2.3. Iot Environment

Kevin Ashton first mentioned the term "Internet of Things". It was the title of the his presentation at Procter & Gamble (P&G) in 1999. IoT is the trend topic nowadays. Notion means that any device with an on/off switch can be connected to, and controlled through the Internet (Morgan, 2014). IoT domain hosts tremendous amount of many-to-many interaction that introduces many challenges. Most important ones are the security and privacy challenges. For this part of security, usage of access control models plays an important role.

CHAPTER 3

BACKGROUND

In this chapter, the main ideas that the thesis is based on are explained separately.

3.1. Context-awareness

Context-awareness is getting more popular every day. The term was first put forward by (Schilit et al., 1994). The application that can adapt themselves to context was their definition. This means, an application that just displays the context is not context aware. In order to be context aware, the application needs to adapt itself to the context.

An example situation can be the following scenario in a smart home. Let us imagine the following components: main door, inhabitants, and smart home application. An inhabitant can open or close the door via smart home application. In an emergency such as fire, we expect the smart home to do nothing different, but if that application has context-awareness, we expect the smart home application to never let the main door stay closed during a fire. The initial setup was following orders from humans, but in an emergency situation it should adapt itself to the environment.

This is a specific example. The most general definition that consists of both using context and adapting to context is asserted by (Abowd et al., 1999): ***"the system is context-aware if it uses context to provide relevant information and/or services to user, where relevancy depends on the user's task (Abowd et al., 1999)".***

3.1.1. Context

There are several different definitions for context, but the most comprehensive and accepted explanation is also introduced by Abowd et al. in 1999. According to them, ***"context is any information that can be used to characterize the state of an entity. An entity can be a person, object or place that is considered relevant to the interaction between a user and an application, including the user and application itself (Abowd***

et al., 1999)". It means that any information from any sensor, or extracted through internet can be used if that information has relevance to your system, such as location, time of the day, temperature, and air quality.

3.2. Related Access Control Models

Access control is a security mechanism that limits who or what can view or use resources in the system based on predefined roles, permissions or policies. There are various types of access control models, but in this project, we will focus on two of them: "Role based access control" and "Attribute based access control". Definitions and structure is as follows:

3.2.1. RBAC: Role Based Access Control

Role Based Access Control has widespread use in industry, business, government etc. After significant studies, NIST has proposed a nearly completed RBAC model in 2003 under a document named NIST solution (Weber, 2003). In this document, four versions of RBAC are explained in detail. The variations originated from used features like hierarchy and constraints. The common aspect of these different versions is that the base model depends on three core elements in its nature; users, roles and permissions, and this version is called as Flat RBAC (Sandhu et al., 2000). Users, roles and permissions, and their relationships are explained separately below.

Users: The users are the entities who can gain access on any object in the system. A user can be a person or non-person like computer or process.

Roles: Function of the role can be considered as grouping operation of users who have same permissions. On the other hand, it also enables grouping the permissions. Therefore, a role which is the main element of RBAC, is a bridge between both the group of users and their permissions. Title, department or job can be an examples of roles.

Permissions: A permission is composed of operations and objects. An object can be a file, document or anything that need to be protected. Operation is an action that is allowed to be performed on a specific object like reading, writing, updating, etc. The diagram that illustrates the relationship between these elements is given below in Figure 3.1.

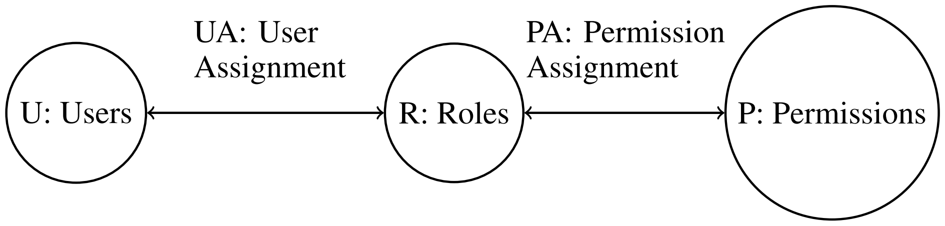


Figure 3.1. Flat Role Based Access Control (Sandhu et al., 2000).

In general terms, RBAC is the method of assigning a role to each user and having its own permissions for each role. It offers a simple and elegant way of administrative management.

- Advantage of RBAC is given below:
  - Simple to manage individual user permissions.
- Disadvantages of RBAC are given below:
  - **Role Explosion Problem**: In order to satisfy large organizations it requires large number of roles and permissions. That leads to high number of storage requirements and reading operations from disk. Moreover, every role needs to be created with caution, otherwise duplication is inevitable.
  - **Role Permission Explosion Problem**: Large number of objects require individual permissions and that causes many permissions under a single role.
  - **Role Creation Time Problem**: Complexity of the system requires a complex relationship with roles and users and this causes time consumption problem.

3.2.2. ABAC: Attribute Based Access Control

Attribute based access control gives more flexibility over RBAC. It introduces attribute concept into RBAC. Users have attributes, and attributes are used to decide granting access (Jin et al., 2012). Attributes are any information that is used for identification. There are 3 kind of attributes:

Subject Attributes: Any information about a user are subject attributes, such as title, job, location, etc.
Object Attributes: Any information regarding object, such as type, status, etc.
Environmental Attributes: Any information defining environment, such as time of the day, temperature, etc.

ABAC decides granting or denying access based on the rules created with attributes of subjects, objects and environment. ABAC model is given below in in Figure 3.2.

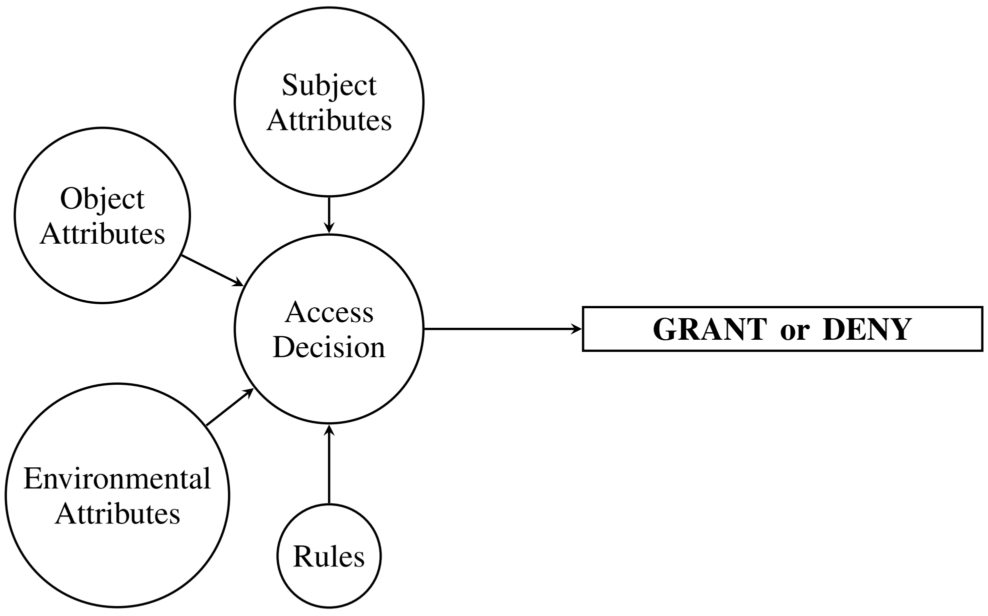


Figure 3.2. Attribute Based Access Control

- Advantages of ABAC are given below:
  - Roles are replaced with subject attributes which speeds up the process.
  - It has high granularity.
  - It is more suitable for dynamic environments.
- Limitation of ABAC is given below:
  - It is hard to determine individual users permissions and limitations.

3.3. MQTT (Message Queuing Telemetry Transport)

MQTT was invented by Dr Andy Stanford-Clark and Arlen Nipper in 1999. It is a publish/subscribe, simple and lightweight messaging protocol. The design principles are for minimizing bandwidth and device's resource usage. Those principles turn to be great for machine to machine communication.

MQTT protocol is based on a server and clients, such as any other internet protocol. Server has specific name in this protocol, MQTT Broker. When a client wants to send data to MQTT broker, it is called publish. When a client wants to receive data from MQTT broker, it is called subscribe.

Let's think about a machine to machine communication example. Image there are three clients: Temperature sensor, mobile phone and laptop. Scenario goes as follows, we want to send temperature sensor data to mobile phone and laptop. MQTT broker creates a channel called "Temperature Information" and Temperature sensor publishes data to that channel. Mobile phone and laptop subscribes to same channel and they receive when a data publishes to that channel. Figure 3.3 demonstrates the given example.

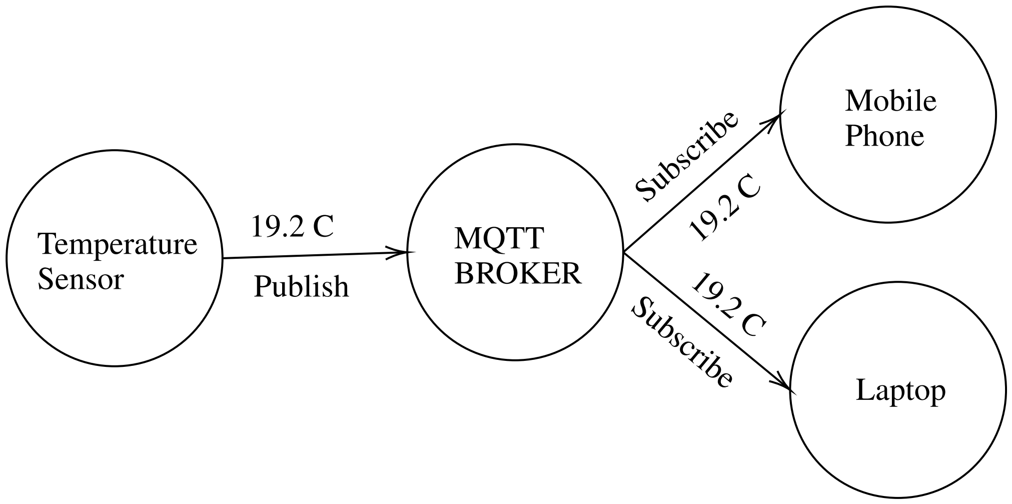


Figure 3.3. MQTT Example Scenario

3.4. Fiware

Fiware is a context data management tool which focuses on IoT. Fiware was choosen for this project because it is an open source and can be containerized. Advantages of the fiware are listed below:

- It has an easy development and deployment process. This allows to create simple or complex structures up and running in a short term.
- Open source platform provides contribution from many other developers outside of the creators of the Fiware. Extra objective eyes from outside are always appreciated for this kind of platforms.
- Scalability is a must in today's environment. Using the power of Docker, a developer can scale up the Fiware system simply by duplicating the Docker image.

3.5. Dependency diagram

Figure 3.4 is the dependency diagram of the project.
Numbers on the 3.4 represents the usage in the specific project.

- Definition was used for business logic entities. Base response and request objects. Access control method spesific entities such as permissions, roles, rules, users, objects.
- Data was used for database operations.
- ContextEngine was used for providing context information. Since this is a simulation context information is arbitrary.
- Repository was used as layer between data and rest of the project. Every database operation should go from repository project. It is the door for the data layer.
- Mapper was used as helper class for mapping definition entities to servis layer entities. That way we managed to secure definition layer objects. Exposing definition layer object may lead to security vulnerabilities.

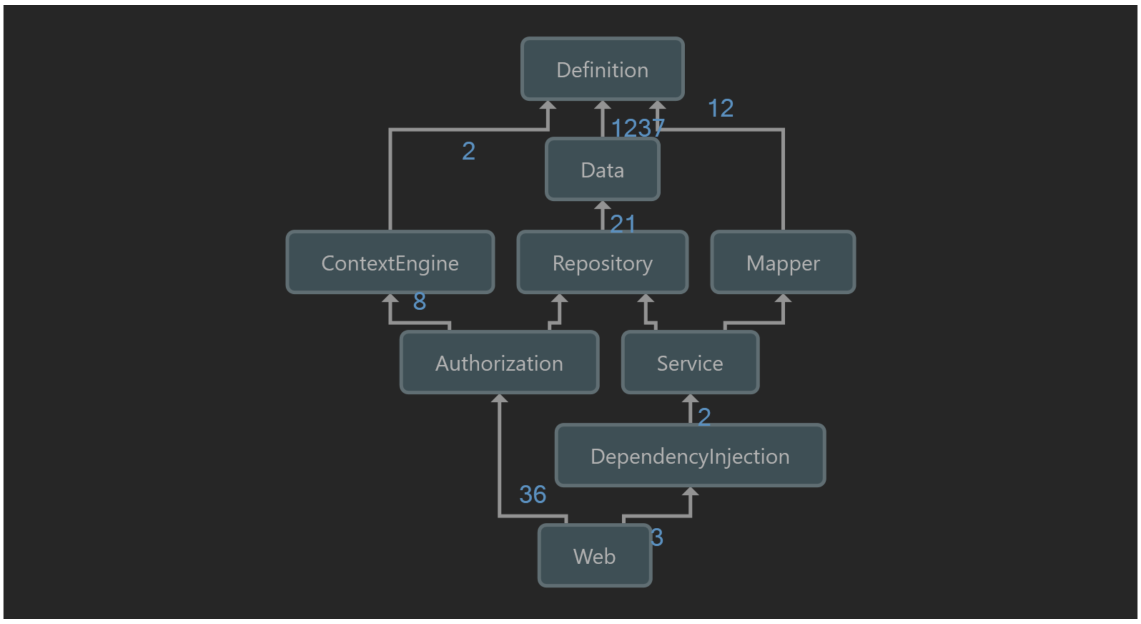


Figure 3.4. Dependency diagram

- Authorization was used to authorize the client. It uses context engine and repository projects.
- Service was used as layer between web and repository.
- DependencyInjection was used for decoupling the creation of the usage of an object.
- Web was used for user interface for client.

3.6. Onion Architecture

The term "Onion Architecture" was coined by Jeffry Palermo back in 2008 in a series of blog posts. The architecture is designed as a solution to common problems like coupling and separation of concerns. In this project we used onion architecture on the backend implementation. Onion architecture is based on inversion of control principle. Instead of depending concrete classes, it encourages to depend on interfaces. The architecture does not depend on Data layer as in classical n-tier architecutes it depends on Domain layer.

- Domain Layer is in the center of the architecture. This layer has business and behavior objects.
- Repository Layer creates a bridge between domain objects and business logic.
- Service Layer holds the business logic and this layer is used for communicating between repository and UI.
- UI Layer is the highest layer and it interacts with clients.

Figure 3.5 is an example representation of the onion architecture.

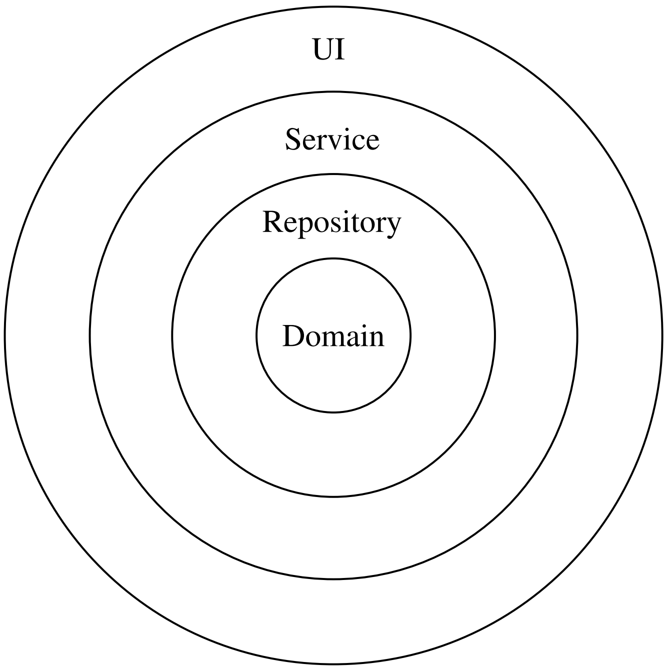


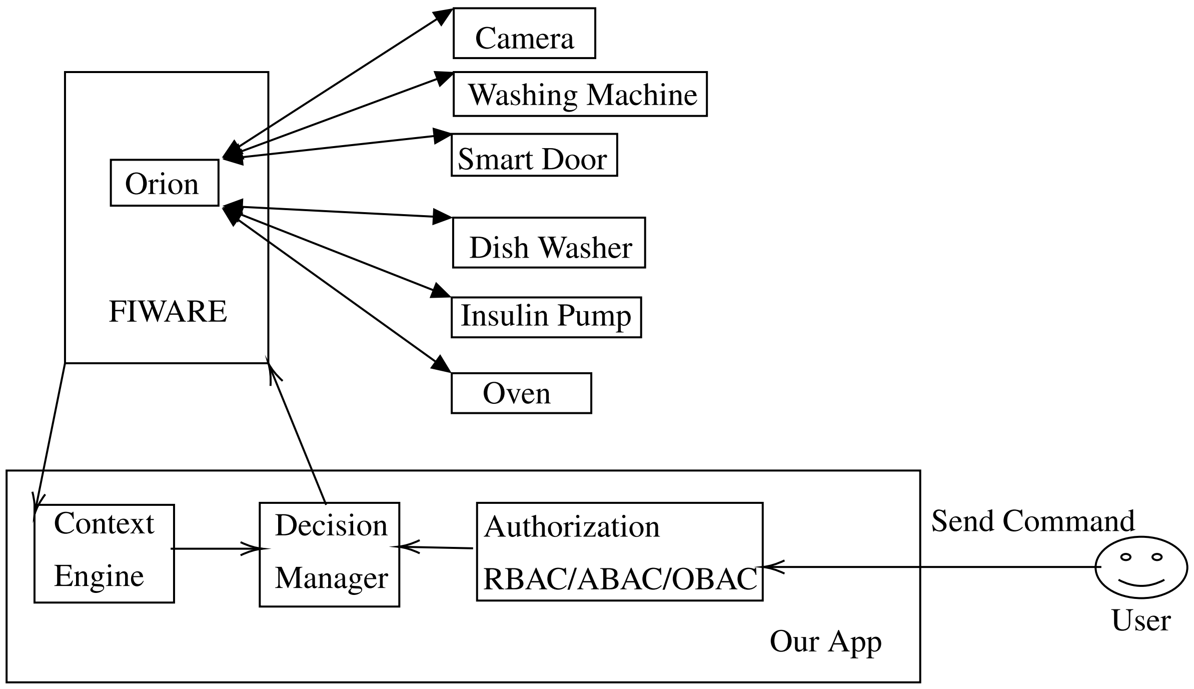
Figure 3.5. Onion Architecture

3.7. Middleware

Asp.NET Core allows to write middleware services to control the request life cycle. In this project, we used authorization as a middleware that way we can switch between authorization methods with one line of code change. A request comes with an authorization token in the header, after validating the token we identify the user, we determine the action want to be taken. Then we check if identified user has privileges to execute the determined action. If user has required privileges we forward that action to Fiware, If user doesn't have required privileges we simply return "access denied".

3.8. Overall Structure

The following figure represents the overall structure. User sends a command to our project. That command goes to authorization module. If user does not have required permission,the user gets a response as 'access denied', otherwise that command goes to a decision manager.The decision manager receives the command and it uses context information forwarding that command to Fiware. Fiware uses orion context broker to execute the command coming from user. Orion context broker allows to manage entire lifecycle of the connected devices.



3.9. Database Diagram

Figure 3.6 is the database diagram of the project.

- Context table is used for storing the context.
- It has fields from the scenario that we mentioned on section 4.2.
- Rule Sets table is for storing the rules for ABAC model.
- Permissions, AspNetRoles, AspNetUsers are for RBAC implementation of the model.
- Object, Object Attributes, Operations, AspNetRoles, AspNetUsers tables are for OBAC implementation of the model.

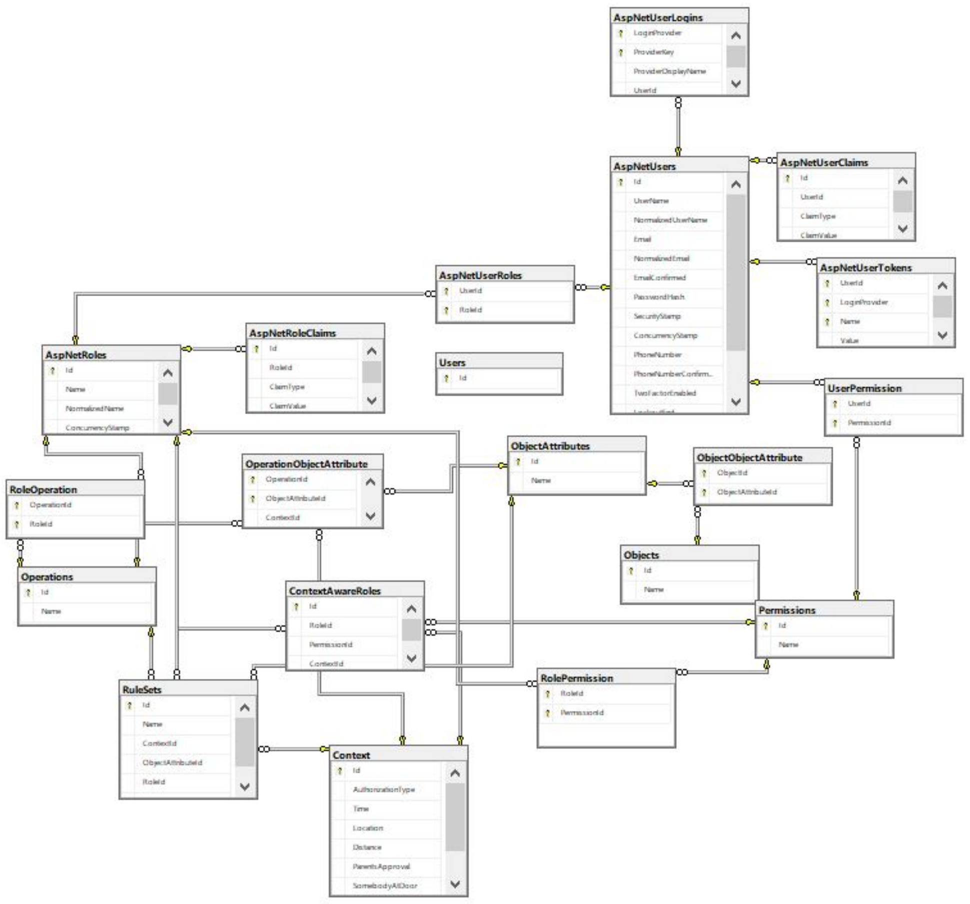


Figure 3.6. Database diagram

CHAPTER 4

EXPERIMENT

4.1. Designed Model: OBAC

This model can be considered as combination of RBAC and ABAC. Role feature of RBAC enables the grouping of the users and permissions, and this provides sustainable user and permission management.ABAC has subject attributes feature which gives more granularity. We combined both features in the designed model. However, IoT environment has dynamic interactions of vast amounts of devices, and it is not possible to define all interactions among these devices exactly and in advance. Besides, forcing to assign a role to each subjects will lead to creation of excessive number of roles due to the variety of vast amount and different types of subjects. So, it is inefficient to write predefined roles and permissions for RBAC and rules for ABAC beforehand. Considering the existence of role-explosion problem of RBAC, it is obvious that a optimal solution is needed to remove these problems for kaleidoscopic environment. The solution in this model is to group the operations like reading, writing, turning off and so on with subject attributes since the number of operations that can be executed in a system are always limited. This way we prevented the role-explosion problem which have high occurrence probability for IoT environment. Also, object attributes are used to group the objects in order to enable easy administration of objects and to prevent the role - permission explosion. This model is introduced in "Context Aware Role Based Access Control Model for Internet of Things Applications" (Genc, 2018).

4.1.1. Model

Operations (Op): Operations indicate defined actions that are allowed to perform on an object in a system, and the subjects are grouped under specific operations.
Subjects(S): Subjects are the entities that have access right on objects under a particular

situation. In the IoT environment; users, services, processes, applications, devices can be the subjects of the system.

Subject Attributes (SA): Subject attributes capture the properties of each subjects. Each subject should have at least one subject attribute. There is a many-to-many mapping between SA's and subjects which means a SA can be assigned to many subjects and a subject can have more than one SA.

Objects (O): Objects are the resources that are protected by the security policies.

Object Attributes (OA): Object attributes are assumed to be given by the manufacturer or administrator of the system, and it defines the properties of the object. Each object should have at least one object attribute. There is many-to-many relationship between objects and OA's like the mapping between subject and SA's.

Context Expression (Con): In this model, the constraints limiting the access to an object is called as context since the restriction is based on context information which contains subject, object or environmental attributes. Contexts are stored in the database as follows:

$Con_i = <ContextName, Operator, Value>$ where $Con_i \in Con$.

For comparison operators; equal ($=$), not equal ($\neq$), greater than ($>$), less than ($<$), greater than and equal ($\geq$), less than and equal ($\leq$), not ($\neg$) can be used.

Context Rule (CR): Context rules are the expressions that are evaluated to decide whether access to associated object will be allowed or denied. CR is composed of two terms: context and action. Action can be allowed or denied. CR is represented as follows:

$CR_i = <(ContextName, Operator, Value), Action>$ where $CR_i \in CR$.

Multiple Context Rules (MCR): For some of the cases, access to an object can be related to many contexts which needs to be checked, so many context rules are required. Multiple context rules enable to write combined context rules by using logical operators like and ($\wedge$), or ($\vee$). MCR is stored in the form of:

$MCR_i = <((CR_1 \wedge CR_2) \vee CR_3), Action>$ where $MCR_i \in MCR$ and $CR_j \in CR$.

4.1.2. Attribute Assignment

Attribute Assignment Table (AAT) includes mapping between attributes and subjects/objects. For simplicity, both SA assignment to subjects and OA assignment to objects are held in the same table. Both of the subjects and the objects can have multiple attributes. The records of the AAT consist of; (S_j, Att_i) or (O_j, Att_i) where $Att_i \in SA$ or

OA and $S_j \in S$, similarly $O_j \in O$. Figure 4.1 given below shows the relation between the entities and the attributes.

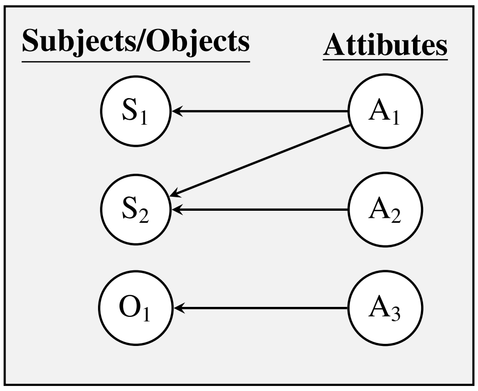


Figure 4.1. Attribute Assignment to Subjects and Objects

4.1.3. Subject Attribute - Operation Assignment

Subject attribute assignment to operation is used for to get easy administration by grouping the subject attributes (implicitly subjects) under allowed operations. Subject attribute - operation assignment (SAOA) table stores data as follows; (Op_k, SA_t) where $SA_t \in SA$ and $Op_k \in Op$. Figure 4.2 shows the relationship between the subject attributes and operations.

As seen in the Figure 4.2 a many-to-many relationship exist between subject attributes and operations: SAOA $\subseteq$ *SA* X *Op*. This means that, a SA can be allowed to perform more than one operation, so can be assigned to different operations. Similarly, one operation may contain many subject attributes.

4.1.4. Object Attribute - Operation Assignment

This access control model designed to facilitate the use of different authentication methods. Some objects may be accessed by using different authentications, whereas accessing to an object requiring high security can be limited with only a specific authentication method. Therefore, authentication methods are related with object attributes

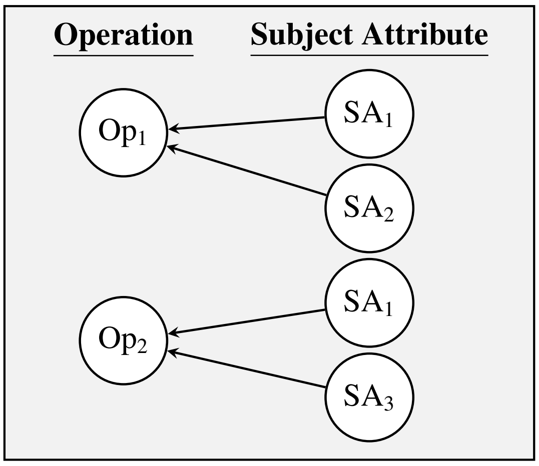


Figure 4.2. SA - Operation Assignment

and operations. Object Attribute - Operation Assignment (OAOA) table is consisted of 4 columns which are operations, object attributes, context and authentication. Each record should include 4 elements; (Op_u, $Auth_z$, OA_v, Con_y) where $Op_u \in Op$, $Auth_z \in Auth$, $OA_v \in OA$ and $Con_y \in Con$. Figure 4.3 illustrates the relationship between operation, OA, authentication and context rule.

As the Figure 4.3 illustrates, Op_1 is allowed to access objects that only have OA_1, however these objects can be reached by using different authentication methods with each having distinct contexts. Op_2 can not be performed on objects in group OA_1, but it can access to objects having OA_2 and OA_3. While OA_1 and OA_2 accept the access requests by using different authentication methods associated with various contexts, access to OA_3 can only be allowed through a specific authentication. Some of the object attributes have no context associated with it, which means that all subjects having SA that are assigned to related operation are granted to perform that operation on requested OA. Allowing to utility of various kinds of authentication methods is enabled to access control being flexible and fine-grained.

4.1.5. Working Principle of Model

Operations, subject attributes, object attributes and context is being checked step by step in Figure 4.4.

Algorithm 1 Working principle algorithm

```
Input: Access_ Request (Request<Subjectid, Objectid, Operation, AuthType, ContextInformation>)
Output: Access
 1: $Operations \leftarrow List\_Operations$
 2: if (Op) in list Operations then        ▷ Check if requested operation defined in the list
 3:     $SA \leftarrow get\_attributes(Sid)$
 4: else
 5:     $Access = Deny$
 6: end if
 7: $SA\_List \leftarrow List\_Subject\_Attibutes(Op)$        ▷ Get the list of the subject attributes defined for that operation
 8: if (SA) in list SA_ List then
 9:     $OA \leftarrow get\_attributes(Oid)$        ▷ Get the object attributes of the requested object
10: else
11:     $Access = Deny$
12: end if
13: $OA\_List \leftarrow List\_Object\_Attibutes(AuthType, Op)$        ▷ Get the object attributes with provided Authentication type and operation
14: if ((OA) in list OA_ List) then
15:     $Context \leftarrow context(AuthType, Op, OA)$        ▷ If we can find any object attribute get the context information with provided authentication type and operation
16: else
17:     $Access = Deny$
18: end if
19: $Result \leftarrow evaluate(Context, ContextInformation)$        ▷ Evaluate stored context and provided context information if it is a match
20: if Result = False then
21:     $Access = Deny$
22:     return$(RequestDenied)$
23: else
24:     $Access = Grant$
25:     return$(RequestGranted)$
26: end if
```

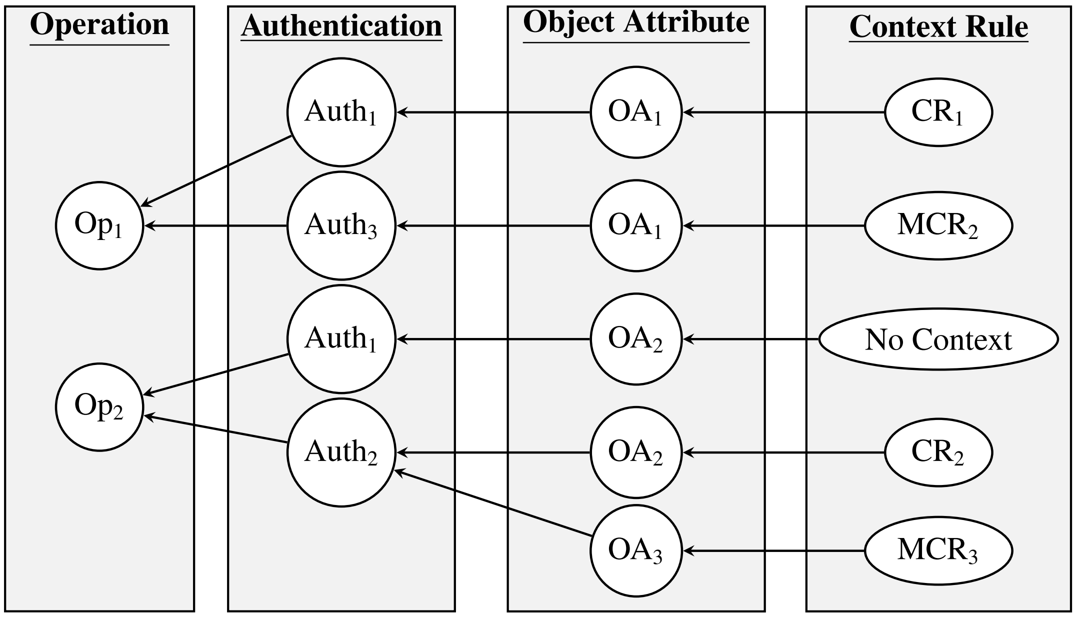


Figure 4.3. OA - Operation Assignment

The working principle flow diagram of the designed model is given in Figure 4.4 to show the general idea of the execution sequence.

Login is required to use our application. Providing correct username and password to login endpoint returns a bearer access token. After receiving access token, every request should be done with this token. When a user sends a command, firstly it is being checked by the authorization module for the correct token. After validating the token we can proceed with the algorithm.

The following technologies have been used for this project: Asp.NET Core, React.Js, EntityFramework, Docker, Fiware.

The backend of the project is written in Asp.NET Core. Asp.NET is a widely used technology in web development. Core framework is designed to work on non-Windows machines too.

EntityFramework is an object-relational mapper for .NET developers to work with

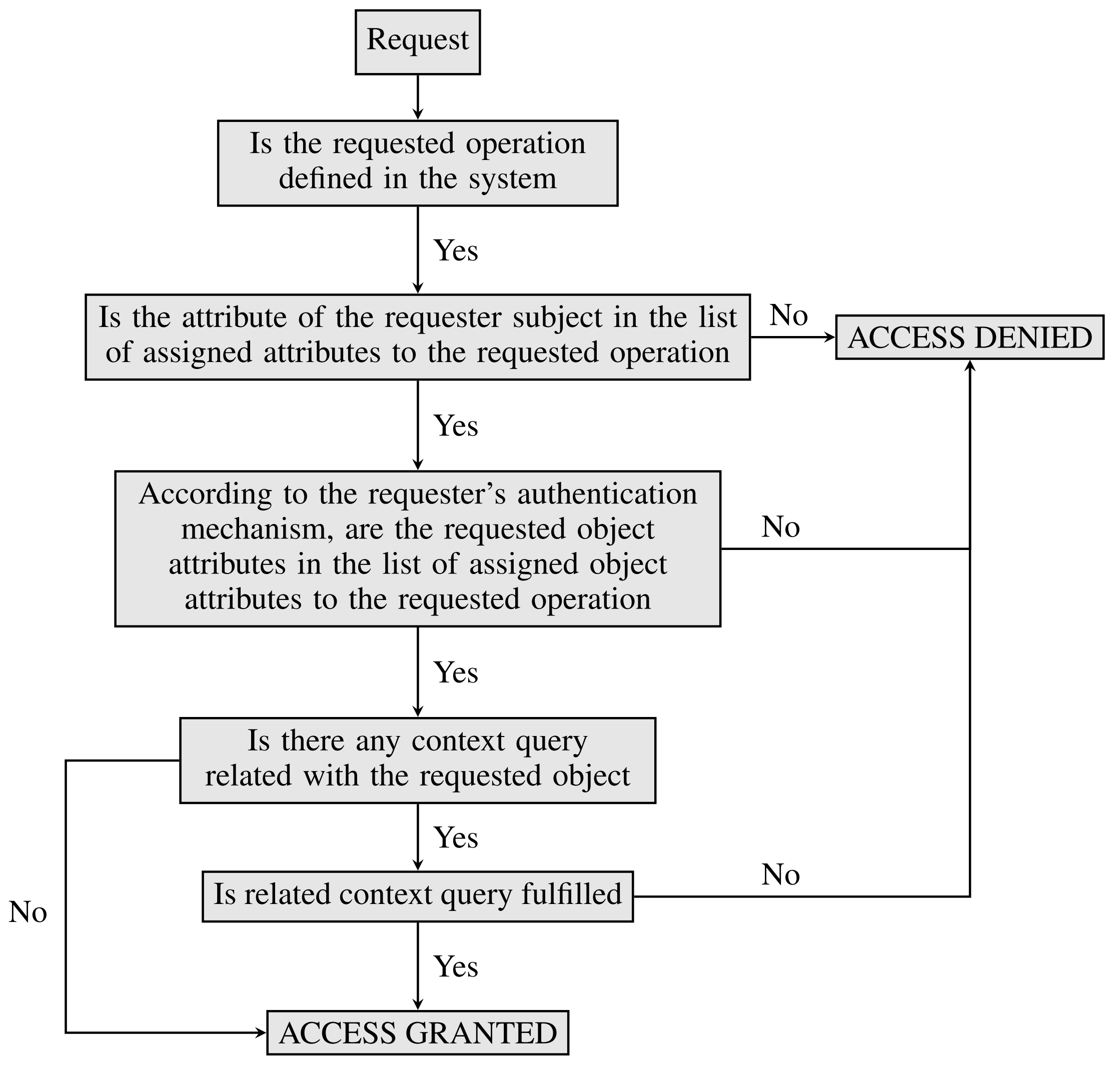

Request
Is the requested operation defined in the system
Yes
Is the attribute of the requester subject in the list of assigned attributes to the requested operation
No
ACCESS DENIED
Yes
According to the requester's authentication mechanism, are the requested object attributes in the list of assigned object attributes to the requested operation
No
Yes
Is there any context query related with the requested object
Yes
Is related context query fulfilled
No
No
Yes
ACCESS GRANTED

Figure 4.4. Flow Diagram of Working Principle

database with .NET objects.

React.Js is a javascript library for building user interfaces. React.Js allows to create single page applications.

Docker is an application that performs operating-system-level virtualization. In this project we run fiware on docker for isolation.

4.2. Scenario

For testing purposes we come up with a scenario. All the access control methods compared in this paper (RBAC, ABAC and OBAC) are evaluated using this scenario. There are 8 subjects and 6 objects, we compared access control methods under various contexts.

Subjects:

1. Katie (mother)
2. John (father)
3. James (child)
4. Joe (child)
5. Sue (child)
6. Jessica (babysitter)
7. Smart Home Application
8. Smart Healthcare Application

Objects:

1. Smart Door
2. Oven
3. Washing Machine
4. Dish Washer
5. Camera
6. Wearable Insulin Pump

Subject Attributes:

- Title: Parent, Children, Babysitter, Healthcare App, Home App

Object Attributes:

- Object Type: Smart Door, Camera, Household Appliances, Wearable Devices

Environmental Attributes:

- Authentication: Mobile Device, Biometric
- Time: school hours, working hours
- Location: inside house, outside house
- Distance
- Parent's Approval: yes, no
- Somebody in front of door: yes, no
- Emergency: yes, no

Object type and *title* are the defined attributes for objects and subjects respectively. Each object and subject should have at least one attribute. There are 7 environmental attributes defined to achieve fine-grained access control. Distance attribute is defined as atomic valued, and the others are defined as set type attributes. Principals are free to use biometric or mobile device authentication. Subjects included are both person and application. The scenario is explained in detailed below:
Smart Door: The smart door can be opened in given cases by predefined subjects via different authentication type usage:

- In all cases, mother and father can open the smart door by using biometric authentication.
- Mother and father can also open the smart door by using their mobile device if their smart car, that is defined in the system, is at less than 10 meters to the house, and the time is outside the working hours.
- If the children are outside the house, in all situations, they can open the smart door by using biometric authentication. However, if they are inside the house, to open the smart door some adult(at least one of the parents, or babysitter) should be inside the house.
- When the school bus is closer than 10 meters to the house, and the time is outside the school hours, the door can be opened by children using their mobile devices.

- Babysitter can open the smart door using only biometric authentication if he/she is outside the house and time is working hours for her. When babysitter is inside the house and there is somebody in front of the door, parent's approval is required to be able to open the door. However, in the same situation, if nobody is outside the house, parent's approval is not needed to open the door.

- Smart home application can also send request to open the smart door if emergency situation context is asserted, and the ambulance is close to the house less than 10 meters away.

Household Appliances:

- Mother and father can turn the oven, dish washer and washing machine on using their mobile devices when they are not inside the house.

- Children are not allowed to turn any electrical household appliances on or off.

- Babysitter can turn the electrical household appliances on by using his/her mobile device if he/she is inside the house, and the time is within the working hours.

- Mother, father and smart home application can turn the electrical household appliances off, if 30 minutes passed since turn them on request has been made, and parent or babysitter is not inside the house.

Camera:

- Camera can be viewed by people who have parent attribute via biometric authentication.

- Mother and father can view camera using their mobile devices if the emergency situation context is enabled.

- Smart home application can also view camera in case of emergency.

Wearable Insulin Pump:

- Smart healthcare application can read data from wearable insulin pump at all times since no context is assigned to this object.

- Smart home application can also read data from wearable devices in emergency situations to verify the context.

4.2.1. RBAC implementation of the scenario

For RBAC implementation of the scenario, each subject have roles, and the roles have corresponding permission sets. Role members are given below the role names in the Table 4.1. The permissions consist of object and the operation. The context terms are assigned to the permissions in this model. The 'Access' column shows the results of evaluation in case of fulfilling the context. Rule set is given in Table 4.1 and Table 4.2.

Table 4.1. Rule set of CA-RBAC

Roles	Permission	Context	Access
PARENT **Katie** **John**	Open the smart door	Authentication=biometric	ALLOW
	Open the smart door	Authentication=mobile device $\wedge$ time>working hour $\wedge$ distance(car)<10m	ALLOW
	Surveillance camera	Authentication=biometric	ALLOW
	Surveillance camera	Authentication=mobile device $\wedge$ emergency=yes	ALLOW
	Turn the oven on	Authentication= mobile device $\wedge$ loc(requestor)= ¬inside house	ALLOW
	Turn the oven off	Authentication= mobile device $\wedge$ (loc(requestor) $\vee$ loc(Jessica-babysitter))= ¬inside house $\wedge$ time(req-turn the oven on) - time(current)>30min	ALLOW
	Turn the washing machine on	Authentication= mobile device $\wedge$ loc(requestor)= ¬inside house	ALLOW
	Turn the dish washer on	Authentication= mobile device $\wedge$ loc(requestor)= ¬inside house	ALLOW
CHILDREN **James** **Joe** **Sue**	Open the smart door	Authentication=biometric $\wedge$ loc(requestor)=outside house	ALLOW
	Open the smart door	Authentication=biometric $\wedge$ loc(requestor)=inside house $\wedge$ (loc(Katie-mom) $\vee$ loc(John-dad) $\vee$ loc(Jessica-babysitter)=inside house $\vee$ emergency=yes)	ALLOW

The RBAC scenario rule set continues in Table 4.2 below.

The roles, operations, objects and the contexts are shown in Table 4.3 below.

4.2.2. ABAC implementation of the scenario

Rule sets for ABAC has been designed as follows: *<Context, Operation, Access>*
Matching between the subject's context and the specified context within the evaluated rule, the result which is stated in 'Access' parameter, that can be allow or deny, will be returned to the subject. Rule engine searches the rules until finding the contexts that are matching with the subject's. When it finds such a match between the contexts, if the access parameter is 'deny', it denies the request, or if the access parameter is 'allow', it allows the request. Rule set of ABAC is given below in Table 4.4.

Complexity analysis of the ABAC model is given below Table 4.5.

4.2.3. OBAC implementation of the scenario

In the OBAC implementation roles are replaced with operations. Each subjects, who are allowed to perform an operation, have attributes and these attributes are assigned to corresponding operations. Object attributes are assigned to operations that indicate the actions allowing to perform on related object.

The security policies of OBAC model is given in Table 4.6.

The related attributes, contexts and operations are shown in Table 4.7.

Table 4.2. Rule set of CA-RBAC (cont.)

CHILDREN	Open the smart door	Authentication=biometric ∧ loc(requestor)=outside house	ALLOW
BABYSITTER **Jessica**	Turn the dish washer on	Authentication= mobile device ∧ loc(requestor)=inside house ∧ time=working hour	ALLOW
	Turn the oven on	Authentication= mobile device ∧ loc(requestor)=inside house ∧ time=working hour	ALLOW
	Turn the washing machine on	Authentication= mobile device ∧ loc(requestor)=inside house ∧ time=working hour	ALLOW
	Open the smart door	Authentication=biometric ∧ loc(requestor)=inside house ∧ sb. in front of door=no	ALLOW
	Open the smart door	Authentication=biometric ∧ loc(requestor)=outside house ∧ time=working hour	ALLOW
	Open the smart door	Authentication=biometric ∧ loc(requestor)=inside house ∧ sb. in front of door=yes ∧ parent's approval=yes	ALLOW
SMART HOME APPLICATION **Home app.**	Turn the oven off	Authentication=mobile device ∧ time(req-turn the oven on) - time(current))>30 min	ALLOW
	Open the smart door	Authentication=mobile device ∧ distance(ambulance)<10m ∧ emergency=yes	ALLOW
	Surveillance camera	Authentication=mobile device ∧ emergency=yes	ALLOW
	Data read from insulin pump	Authentication=mobile device ∧ emergency=yes	ALLOW
SMART HEALTHCARE APPLICATION **Healthcare app.**	Data read from insulin pump	No context	ALLOW

Table 4.3. Complexity Analysis of CA-RBAC Model

Number of Created Roles	5	Parent, Babysitter, Children, Smart Home App., Smart Healthcare App.
Number of Operations	3	Data Read, Open, Turn off
Number of Objects	6	Smart door, Camera, Washing Machine, Dishwasher, Oven, Insulin Pump
Number of Contexts	13	Authentication: Biometric, Mobile Device Distance Parent's Approval: yes, no Sb. in front of door: yes, no Location: inside house, outside house Time: working hour, school hour Emergency: yes, no

Table 4.4. Rule set of ABAC

RULES
<(SA=parent ∧ OA=smart door ∧ auth=biometric), open, allow>
<(SA=children ∧ OA=smart door ∧ auth=biometric ∧ ((loc(requestor) ∧ (loc(parent) ∨ loc(babysitter)))=inside house ∨ emergency=yes), open, allow>
<SA=children ∧ OA=smart door ∧ auth=biometric ∧ loc(requestor)=outside house, open, allow>
<SA=babysitter ∧ OA=smart door ∧ auth=biometric ∧ loc(requestor)=outside house ∧ time=working hour, open, allow>
<SA=babysitter ∧ OA=smart door ∧ auth=biometric ∧ loc(requestor)=inside house ∧ sb. in front of door=yes ∧ parent’s approval=yes, open, allow>
<SA=babysitter ∧ OA=smart door ∧ auth=biometric ∧ loc(requestor)=inside house ∧ sb. in front of door=no, open, allow>
<SA=home app. ∧ OA=smart door ∧ auth=mobile device ∧ distance(ambulance)<10m ∧ emergency=yes, open, allow>
<SA=children ∧ OA=smart door ∧ auth=mobile device ∧ time>school hour ∧ distance(schoolbus)<10m, open, allow>
<SA=parent ∧ OA=household appliances ∧ auth=mobile device ∧ loc(requestor)=¬inside house, open, allow>
<SA=babysitter ∧ OA=household appliances ∧ auth=mobile device ∧ loc(requestor)=inside house ∧ time=working hour, open, allow>
<SA=parent ∧ OA=camera ∧ auth=biometric, data read, allow>
<SA=parent ∧ OA=camera ∧ auth=mobile device ∧ emergency=yes, data read, allow>
<SA=home app. ∧ OA=camera ∧ auth=mobile device ∧ emergency=yes, data read, allow>
<SA=home app. ∧ OA=wearable devices ∧ auth=mobile device ∧ emergency=yes, data read, allow>
<SA=healthcare app. ∧ OA=wearable devices ∧ auth=mobile device, data read, allow>
<(SA=home app. ∨ parent) ∧ OA=household applicances ∧ auth=mobile device ∧ (time(req-open the household appliances) - time(current))>30min∧ loc(requestor) ∨ loc(babysitter)=¬inside house, turn off, allow>
<SA=parent ∧ OA=smart door ∧ auth=mobile device ∧ time>working hour ∧ distance(car)<10m, open, allow>

Table 4.5. Complexity Analysis of ABAC Model

Number of Subject Attributes	5	Parent, Babysitter, Children, Smart Home App., Smart Healthcare App.
Number of Object Attributes	4	Smart door, Camera, Household Appliances, Wearable Devices
Number of Environmental Attributes	13	Authentication: Biometric, Mobile Device Distance Parent's Approval: yes, no Sb. in front of door: yes, no Location: inside house, outside house Time: working hour, school hour Emergency: yes, no
Number of Operations	3	Data Read, Open, Turn off

Table 4.6. Rule set of CA-IRBAC

Operation	Auth.	OA	Context	Access
OPEN: **Parent** **Babysitter** **Home App** **Children**	Biometric	Smart Door	SA= parent	ALLOW
			SA=children ∧ loc(requestor)=outside house	ALLOW
			SA=children ∧ (loc(requestor)=inside house ∧ loc(parent) ∨ loc(babysitter)=inside house) ∨ context=emergency	ALLOW
			SA=babysitter ∧ loc(requestor)=outside house ∧ time=working hour	ALLOW
			SA=babysitter ∧ loc(requestor)=inside house ∧ sb. in front of door=yes ∨ parent's approval=yes	ALLOW
			SA=babysitter ∧ loc(requestor)=inside house ∧ sb. in front of door=no	ALLOW
	Mobile Device	Smart Door	SA=Home app ∧ distance(ambulance)<10m ∨ emergency=yes	ALLOW
			SA=parent ∧ time>working hour ∧ distance(car)<10m	ALLOW
			SA=child ∧ time>school time ∧ distance(schoolbus)<10m	ALLOW
		Household Appliances	SA=parent ∧ loc(requestor)=¬inside house	ALLOW
			SA=babysitter ∧ loc(requestor)=inside house ∧ time=working hour	ALLOW
DATA READ: **Parent** **Home App.** **Healthcare App.**	Biometric	Camera	SA=parent	ALLOW
	Mobile Device	Camera	SA=Home app. ∧ context=emergency	ALLOW
			SA=parent ∧ context=emergency	ALLOW
		Wearable Devices	SA=Healthcare app.	ALLOW
			SA=Home app. ∧ context=emergency	ALLOW
TURN OFF: **Home App.** **Parent**	Mobile Device	Household Appliances	(time(req-open the household appliances) - time(current)) >30 min ∧ (loc(requestor) ∨ loc(babysitter)=¬inside house)	ALLOW

Table 4.7. Complexity Analysis of CA-IRBAC Model

Authentications	2	Biometric, Mobile Device
Number of Object Attributes	4	Smart door, Camera, Household Appliances, Wearable Devices
Number of Contexts	16	Subject Attributes: Parent, Children, Babysitter, Home app., Healthcare app. Distance Parent's Approval: yes, no Sb. in front of door: yes, no Location: inside house, outside house Time: working hour, school hour Emergency: yes, no
Number of Operations	3	Data Read, Open, Turn off

CHAPTER 5

RESULTS

5.1. Results

Section 5.1.1 represents the execution times in millisecond. Execution time measurement is not deterministic, so we run the same cases for 100 times each and calculated the average. There are four cases for each of the access control methods. We have used four keywords. "Accept" means grant access. "Deny" means deny access. "Complex" means context with at least three parameters. These can be authentication type, location of the requester, time of the request, etc. "Simple" means context with one parameter which is authentication type.

5.1.1. Latency

	Accept Complex	Accept Simple	Deny Complex	Deny Simple
RBAC	107.8	84.8	106.2	90.5
ABAC	773.0	27.4	647.2	30.7
OBAC	104.3	32.6	110.7	29.9

Table 5.1. Test case's average execution time in milliseconds

We can interpret following from the results:

1. RBAC is better than ABAC when handling complex context information

2. ABAC is better than RBAC when handling simple context information

In this project we aimed to use both access control method's advantages. Being fast and flexible at the same time.

- RBAC was faster with complex context, our app showed similiar execution times.
- ABAC was faster with simple context, our app showed similiar execution times.

Therefore we managed to overcome ABAC's complex context issue and RBAC's limited usage.

5.1.2. Sustainability

We asked two questions for sustainability:

1. How many steps are required to find out one person's whole rights?
2. How many steps are required to create a new rule?

	Question 1	Question 2
RBAC	2	1
ABAC	2	5
OBAC	2	2

Table 5.2. Number of steps for questions defined above

Let us check for each access control method:

1. RBAC
   (a) Two steps required. First get user's permissions, second get role's permissions which user in it.
   (b) One step required. Either create a row into user's permissions or role's permissions.
2. ABAC
   (a) Two steps required. Find user's role and get all operations linked to that role.
   (b) Five steps required. Operation, Role, Object attribute and Context need to be defined and then all of them need to be linked into one rule.
3. OBAC

(a) Two steps required. First get user’s role, then get that role’s permissions.

(b) Two step required. Create the related context and link with permission and user’s role with it.

With OBAC we tried to achieve as easy as RBAC with as flexible as ABAC. Latency and sustainability results show that we achieved that goal.

CHAPTER 6

FUTURE WORK

6.1. Future Work

The following sections are ideas for taking this work further.

6.1.1. Real World Implementation

IoT devices were simulated with Fiware. As a future work we can implement the same system in real world. We may not predict everything from simulation. Real world implementation may bring unexpected directions for the project.

6.1.2. Another Access Control Method Addition

Using middlewares was the key approach for this project. This way helped to switch between access control methods really quick. This approach also follows open-close relationship. Open for extension close for modification. Another access control method not mentioned in this project can be implemented and included in the comparison.

REFERENCES

Amazon AWS user management. `https://docs.aws.amazon.com/IAM/latest/UserGuide/id.html`. Accessed: 2019-09-02.

Azure Active Directory management. `https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-manage-groups`. Accessed: 2019-09-02.

Google Cloud identity management. `https://cloud.google.com/iam/docs/overview`. Accessed: 2019-09-02.

Abdella, J., M. Özuysal, and E. Tomur (2016). Ca-arbac: privacy preserving using context-aware role-based access control on android permission system. *Security and Communication Networks 9*(18), 5977–5995.

Abowd, G. D., A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles (1999). Towards a better understanding of context and context-awareness. In *International symposium on handheld and ubiquitous computing*, pp. 304–307. Springer.

Al-Muhtadi, J., A. Ranganathan, R. Campbell, and M. D. Mickunas (2003). Cerberus: a context-aware security scheme for smart spaces. In *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*, pp. 489–496. IEEE.

Bai, G., L. Yan, L. Gu, Y. Guo, and X. Chen (2014). Context-aware usage control for web of things. *Security and Communication Networks 7*(12), 2696–2712.

Covington, M. J., P. Fogla, Z. Zhan, and M. Ahamad (2002). A context-aware security architecture for emerging applications. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pp. 249–258. IEEE.

Covington, M. J., W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd (2001). Securing context-aware applications using environment roles. In *Proceed-*

ings of the sixth ACM symposium on Access control models and technologies, pp. 10–20. ACM.

Dr Stanford-Clark, A. and A. Nipper (1999). Message Queuing Telemetry Transport. `http://mqtt.org`.

Genc, D. (2018). Context aware role based access control model for internet of things applications.

Hasiba, B. A., L. Kahloul, and S. Benharzallah (2017). A new hybrid access control model for multi-domain systems. In *Control, Decision and Information Technologies (CoDIT), 2017 4th International Conference on*, pp. 0766–0771. IEEE.

Jin, X., R. Krishnan, and R. Sandhu (2012). A unified attribute-based access control model covering dac, mac and rbac. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pp. 41–55. Springer.

Jin, X., R. Sandhu, and R. Krishnan (2012). Rabac: role-centric attribute-based access control. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pp. 84–96. Springer.

Kulkarni, D. and A. Tripathi (2008). Context-aware role-based access control in pervasive computing systems. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, pp. 113–122. ACM.

Morgan, J. (2014). A simple explanation of'the internet of things'. *Retrieved November 20*, 2015.

Qi, H., H. Ma, J. Li, and X. Di (2015). Access control model based on role and attribute and its applications on space-ground integration networks. In *Computer Science and Network Technology (ICCSNT), 2015 4th International Conference on*, Volume 1, pp. 1118–1122. IEEE.

Rajpoot, Q. M., C. D. Jensen, and R. Krishnan (2015). Attributes enhanced role-based access control model. In *International Conference on Trust and Privacy in Digital*

Business, pp. 3–17. Springer.

Sandhu, R., D. Ferraiolo, R. Kuhn, et al. (2000). The nist model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, Volume 2000, pp. 1–11.

Schilit, B., N. Adams, and R. Want (1994). Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, pp. 85–90. IEEE.

The OWASP IoT Security Team, T. (2018). Owasp Internet of Things Vulnerabilities Top 10 2018. `https://www.networkworld.com/article/3332032/top-10-iot-vulnerabilities.html`.

Weber, H. A. (2003). Role-based access control: the nist solution. *SANS institute InfoSec Reading Room*.

APPENDIX A

CORE PART OF THE CODE BASE

A.1. Managers

Managers are the decision makers of the access controls. All access control methods have one in the project.

A.1.1. Abac Authorization Manager

```
using System;
using System.Linq;
using ContextEngine;
using Data;
using Definition.Authentication;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;

namespace Authorization
{
    public class AbacAuthorizationManager
    {
        private readonly WebDbContext _dbContext;
        private readonly UserManager<AppIdentityUser>
            _userManager;
        public AbacAuthorizationManager(WebDbContext
            dbContext, UserManager<AppIdentityUser>
            userManager)
        {
```

```
        _dbContext = dbContext;
        _userManager = userManager;
    }

    public bool CanAccess(string requestpath, string
       username)
    {
        var pathModel = PathService.SeperateRequest(
           requestpath);

        var foundObject = _dbContext.Objects.Include(o
           =>o.ObjectAttributes).FirstOrDefault(o=>o.
           Name.Equals(pathModel.ObjectName,
           StringComparison.InvariantCultureIgnoreCase)
           );

        var foundOperation = _dbContext.Operations.
           FirstOrDefault(o =>
             o.Name.Equals(pathModel.Action,
                StringComparison.
                InvariantCultureIgnoreCase));

        var foundRoles = _userManager.GetRolesAsync(
           _userManager.FindByNameAsync(username).
           Result).Result;

        var ruleSet = _dbContext.RuleSets.Include(rs=>
           rs.Role)
             .Include(rs=>rs.ObjectAttribute)
             .Include(rs=>rs.Context).Include(rs=>rs.
                Operation).ToList();

        ruleSet = ruleSet.Where(rs => foundRoles.
           Contains(rs.Role.Name)).ToList();
```

```
            ruleSet = ruleSet.Where(rs=>rs.ObjectAttribute.
                Id.Equals(foundObject.ObjectAttributes.
                FirstOrDefault().ObjectAttributeId)).ToList
                ();

            ruleSet = ruleSet.Where(rs=>rs.Operation.Id.
                Equals(foundOperation.Id)).ToList();

            foreach (var rule in ruleSet)
            {
                if (ObjectComparer.IsSame(pathModel.Context
                    , rule.Context))
                    return true;
            }
            return false;
        }
    }
}
```

A.1.2. Rbac Authorization Manager

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using ContextEngine;
using Data;
using Definition.Authentication;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Repository.Interfaces;
```

```
namespace Authorization
{
    public class RbacAuthorizationManager
    {
        private readonly WebDbContext _webDbContext;
        private readonly UserManager<AppIdentityUser>
           _userManager;
        private readonly RoleManager<AppIdentityRole>
           _roleManager;

        public RbacAuthorizationManager(WebDbContext
           dbContext, RoleManager<AppIdentityRole>
           roleManager, UserManager<AppIdentityUser>
           userManager)
        {
            _webDbContext = dbContext;
            _userManager = userManager;
            _roleManager = roleManager;
        }

        public bool CanAccess(string requestpath,string
           username)
        {
            var pathModel = PathService.SeperateRequest(
               requestpath);
            var usersRole = _userManager.GetRolesAsync(
               _userManager.FindByNameAsync(username).
               Result).Result;

            var contexts = _webDbContext.ContextAwareRoles.
               Where(car =>
                    car.Permission.Name.Equals(pathModel.
                       Action, StringComparison.
                       InvariantCultureIgnoreCase) &&
                        usersRole.Contains(car.Role.Name))
                    .Include(car => car.Role)
                    .Include(car => car.Context)
                    .Include(car => car.Permission).ToList();

                foreach (var context in contexts)
                {
                    if (ObjectComparer.IsSame(context.Context,
                        pathModel.Context))
                    {
                        return true;
                    }
                }

                return false;
            }
        }
}
```

A.1.3. Obac Authorization Manager

```
using System.Collections.Generic;
using System.Linq;
using ContextEngine;
using Data;
using Definition.Authentication;
using Definition.Authentication.ABAC;
using Microsoft.AspNetCore.Identity;

namespace Authorization
{
    public class OpbacAuthorizationManager
    {
```

```
public bool CanAccess(string requestpath, string
   username, WebDbContext dbContext,
     UserManager<AppIdentityUser> userManager,
        RoleManager<AppIdentityRole> roleManager)
{
     var requiredParameters = PathService.
        SeperateRequest(requestpath);

     //check operation exist
     var operation = dbContext.Operations.
        FirstOrDefault(o => o.Name.Equals(
        requiredParameters.Action));
     if (operation == null)
          return false;

     //check operation exist in roleOperations table
     var roleOperations = dbContext.Set<
        RoleOperation>().Where(ro => ro.OperationId.
        Equals(operation.Id)).ToList();
     if (!roleOperations.Any())
          return false;

     var roleListFromOperations = new List<string>()
        ;
     foreach (var roleOperation in roleOperations)
     {
          roleListFromOperations.Add(roleManager.
             FindByIdAsync(roleOperation.RoleId).
             Result.Name);
     }

     var user = userManager.FindByNameAsync(username
        ).Result;
     var roles = userManager.GetRolesAsync(user).
        Result;
     var roleIntersectionResult = roles.Intersect(
        roleListFromOperations);

     if (!roleIntersectionResult.Any())
          return false;

     var foundObject = dbContext.Objects.
        FirstOrDefault(o => o.Name.Equals(
        requiredParameters.ObjectName));
     var foundObjectObjectAttributes = dbContext.Set
        <ObjectObjectAttribute>()
          .Where(oa => oa.ObjectId.Equals(foundObject
             .Id)).ToList();
     var foundObjectAttributes = new List<
        ObjectAttribute>();
     foreach (var objectObjectAttribute in
        foundObjectObjectAttributes)
     {
          foundObjectAttributes.Add(dbContext.
             ObjectAttributes.FirstOrDefault(oa =>
              oa.Id.Equals(objectObjectAttribute.
                ObjectAttributeId)));
     }

     var operationObjectAttributes = dbContext.Set<
        OperationObjectAttribute>()
          .Where(ooa => ooa.OperationId.Equals(
             operation.Id)).ToList();
     var operationObjectAttributeList = new List<
        ObjectAttribute>();

     foreach (var operationObjectAttribute in
        operationObjectAttributes)
```

```
            {
                operationObjectAttributeList.Add(dbContext.
                    ObjectAttributes.FirstOrDefault(oa =>
                    oa.Id.Equals(operationObjectAttribute.
                        ObjectAttributeId)));
            }

            var objectIntersectionResult =
                foundObjectAttributes.Intersect(
                operationObjectAttributeList);
            if (!objectIntersectionResult.Any())
                return false;

            return ObjectComparer.IsSame(requiredParameters
                .Context, ContextProvider.GetContext());
        }
    }
}
```

A.2. Authorization middlewares

Authorization in this project has been used as middlewares. The following files are the implementation of each method.

A.2.1. Abac Authorization Middleware

```
using System;
using System.Threading.Tasks;
using Authorization;
using Data;
using Definition.Authentication;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;

namespace Web.Authorization
{
    public class AbacAuthorizationMiddleware
    {
        private readonly RequestDelegate _next;

        public AbacAuthorizationMiddleware(RequestDelegate
           next)
        {
            _next = next;
        }

        public async Task Invoke(HttpContext context,
           WebDbContext dbContext, UserManager<
           AppIdentityUser> userManager)
        {
            var requestPath = context.Request.Path;
            if (!PageHelper.IsFrontEnd(requestPath))
            {
                var token = context.Request.Headers["
                   Authorization"].ToString().Replace("
                   Bearer␣", "");
                if (string.IsNullOrEmpty(token))
                {
                    context.Response.StatusCode = 403;
                    return;
                }

                var tokenService = new TokenService();
                var username = tokenService.Authenticate(
                   token);
                if (string.IsNullOrEmpty(username))
                {
                    context.Response.StatusCode = 403;
                    return;
                }

                var authorizationManager = new
                  AbacAuthorizationManager(dbContext,
                  userManager);
                if (authorizationManager.CanAccess(
                  requestPath, username))
                {
                    await _next(context);
                }
                else
                {
                    context.Response.StatusCode = 403;
                }
            }
            else
            {
                await _next(context);
            }
        }
    }
}
```

A.2.2. Rbac Authorization Middleware

```
using System.Threading.Tasks;
using Authorization;
using Data;
using Definition.Authentication;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;
using Repository.Interfaces;

namespace Web.Authorization
{
    public class RbacAuthorizationMiddleware
    {
        private readonly RequestDelegate _next;

        public RbacAuthorizationMiddleware(RequestDelegate
           next)
        {
            _next = next;
        }

        public async Task Invoke(HttpContext context,
           WebDbContext dbContext,RoleManager<
           AppIdentityRole> roleManager,UserManager<
           AppIdentityUser> userManager)
        {
            var requestPath = context.Request.Path;
            if (!PageHelper.IsFrontEnd(requestPath))
            {
                var token = context.Request.Headers["
                   Authorization"].ToString().Replace("
                   Bearer␣", "");
                if (string.IsNullOrEmpty(token))
                {
                    context.Response.StatusCode = 403;
                    return;
                }

                var tokenService = new TokenService();
                var username = tokenService.Authenticate(
                    token);

                if (string.IsNullOrEmpty(username))
                {
                    context.Response.StatusCode = 403;
                    return;
                }

                var authorizationManager =
                    new RbacAuthorizationManager(dbContext,
                       roleManager, userManager);

                if (authorizationManager.CanAccess(
                    requestPath, username))
                {
                    await _next(context);
                }
                else
                {
                    context.Response.StatusCode = 403;
                }
            }
            else
            {
                await _next(context);
            }
        }
    }
}
```

A.2.3. Obac Authorization Middleware

```
using System;
using System.Threading.Tasks;
using Authorization;
using Data;
using Definition.Authentication;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Identity;

namespace Web.Authorization
{
    public class OpbacAuthorizationMiddleware
    {
        private readonly RequestDelegate _next;

        public OpbacAuthorizationMiddleware(RequestDelegate
            next)
        {
            _next = next;
        }

        public async Task Invoke(HttpContext context,
           WebDbContext dbContext, UserManager<
           AppIdentityUser> userManager,
            RoleManager<AppIdentityRole> roleManager)
        {
            var requestPath = context.Request.Path;
            if (!PageHelper.IsFrontEnd(requestPath))
            {
                var token = context.Request.Headers["
                   Authorization"].ToString().Replace("
                   Bearer␣", "");
                if (string.IsNullOrEmpty(token))
                {
                    context.Response.StatusCode = 403;
```

```
                        return;
                    }

                    var tokenService = new TokenService();
                    var username = tokenService.Authenticate(
                        token);

                    if (string.IsNullOrEmpty(username))
                    {
                        context.Response.StatusCode = 403;
                        return;
                    }

                    var authorizationManager = new
                        OpbacAuthorizationManager();

                    if (authorizationManager.CanAccess(
                        requestPath, username, dbContext,
                        userManager, roleManager))
                    {
                        await _next(context);
                    }
                    else
                    {
                        context.Response.StatusCode = 403;
                    }
                }
                else
                {
                    await _next(context);
                }
            }
        }
    }
```