CITY IoT

Vipuvoimaa EU:lta 2014–2020

Euroopan unioni
Euroopan aluekehitysrahasto

CityIoT reference platform access control

Version 1.0

Mikko Nurminen (mikko.nurminen@tuni.fi)

6Aika · Uudenmaan liitto Nylands förbund · OULU · TAMPERE · OULUN YLIOPISTO · OAMK Oulun ammattikorkeakoulu · Tampereen yliopisto Tampere University

Table of Contents

Introduction

In building the reference platform (hereafter Platform) of the CityIoT (the Project) the access control mechanisms were first built to meet the immediate needs of the first pilot projects. Later a more fine-grained access control was needed, and the Project also wanted to evaluate FIWARE's own access control components. Consequently, a study was conducted with three goals:

- Understand and report the limitations of the current reference platform in relation to the CityIoT requirements specification [1][2]
- Understand and report the functionality and limitations of FIWARE access control in the light of CityIoT requirements [3]
- Demonstrate the capabilities of FIWARE-based access controls by implementing a *FIWARE-based access control test system*

This document is organized as follows. First a high level view of the access control systems is given. Then the current nginx-based access control as running in the Platform is described. After this the access control system based on FIWARE components is described as it is running in out "*FIWARE-based access control test system".* Then, the document compares these two different access control mechanisms. Two earlier documents are used as basis of comparison: one that defined the requirements and use cases for access control [1], and second mapped these use cases against the capabilities of the FIWARE components[2]. Finally, a plan is offered on how to move forward with the access control mechanisms in the Platform.

This document concentrates on the mechanisms for access control. Different access control policies may appear in example use cases, and the document implies implementability of some policies, However, this document does not define or recommend any access control policies.

High level view of the access control

Abstractly speaking the we assume the following requirements:

- For each data-set the access can be defined as public or the access can be limited to certain users. The user can be a human user or an application that works on behalf and trust of some human user.
- Owner of the data can control the access by adding or removing access rights for certain users.
- Three types of access can be controlled: read, modify and delete.

The above requirements imply the following:

- System should have different user roles: two of the most central ones being the U*ser of the system and its data* and the O*wner of the data*. In addition, the *owner of the running IoT Platform* has control and responsibility of the system, and the *System administrator* role is responsible for the day-to-day operations of the Platform. These roles have been described and explored in an earlier document [2, "User roles in the Platform"].
- There needs to be an authentication service that the users in all roles use. Also, if an external authentication system is not used, a user management system is needed.
- The system should support separation of data sets in reasonable level granularity. The finest granularity level that some data owners could require is discussed in an earlier document [2, section "Access control models and the needed access control rule precision"].

Typically the *owner of the data* grants specific users permission to read, modify and/or delete specific data. It is the responsibility of the *owner of the running IoT Platform* to ensure that there are suitable security and access control mechanisms in the Platform, that a reasonable level of granularity of access control can be achieved. In FIWARE-based systems this works differently, by default everything is open. To be able to label a system as "Powered by FIWARE" only requires using one FIWARE component: Orion Context Broker. Orion does not have any access control mechanisms, so to have a working access control setup other components need to be used. These components can be FIWARE's own access control components, or other software like nginx, or an API gateway like Kong, or other suitable access control component.

The actual End User is different from the *User of the system and its data.*. The typical way an end user would consume data from the Platform would be by using an Web application that is developed by a third-party. Users in the *User of the system and its data* role is typically someone representing that third-party, and only in very rare cases would the end user directly communicate with the Platform, and assume the *User of the*

system and its data role. The third-party applications use the Platform, but the relation between the end users and Platform depend on the case. See the application cases in below.

Two categories of applications can be identified, which have different characteristics, which lead to differing communication flows with the Platform. These categories are:

1. Internal applications that run “on the Platform”. These applications may be implemented with technologies such as Grafana. Internal applications typically use the identity service of the Platform. Since these applications are installed “on the Platform”, the access controls deal with accessing the application rather than application accessing Orion and Quantum Leap. These applications are run in the Platform and are started on behest of the the Owner of the running IoT platform. In the current Platform setup Grafana and Wirecloud use their own login mechanisms, and not the Platform's authentication mechanisms.
2. External third-party applications that run elsewhere and access the Platform through the external interface. Thus, these applications need a key to access the Platform. The user base of these applications might be completely separate from the one in the Platform. For example:
    a. The application is completely open for the public but uses data with a limited access. It might be that use of the data requires paying, but the application collects its income through ads. Or, the platform- and data owners trust the application and its vendor to filter sensitive information out.
    b. The application might have its own user base, its own billing etc. The data provided by the Platforms is just one source of input.

When this document discusses access control, 2. category applications are the focus, unless explicitly otherwise stated.

Access control by example

To exemplify the access control systems we assume a case of two buildings where the collected data includes electricity consumption and water consumption, as is shown in Image 1. IBuildings A and B have been instrumented with sensors that send energy and water consumption measurements to a running instance of the Platform.

The data of the buildings have three users in our example: Leenu, Liinu and Tiinu. They are in the role of the u*ser of the system and its data*, and they all must have access to some subset of the data, while access to other data sets must be denied. Table 1 shows what data the users should be able to access.

*Table 1. Users’ access to data sets. In the table “*X*” means that access is permitted, while “-” means it is denied.*

System \ User	Leenu	Liinu	Tiinu
Water consumption - Building A	-	X	X
Water consumption - Building B	-	X	-
Electricity consumption - Building A	X	-	X
Electricity consumption - Building B	X	-	-

In other words, Leenu has access to electricity consumption of both buildings, Liinu has access to water consumption of both buildings, and Tiinu has full access to building A. Thus, we have three partially overlapping data access regions as shown in Image 1:

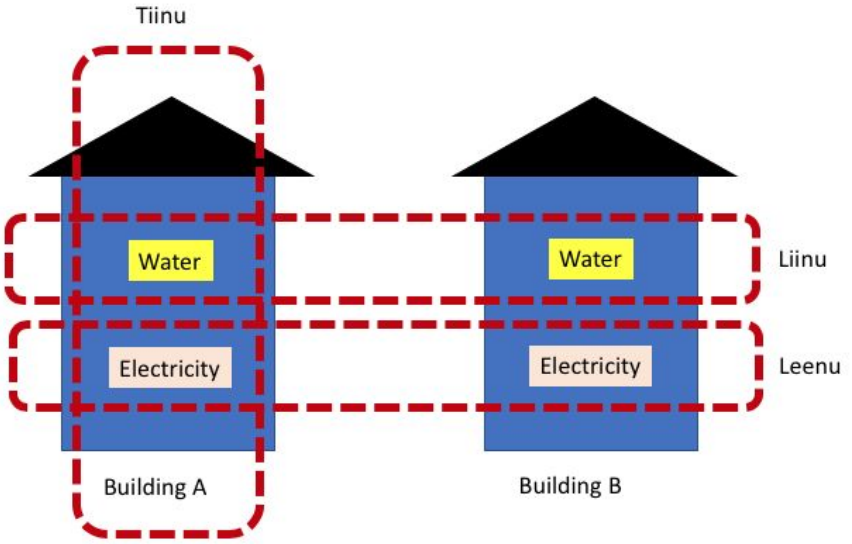


Image 1. Example data: data access regions, and the users who can access them

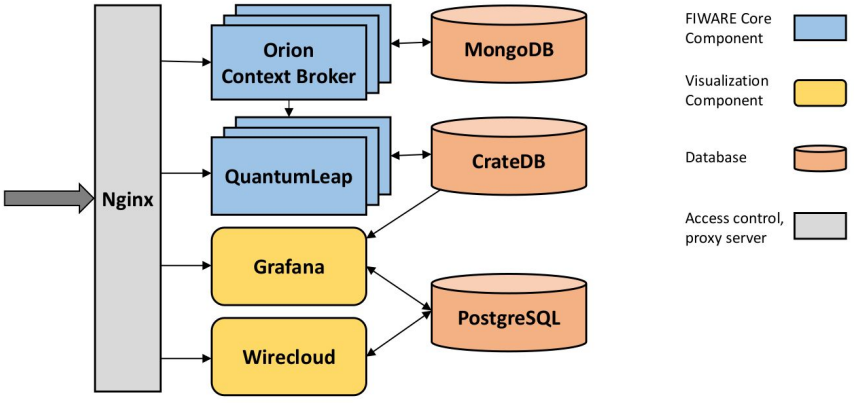
As will be shown below, managing data access even in this example is non-obvious. The way of organizing the data in the Platform affects what type of access control rules can be created. Two different approaches for organizing data will be explored next, namely the hierarchical and service approaches.

FIWARE components and the two approaches

The architecture of the current Platform's nginx-based access system is shown in image 2. Of the components shown, three are FIWARE components: Orion Context Broker, Quantum Leap and Wirecloud. Orion Context Broker and Quantum Leap are of special interest in this document, as these components are the most crucial to the Platform. From Orion Context Broker the user/application can query available resources and read the latest value of measurements. In addition, the client can create subscriptions to data (when data changes the client application receives a notification). From Quantum Leap users/applications can fetch history data about the earlier measured values for the resources.

FIWARE components implement NGSI APIs, Orion currently supports newer NGSIv2 and previous NGSIv1, while Quantum Leap only supports NGSIv2 (NGSIv2: http://telefonicaid.github.io/fiware-orion/api/v2/stable/). In NGSIv2 the data is modeled using entities, which have attributes, and optional metadata connected to attributes. An entity represents the thing that is modelled. In the Leena, Liinu, and Tiinu example the entity to model could be chosen to be the building. The entities must have an *ID* attribute, and with FIWARE's implementation of NGSIv2 it could have a *type* attribute which indicates which data model the entity is an instance of. Other attributes can be added to the entities as are needed. Metadata can be used to describe an attribute further. An example of a JSON entity representation from NGSIv2 documentation is shown below:

```
{
  "type": "Room",
  "id": "Bcn-Welt",
  "temperature": {
    "value": 21.7
  },
  "location": {
    "value": "41.3763726, 2.1864475",
    "type": "geo:point",
    "metadata": {
      "crs": {
        "value": "WGS84"
      }
    }
  }
}
```



NGSIv2 API and the extensions FIWARE has made to it enable using the Service Approach, as well as the Hierarchical Approach within a service. FIWARE components have mechanisms for users to create and use services called hereafter *FIWARE services*. When the Owner of the data or the User of the Platform and its data adds data to the Platform, they include a *Fiware-Service* header in their request to create and name a *FIWARE service*. If the *FIWARE service* identified by the header already exists, sent data is inserted under it.

Fiware-Servicepath header can be used to create a data hierarchy within a *FIWARE service*. In the example of Leena, Liinu, and Tiinu under a FIWARE service named “Buildings” there could be FIWARE service paths “Building_A” and “Building_B” for each building. This would collect the data from both buildings under the same FIWARE service, but under separate FIWARE service paths.

So, *Fiware-Service* header is used to define *FIWARE services* as is needed in the Service Approach, and *Fiware-Servicepath* header can be used to organize data adhering to Hierarchical Approach. If data is saved to FIWARE with no FIWARE service and FIWARE service path, it is stored in the default FIWARE service and FIWARE service path, which both can be queried without specifying any Fiware-Service or Fiware-Servicepath header.

At least some FIWARE developers have expressed views that using FIWARE service paths should be avoided (see Appendix B). This is because with using FIWARE service paths “ you may end up having two entities in different service paths with the same id”, and “another issue is that when you query entities the result set does not provide any information about the service path that provided the data”. So, while the Fiware-Servicepath can be used, it could come with complications. Also, using only Fiware-Service will lead to cleaner implementation of the Service Approach, which likely will lead to a more maintainable service.

An example showing the problem with FIWARE service headers. First the a data entity with the same ID (*"Building_A")* is added to the the same FIWARE service (“cityiot”), but under three different FIWARE service paths (the default empty service path, “/buildings/building_a”, and “/buildings/building_b”. Then the entities are queried using the FIWARE service path wildcard #. The results sent from the Platform include all the three entities, which look identical.

Request: *curl -POST localhost:1026/v2/entities --header "Fiware-service: cityiot" -d '{"id": "Building_A", "type": "Building"}' --header "Content-type: application/json"*
Response: HTTP/1.1 201 Created

Request: *curl -POST localhost:1026/v2/entities --header "Fiware-service: cityiot" --header "Fiware-servicepath: /buildings/building_a" -d '{"id": "Building_A", "type": "Building"}' --header "Content-type: application/json"*
Response: HTTP/1.1 201 Created

Request: *curl -POST localhost:1026/v2/entities --header "Fiware-service: cityiot" --header "Fiware-servicepath: /buildings/building_b" -d '{"id": "Building_A", "type": "Building"}' --header "Content-type: application/json"*
Response: HTTP/1.1 201 Created

Request: *curl localhost:1026/v2/entities --header "Fiware-service: cityiot" --header "Fiware-servicepath: /#"*
Response:
[{"id":"Building_A","type":"Building"},{"id":"Building_A","type":"Building"},{"id":"Building_A","type":"Building"}]

Hierarchical approach to organizing data

In the FIWARE context broker Orion the data is typically organized as hierarchical compositions of several data elements. The selected hierarchy affects how the access to various elements can be organized. In this section two ways of organizing data are discussed. Hierarchical way of organizing data gathered from the buildings A and B could be organized as hierarchical context entities as shown below in diagram 1.

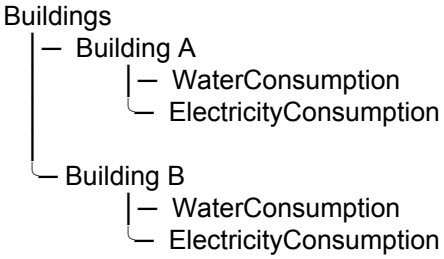


Diagram 1. Hierarchical Approach to organizing data

The approach shown in diagram 1 is called *Hierarchical Approach* in the rest of the document. This approach is suitable for scenarios where the measurements can logically be arranged hierarchically, and can be formed to a tree-like structure similar to diagram 1.

In Hierarchical Approach the access control rules a natural way to control user access is to grant access rights on certain nodes and all sub-nodes in the hierarchy. In this approach the access rules for Tiinu can be easily created, as Tiinu should be able to access all data from Building A. But access rules for Leenu and Liinu cannot be created as easily when using the hierarchy described, as those users should be able to access *some* data from both Building A and Building B, but not *all* data from them. This problem can be solved by either giving access on the consumption data level, or by using the Service Approach, which is described next.

Service approach to organizing data

The NGSI API has been extended by FIWARE supports division of the data into separate *Services* and that separation can be used to implement the access control. This approach is called *Service Approach* in the rest of the documentOne service can store and serve all the data that is needed for a certain application, or a service can be used to store specific types of data, such as water consumption data.

In the building example, the water consumption data from both buildings could be stored in one service, while another service would hold all electricity consumption data. By limiting access by service Leenu can be easily given access only to electricity data, but since Tiinu wants to access both water and electricity consumption data she would need access to these both services. In an alternative approach, a separate service would be needed for each building so that Tiinu can be given access to both water and electricity data for building A. But these building-specific services cannot be used in implementing proper access control for Leenu (who needs to access electricity consumption data) and Liinu (similarly to water consumption data).

It could be possible to have both services but with current implementation will lead to data duplication and increase in data storage needs. Then propagating possible future changes to data in all the services it is stored in requires bookkeeping. This bookkeeping could be left to the owner of the data, but the Platform would be a logical place to have functionality for either automatically recognizing duplicates, or maybe letting

the user who enters the data to mark the duplicates. It might be possible to implement the system so that data is stored only once and the "Service"-header acts as a filter. We have not yet investigated this, but we assume that it would require changes to the implementation FIWARE implementation.

How the access control is implemented in nginx-based and FIWARE-based access control systems and how using FIWARE components and *FIWARE services* affect them are discussed in the next chapters.

Nginx-based access control system

Nginx-based access control mechanism (as shown in Image 2) relies on shared secrets between the user and the Platform. This shared secret is a string of text that is used as an *API key**. The access control system was implemented using nginx. nginx web page sums this software up as: “nginx is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server”.

Using API keys means that users are not authenticated, as anyone who has obtained an API key can use it to access all resources the API key is allowed to access. Even though using API keys is a wide-spread practice with API access controls, not authenticating the users is contrary to the Platform’s requirements.

Nginx is responsible for receiving incoming requests from the client applications, and checking if the API key in the request headers matches any key stored in nginx, and if that API key has the proper access rights for the requested service. If the API key passes all these checks, the user’s request is directed to the service. If we look at the example of Leenu, Liinu, and Tiinu, the water and electricity consumption data from both of the two buildings should be in their own services. This mitigates the fact that Fiware services are the lowest level and highest precision access rules can handle.

The API key management process is not automated. The process involved with a user acquiring a new API key has the following steps:

- the user/3rd party application sends a message to Platform administrators asking for a new API key, so they could access data under specific services, read/write access levels can be enabled
- Platform administrators create the new API key, and set the access rights and levels
- API key is stored in nginx configuration files, and is checked when there are incoming requests
- Platform administrators send the user the new API key

To access a service the user must include the API key in all their requests to the Platform to access services. As the creation of new API keys must be done by the Platform administrators, managing access for a large user base would result in considerable workload to them.

Currently the Platform’s nginx-based access control system only uses the Fiware-Service header in the access control rules it enforces. So, all the data available under a service specified by a *Fiware-Service* header can be accessed by requests that have a valid API key to access that particular service. The value of *Fiware-Servicepath* header is ignored in the checks. This means that the *Fiware-Servicepath* header can only be used for organizing the data, it cannot be used in access control rules to limit users’ access to data, as doing otherwise would expose data to users that have no permission to access that data.

So, in the above example if Leenu (who can access electricity measures of both buildings) wanted to access energy consumption data for Building A, she would be required to use an API key that allows access of electricity data, and provide the Fiware-Service header with value *EnergyConsumption* and Fiware-Servicepath header with value */buildings/A*. Leenu would however be able to access all data under *EnergyConsumption* service, which violates the access regions as shown in the image 1. With nginx-based access rules users are given access to all data within a Fiware-Service, and so those users can access all data within it. Limiting access to just some data within the Fiware-Service isn’t implemented in our current Nginx-based access control system. So, a different access control mechanism is needed for managing hierarchies within a service.

Appendix C lists excerpts from the nginx configuration files in the Platform that are used in defining access rules. These files and the configurations they hold:

- users.conf
    - holds the usernames that are mapped to the incoming requests’ API key

- services.conf
    - maps username and service name combinations to read/write-permissions
- nginx.conf
    - maps incoming request's method to no-access, read, or write permissions
- servers.conf
    - incoming request's path is routed to a service like a FIWARE component

The configuration as presented is not static, but can be extended, if needed.

Access control based on FIWARE components

FIWARE access control components are Keyrock, Wilma, AuthZForce. Their roles in the FIWARE access control are:

- Keyrock provides a way to manage users, and authenticate them. With Keyrock the users can be given access to services based on their roles. Users can get OAuth2 tokens from Keyrock by providing their credentials. Users can use these tokens in applications as long as the token has not been revoked.
- Wilma serves as a security proxy for a specified service, so it intercepts requests from users that are directed to that service. From the user's request Wilma forms an access decision request. This access decision request describes the attributes of the request. Wilma sends the access decision request to AuthZForce.
- AuthZForce stores and manages the created access rules, and is able to make permit/deny access decisions based on these access control rules and the attributes described in the decision requests.

Keyrock is not the only choice available as an authentication server, so OAuth2 Authorization can be federated, but that has been left out of the document.

More in-depth look at the components functioning can be found in another document. Direct links are provided below for the reader's convenience: Keyrock, Wilma, AuthZForce.

With the FIWARE access control components, if a user wants to access certain service, the process of getting an authorization to use data is as follows:

- an *application* representing the service (Orion, Quantum Leap, a 3rd party application) needs to be registered in Keyrock.
- a *role* needs to exist in that application which needs to have permissions to access the wanted service
- a *Keyrock user* needs to exist that represents the user
- this *Keyrock user* needs to be assigned to the role in the
- application

To give an example of the process, let's look at how Liinu would access water consumption data in Orion Context Broker, which is proxied by Wilma. For this to work both Liinu and the owner of the Orion Context Broker instance need to perform certain tasks with FIWARE access control components. A user account for Liinu needs to be created in Keyrock. The owner of the Orion Context Broker instance needs to register the application as an application in Keyrock. In Keyrock the owner of Orion enters the host URL, path, and port from where the Wilma proxy securing the Orion can be accessed. Owner of Orion instance would then create a role in this Keyrock application, and assign the role permissions to access the water consumption data. Liinu would use her credentials to get an OAuth2 token from Keyrock. Liinu would then need to include this OAuth2 token in all requests to the Wilma proxying Orion Context Broker. Wilma communicates with Keyrock and with AuthZForce to find out if Liinu's request should be redirected to Orion. If there is an access control rule in AuhtZForce which states that the combination of roles assigned to Liinu and the request Liinu sent are allowed to access Orion, it will be redirected. If there is no such rule, access will be denied, and an error message sent to the user.

The users are identified by the *OAuth2* tokens which they must include in all requests. The token is specific to a user and a service. In the Building example, if Liinu wanted to access the water consumption data, they would need to be authorized to access a service that includes that data. When sending requests to that service Liinu must first get a valid OAuth2 token from Keyrock by providing their own credentials, and then

include that token in all requests to that service that pass through the FIWARE access control components. Also the required *Fiware-Service* and *Fiware-Servicepath* headers must be in all requests.

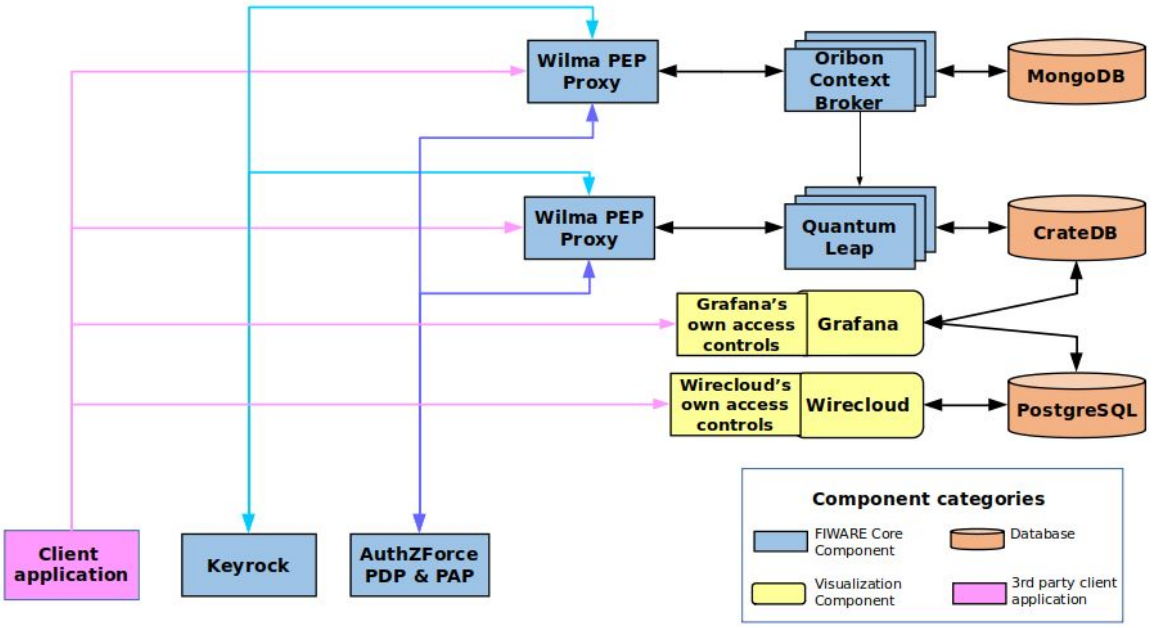


Image 3. Architecture of the FIWARE-based access control test system

Image 3 shows the components and the communication in the *FIWARE-based access control test system*. When compared to the nginx-based system shown in Image 1, it can be seen that on the component level Keyrock, WIlma, and AuthZForce have replaced nginx. There are separate Wilma instances running for each secured service (Orion, Quantum Leap, WireCloud in this image). Compared to nginx-based access control the flow of communication between components is more complex, as the FIWARE components communicate between themselves to create an access control decision for the request from the *Client application*.

Appendix D shows an example of an access rule that can be saved in AuthZForce, and used in the FIWARE-based access control system. It uses XACML, a standard designed for access control. It includes Rules, which can be used to form a permit-or-deny access control decision. XACML is quite verbose, but in summary the rule set in Appendix D declares the following access rules:

- at the highest level this set of these access rules are enforced for every incoming request, checked for all the access decision requests
    - inside this set there are there are 2 separate sets of access rules targeted to an application
        1st set of application access rules
        - these rules for a certain role in the application
        - the following are checked for the incoming request:
            - the HTTP method
            - the path
            - roles
            - header Fiware-Service
            - header Fiware-Servicepath

        2nd set of application access rules
        - otherwise similar to 1st set of application access rules, but includes:
        - checks if request query parameters match certain strings, which can be used as a rudimentary form of limiting queries to certain type of data

The pseudocode that checks the incoming access control decisions would look something like:

for all incoming requests
    if APPLICATION_ID_FROM_REQUEST = APPLICATION_ID_FROM_ACCESS_CONTROL_RULE **and**
    USER_ROLE_ID_FROM_REQUEST = USER_ROLE_FROM_ACCESS_CONTROL_RULE **and**
    HTTP_METHOD_FROM_REQUEST = HTTP_METHOD_FROM_ACCESS_CONTROL_RULE **and**
    FIWARE_SERVICE_FROM_REQUEST =
        FIWARE_SERVICE_FROM_ACCESS_CONTROL_RULE **and**
    FIWARE_SERVICEPATH_FROM_REQUEST =
        FIWARE_SERVICEPATH_FROM_ACCESS_CONTROL_RULE
        permit access to requested resources
    else
        deny access

When someone is creating these access rules, they must have access to the application's data in Keyrock to gather the application's and roles' IDs, and other needed data.

Advantages of integrating FIWARE-based access control to the Platform

In this chapter the benefits of integrating the FIWARE Security components to the Platform are discussed. Using Keyrock and Wilma would provide comparable access controls to nginx-based one. Adding AuthZForce to Keyrock and Wilma would provide the advantages over nginx-based one:

- Managing users, applications and organizations
    - users can add their own services, and manage the users of those services
- OAuth2-token based access control to data assets in the Platform
    - tokens can be for a user or an organization
    - tokens have an expiration date
    - tokens are created and managed by Keyrock automatically
    - organizations and users who add applications to the Platform could manage the access rules in their own applications by themselves
- More detailed access control rules can be created and used with AuthZForce
    - an example of more detailed access rule would be a rule which limits the write rights to a traffic light data to a Traffic Light System Manager role in an application/service that holds traffic data for a city
    - but added complexity in the access rules naturally brings more complexity to creation and management of these rules
    - no tool support in FIWARE components for creation of complex access control rules

When considering the example of users Leenu, Liinu, and Tiinu accessing building data, in the current version of the Platform nginx-based and FIWARE-based access controls can both be used to fill the requirements, but only if data is added to the right services. Using AuthZForce and XACML enables both role and attribute based access control. This means that the access rules can use any attributes that Wilma is modified to capture from requests it receives. Modified Wilma will relay these new attributes to AuthZForce with the other attributes. These possible attributes by default fall under four categories:

1. Subject
   - currently in the Platform the role ID(s) from Keyrock application are used
2. Resource
   - currently the HTTP(S) request's /path (for example /v2/entities), not including the query string
3. Action
   - currently the HTTP method
4. Environment
   - currently these are not used in the Platform

Users can also define their own categories. This flexibility and extensibility gives access rules relatively wide range. But XACML used in the access rules is relatively complicated, and XACML experts seems to be quite rare, which could lead to problems when an organization starts to create the needed finegraned access rules.

Features the FIWARE-based access control test system is missing, or that need to be improved

This section lists the features that are not available, or need improving in *FIWARE-based access control* to be able to meet the requirements of the Project.

- integrated tools for easy creation and validation of access rules (not available at the moment)
- deployment of the components needs manual steps (could be automated)
  - after an application is created in Keyrock, and a Wilma proxy is to be used, the application's information has to be manually inserted into the Docker Compose file where Wilma is configured
- Marketplace functionality, available with FIWARE Business API Ecosystem (BAE) [4]. "*The Business API Ecosystem provides sellers the means for managing, publishing, and generating revenue of their products, apps, data, and services.*"
- as [2] shows demand for access control rules that could describe data access on the level of a data entity attribute, with possible additional conditions applied, for example: certain user role can access certain resources if an attribute's value is higher than specified limit. This is currently beyond the precision of the access rules in FIWARE-based access control test system

Integrating FIWARE access controls to the Platform

Integrating FIWARE-based access controls into the Platform can be achieved in three possible ways:

1. Require a period where the running instances Platform are not used, and deploy FIWARE access control components can be deployed in new instances which replace previous Platform deployments.
2. Leave the current instances of the Platform running as they are, and create new systems that would include the FIWARE access control components. The second option would avoid downtime and possible end-user uproar, but it would lead to parallel running systems, which all require maintenance and development.
3. Include the FIWARE access control components in to the currently running instances of the Platform. In this option requests that include the API key would be checked for authorization using the access rules stored in nginx, while those with a OAuth2 token header would be directed to FIWARE access control components.

Currently there is work going on in the Platform to implement the third option.

Conclusions

Not all CityIoT requirements can be implemented with access control mechanisms implemented in the current Platform. The data in FIWARE is stored under FIWARE Services and FIWARE Servicepaths, and this makes FIWARE Service the level at which access rules operate. More fine-grained access control should be implemented by a trusted application.

Access control and API marketplace

A second goal for the access control and data management in the Platform is to create an API marketplace. This can be done by leveraging the FIWARE access control components and using FIWARE BAE [4] or other similar components to provide a framework for implementing an API marketplace. API marketplace would be a suitable mechanism for distributing and possible monetization of the data stored in the Platform, which would be integrated with FIWARE other components.

Further discussion on the security aspects of the Platform

While this document concentrated on access controls in the Platform, another document created in the Project, “Security Guidelines for CityIoT” [5], discusses a wide array of security concerns pertinent to the Platform. Subject matter discussed there includes:

- Security Requirements of the CityIoT platform
- Identity Management and Access Control
- Secure Communication in CityIoT
- Assessment of Secure Communication
- Data Integrity and Privacy
- Data Integrity
- Data Privacy
- Operational Considerations
- Security Profiles

References

[1] 6Aika: Tulevaisuuden toimijariippumaton dataintegraatioalusta (CityIoT) VAATIMUSMÄÄRITTELY Versio 1.00 [https://drive.google.com/file/d/1B48QHkRFOgvvJdbGV5OyDh-CJU_vK4dP/view?usp=sharing]

[2] IoT platforms, data ownership and data access, version 0.3: [https://drive.google.com/file/d/1vnvedNHD9YfNSCkpnfS7GEid8d_U1G3k/view?usp=sharing]

[3] Mapping CityIoT project's requirements to FIWARE components, version 0.2 [https://drive.google.com/file/d/1qNDJ6EZmPWHv3ttmQ7D2LaFIoDCmUtDn/view?usp=sharing]

[4] Business API Ecosystem https://github.com/FIWARE-TMForum/Business-API-Ecosystem

[5] Security Guidelines for CityIoT [https://drive.google.com/file/d/17tg4hlTv4r1PaOtnmPW8HMfPBxpcPI5N/view?usp=sharing]

Appendix A - FIWARE-based test system

Hostname: http://easi-vm-11.rd.tut.fi/
IP: 130.230.142.107

Available for users with SSH keys in the system

Documentation available within this system:
/tampere-fiware/platform/experiments/fiware_access_control/advanced_access_control/build_wilma_wiht_fiware_header_support.txt
/tampere-fiware/platform/experiments/fiware_access_control/advanced_access_control/creating_xacml_rules_for_fiware_headers.txt

Appendix B. Email discussion about whether Fiware-Servicepath should be used or not

Re: FIWARE servicepath not promoted, but is it going away?

NAME REDACTED - EMAIL REDACTED
13.11.2019. 10.21

Hi,

Service-path is going to remain and you can still use it, up to you.

The main issue of Service Path is that you may end up having two entities in different service paths with same id, which It is weird.

Another issue is that when you query entities the result set does not provide any information about the service path that provided the data, which it is bad.

Thus, those are the caveats around service-path. From my point of view, fiware-service is just enough … and I would avoid service-path as much as possible.

From: Mikko Nurminen (TAU) <mikko.nurminen@tuni.fi>
Monday 12/08/2019

Hello!

In the GitHub issue https://github.com/FIWARE/dataModels/issues/477 you wrote "As we no longer want to promote the usage of FIWARE Service Path, let's get rid of it from the public instances (so the harvesters should not use the fiware-service-path)." Was this only specific to data models, or is servicepath being phased out?

I personally wouldn't miss servicepaths as they are implemented now. But we do sometimes use them at Tampere University when working with FIWARE in CityIoT project, so it would be interesting to know if this is not the recommended way.

Appendix C. nginx access rules in practice

The files used in nginx-based access control in the Platform.

An excerpt from file: /platform/production/secrets/users.conf
This file holds the usernames that are mapped to the incoming requests' API key

```
# map the apikey to the username
map $http_apikey $fiware_user
{
        # when apikey is empty or missing
        default             unknown;
        # normal users
        'user1_api_key_secret'  user1;
        'user2_api_key_secret'  user2;

        # users with only view access
        'view_only_user_secret'             view_only_user;

        # when incorrect apikey is used
        '~*(.)*'            intruder;
}
```

An excerpt from file: /platform/production/secrets/services.conf

```
# use the username and FIWARE service to determine the access rights
# NOTE: empty service name means the default service ''
map $fiware_user:$http_fiware_service $allowed_fiware_access_type
{
        default                     no-access;

        # give the normal users write access to specific services
        user1:service1              write-access;
        user2:service2              write-access;
        user1:service2              read-access;

        # give the normal users read access to everything
        '~^user1:(.)*'              read-access;
        '~^user2:(.)*'              read-access;
        '~^user3:(.)*'              read-access;

        # give the view only users read access to specific services
        view_only_user:service1         read-access;
        view_only_user:service2         read-access;
}
```

An excerpt from file: platform/production/nginx/nginx.conf

```
# load the mapping from the apikey to the username
# $fiware_user will contain the current username
include /run/secrets/users.conf;

# load the mapping from the username and service name to the allowed access type
# $allowed_fiware_access_type will be either 'write-access', 'read-access' or 'no-access'
include /run/secrets/services.conf;
```

```
# map the requests method to either read or write type
map $request_method $request_method_type
{
        default   not_supported;
        GET     read;
        HEAD   read;
        DELETE          write;
        PATCH           write;
        POST   write;
        PUT     write;
}

# check whether FIWARE access is allowed (access allowed if )
map $request_method_type:$allowed_fiware_access_type $fiware_access_allowed
{
        default             'no';

        read:no-access          'no';
        read:read-access        'yes';
        read:write-access       'yes';

        write:no-access         'no';
        write:read-access       'no';
        write:write-access   'yes';
}
```

An excerpt from file: platform/production/nginx/servers.conf

```
# allow checking for the Orion version even without credentials
location = /orion/version/
{
        proxy_pass http://orion:1026/version;
}

# proxy to Orion with authentication
location /orion/
{
        if ($fiware_access_allowed != 'yes')
        {
        return 401 $authentication_error;
        }

        proxy_pass http://orion:1026/;
}
```

Appendix D. An example of a FIWARE-based access rule

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- POLICY SET has all-inclusive TARGET. -->
<ns3:PolicySet xmlns="http://authzforce.github.io/rest-api-model/xmlns/authz/5"
xmlns:ns2="http://authzforce.github.io/pap-dao-flat-file/xmlns/properties/3.6"
xmlns:ns3="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" xmlns:ns4="http://www.w3.org/2005/Atom"
xmlns:ns5="http://authzforce.github.io/core/xmlns/pdp/6.0"
PolicySetId="914e8f3b-fde1-49df-82cd-617450e0e2ec" Version="83"
PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit">
  <ns3:Description>Policy Set for application 1ce2deef-026e-41a7-95bb-9c104d961a90</ns3:Description>
  <!--TARGET for the POLICY SET is selfclosing, so this policy set is for *everything*, and it is used for all the
access decision requests.-->
  <ns3:Target/>
  <!-- POLICY, which is TARGETED to "Securing Orion" application ID. -->
  <ns3:Policy PolicyId="POSTING_with_fiware_headers" Version="1.0"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit">
        <!--TODO: change the role ID to match an existing role that's authorized in the app in Keyrock. -->
        <ns3:Description>Role  from application
1ce2deef-026e-41a7-95bb-9c104d961a90</ns3:Description>
        <!-- TARGET: here Match is made for the application ID. -->
        <ns3:Target>
        <ns3:AnyOf>
        <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <!-- Application ID as a string-->
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">1ce2deef-026e-41a7-95bb-9c104d961a90</ns3:At
tributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </ns3:Match>
        </ns3:AllOf>
        </ns3:AnyOf>
        </ns3:Target>
        <!-- Two Keyrock-generated rules, one for Matching the request's path, and second for Matching
request's HTTP method.-->
        <ns3:Rule RuleId="15614780230160.r6oicq0yp9" Effect="Permit">
        <ns3:Description>POST /v2/entities</ns3:Description>
        <ns3:Target>
        <ns3:AnyOf>
f       <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">/v2/entities</ns3:AttributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
AttributeId="urn:thales:xacml:2.0:resource:sub-resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </ns3:Match>
        </ns3:AllOf>
        </ns3:AnyOf>
        <ns3:AnyOf>
        <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
```

```
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">POST</ns3:AttributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </ns3:Match>
        </ns3:AllOf>
        <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">GET</ns3:AttributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </ns3:Match>
        </ns3:AllOf>
        </ns3:AnyOf>
        !-<- The following two attributes are for the Fiware-Service and Fiware-Servicepath headers.
        Wilma was customed to add them to the access decision request, the changed file is in
platform/experiments/fiware_access_control/fiware_docker_repositories/fiware-pep-proxy/lib/azf.js) -->
        <ns3:AnyOf>
        <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">cityiot</ns3:AttributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
AttributeId="fiware-service" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"
/>
        </ns3:Match>
        </ns3:AllOf>
        </ns3:AnyOf>
        <ns3:AnyOf>
        <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">/cityiot</ns3:AttributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
AttributeId="fiware-servicepath" DataType="http://www.w3.org/2001/XMLSchema#string"
MustBePresent="true" />
        </ns3:Match>
        </ns3:AllOf>
        </ns3:AnyOf>
        </ns3:Target>
        <!-- CONDITION: Role ID is matched to one in access permission request -->
        <ns3:Condition>
        <ns3:Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">
        <ns3:Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">62e0a853-879a-419a-8a28-87fd9b2a17e4</ns3:At
tributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
        </ns3:Apply>
        </ns3:Condition>
        </ns3:Rule>
  </ns3:Policy>
```

```
  <ns3:Policy PolicyId="get-only-certain-types" Version="1.0"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit">
        <!--TODO: change the role to match existing role, that's authorized in the app in Keyrock. -->
        <ns3:Description>Role  from application
1ce2deef-026e-41a7-95bb-9c104d961a90</ns3:Description>
        <!-- TARGET: here Match is made for the application ID. -->
        <ns3:Target>
        <ns3:AnyOf>
        <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <!-- Application ID as a string-->
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">1ce2deef-026e-41a7-95bb-9c104d961a90</ns3:At
tributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </ns3:Match>
        </ns3:AllOf>
        </ns3:AnyOf>
        </ns3:Target>
        <!-- Two Keyrock-generated rules, one for Matching the request's path, and second for Matching
request's HTTP method.-->
        <ns3:Rule RuleId="get_only_the_discos_not_party_animals" Effect="Permit">
        <ns3:Description>Get type Disco entities from Orion</ns3:Description>
        <ns3:Target>
        <ns3:AnyOf>
        <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:3.0:function:string-starts-with">
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">/v2/entities</ns3:AttributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
AttributeId="urn:thales:xacml:2.0:resource:sub-resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </ns3:Match>
        </ns3:AllOf>
        </ns3:AnyOf>
        <ns3:AnyOf>
        <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">GET</ns3:AttributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </ns3:Match>
        </ns3:AllOf>
        </ns3:AnyOf>
        <!-- The following two attributes are for the Fiware-Service and Fiware-Servicepath headers.
        Wilma was customed to add them to the access decision request, the changed file is in
platform/experiments/fiware_access_control/fiware_docker_repositories/fiware-pep-proxy/lib/azf.js) -->
        <ns3:AnyOf>
        <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">testing123</ns3:AttributeValue>
```

```
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
AttributeId="fiware-service" DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"
/>
        </ns3:Match>
        </ns3:AllOf>
        </ns3:AnyOf>
        <ns3:AnyOf>
        <ns3:AllOf>
        <ns3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">/this/is/it</ns3:AttributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
AttributeId="fiware-servicepath" DataType="http://www.w3.org/2001/XMLSchema#string"
MustBePresent="true" />
        </ns3:Match>
        </ns3:AllOf>
        </ns3:AnyOf>
        </ns3:Target>
        <!-- CONDITION: Role ID is matched to one in access permission request -->
        <ns3:Condition>
        <ns3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
        <ns3:Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">
        <ns3:Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">62e0a853-879a-419a-8a28-87fd9b2a17e4</ns3:At
tributeValue>
        <ns3:AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
        </ns3:Apply>
        <ns3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
        <ns3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">type=Disco</ns3:AttributeValue>
        <ns3:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"></ns3:AttributeValue>
        <ns3:AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">type=NotHere</ns3:AttributeValue>
        </ns3:Apply>
        <ns3:AttributeDesignator AttributeId="query_string_parameters"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="false"/>
        </ns3:Apply>
        </ns3:Apply>
        </ns3:Condition>
        </ns3:Rule>
  </ns3:Policy>
</ns3:PolicySet>
```