Monga, Hitesh Chander; Kinnunen, Tuomas; Malhi, Avleen; Javed, Asad; Främling, Kary

# An OAuth-based Authentication Mechanism for Open Messaging Interface Standard

# An OAuth-based Authentication Mechanism for Open Messaging Interface Standard

Hitesh Chander Monga[1], Tuomas Kinnunen[1], Avleen Malhi[1]ª, Asad Javed[1] and Kary Främling[1,2]

[1]*Department of Computer Science, Aalto University, Finland*
[2]*Department of Computer Science, Umeå University, Sweden*

Keywords: Internet of Things, Open Messaging Interface, Open Data Format, Authentication, Authorization, OAuth2.

Abstract: The Internet of Things (IoT) security and privacy remain a major challenge, especially due to the highly scalable and distributed nature of IoT networks. IoT security has an exceptionally wide scope which also leads to the most demanding requirements needed for the widespread realization of many IoT visions. It includes the tasks such as trusted sensing, computation, communication, privacy in terms of security scope. A large number of industry-driven domain specific standards has been developed which hinder the development of a single IoT ecosystem. O-MI and O-DF standards were previously proposed to address the challenge of interoperability in IoT devices. This paper highlights and discusses the measures taken to enhance the security of these standards by integration of OAuth 2.0 service provider functionality with the O-MI authentication module along with the existing security module. The security architecture has been proposed which enhances the security features by providing authorization without use of certificates. The implementation details of the plug-in module for envisioned security model developed for the O-MI and O-DF standards have been discussed.

## 1 INTRODUCTION

The Internet of Things (IoT) has become an inevitable part of our daily lives by drawing an attention of many researchers and industrialists. The latest press release by Gartner[1] forecasts that the enterprise and automotive IoT market will expand to 5.8 billion endpoints in 2020, which is about 21% increase from 2019. By the end of 2019, around 4.8 billion endpoints are expected to be in use, with an increase of 21.5% from 2018. As IoT deployments increase continuously, the need for handling the authentication and authorization tasks to improve the data security in IoT communications also increase. Recently, many IoT frameworks such as IBM BlueMix, AWS IoT, and Azure IoT have been launched, and most of the industry-driven domain specific standards hinder the development of a single IoT ecosystem. As discussed in (Yousefnezhad et al., 2017), requiring a new web browser for each website would be complicated. In the similar context, it would be hard to imagine the current approach of IoT connectivity. The authors (Yousefnezhad et al.,

2017) state that the IoT community must follow the example of the World Wide Web, in which the unified protocols, TCP/IP and HTTP/HTML boosted the Internet to spread all over the world. The Open Group platform has developed the open messaging standards for solving the interoperability problem in communications among different IoT devices. The Open Messaging Interface (O-MI) and Open Data Format (O-DF) standards are proposed as the application-level interface to make a complete and flexible standard to support a variety of organizational needs and structures (Kubler et al., 2014). The security model for these standards provide the suitable access control and authentication mechanisms. As stated in (Onl, d), the OAuth 2.0 authorization protocol enables a third-party application to obtain restricted access to an HTTP service mainly by two methods, either on behalf of a resource owner or on an end user by setting up an approval interaction for user between the resource owner/end user and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. Various roles, grants and terminologies of the OAuth 2.0 flow have been discussed in Subsection 4.

The main contribution of this paper is to enhance

---

ª https://orcid.org/0000-0002-9303-655X
[1][Online]. Available: https://www.gartner.com/en/newsroom/press-releases/

the existing security for O-MI/O-DF standards, providing better authentication plug-in module, access control, and authorization. We can use this new proposed security module without any external provider, specifically with the OAuth 2.0 authentication protocol. In contrast to the earlier security module which uses client certificates, there is a wider user community for OAuth 2.0 (used in the new version of security module) instead of client certificates to support the new development in IoT domain.

The rest of the paper is organized as follows. Section II describes the background and related work. Section III explains O-MI/O-DF standards along with comparison between various standards, the existing security module and the vision for enhancing it. Section IV presents OAuth 2.0 details and its advantages while Section V explains the proposed security architecture with added security features as well as its implementation details. Section VI presents the business use cases and Section VII provides the discussions for the proposed security plug-in module along with the limitations. Finally, Section VIII concludes the paper with the possible future directions.

## 2 RELATED WORK

Security in IoT implementations is critical and correctly implemented. Secure IoT deployments should ensure the proper configuration of the basic security requirements: data confidentiality,data integrity, and data accessibility as part of the solution. An efficient identity management is a primary security requirement for dynamically assigning and managing unique identities for the large number of objects and users. OAuth 2.0 is an authorization framework that allows third party applications to get protected access to user accounts on an HTTP service. Recently, many research works have focused on security related with Internet of Things and discussed various scenarios for implementing OAuth to be used for Internet of Things. The use of OAuth for IoT networks was first proposed in (Emerson et al., 2015), the proposed approach protects IoT network from unauthenticated users with security manager which manages various kinds of constrained devices using OAuth 2.0 protocol. (Fremantle et al., 2014) explores the use of OAuth for IoT systems and built a prototype that uses OAuth 2.0 to enable access control to information distributed via MQTT. (Sciancalepore et al., 2017) proposed a flexible authentication and authorization framework for the Internet of Things, namely OAuth-IoT which leverages and properly harmonizes existing open-standards while taking into account the

limited capabilities of constrained devices. A configurable and easily integrable proposal was made in (Cirani et al., 2014) which proposes an architecture targeting HTTP/CoAP that provides an authorization framework which can be integrated with an external oauth-based authorization service (OAS) collaboratively making it the IoT-OAS architecture. (Khan et al., 2018) proposed an authentication scheme based on the OAuth 2.0 protocol to secure the IoT network by providing an authentication service. (Fremantle and Aziz, 2016) proposed model for IoT using OAuth 2.0 protocol which allowed the identity of users and devices on the network to be federated. Other lightweight approaches to solve the authentication and authorization problems in the IoT networks have been discussed in (Yao et al., 2013).

This article is an extension of the work done in (Yousefnezhad et al., 2017) to provide suitable access control and authentication mechanisms that could regulate the rights of different principles and operations defined in O-MI/O-DF standards. This paper discusses a major step taken to enhance the security modules and added functionalities in the security module of the O-MI/O-DF standards.

## 3 OPEN MESSAGING STANDARDS

### 3.1 O-MI/O-DF Standards

With the vision of creating a protocol for Internet of Things which serves in the same way as HTTP protocol serves to the Internet, Open Group devised the O-MI/O-DF standards. In other words, O-MI/O-DF standards will offer connectivity between various things and will allow rapid integration of these links with other systems and networks, making it an integrated standard for Internet of things. O-MI provides a communication framework between commodities and distributed information systems that consume and publish information on a real-time basis. It can be seen as a transportation mechanism which allows to exchange information between different O-MI nodes across a network. In O-MI Node, each node can act both as a "server" and as a "client" with other nodes or systems. O-DF is specified as an expandable XML Schema for representing payload in Internet of things applications (Javed et al., 2019) (Madhikermi et al., 2018).

## 3.2 Reference Implementation of O-MI/O-DF Standards

Reference Implementation of O-MI/O-DF standards shows the executable documentation as well as request and response examples which can cover each and every aspect of these standards[2]. The developers can understand with the help of examples and implementations what these standards can do in reality. Current reference implementation includes three modules: *O-MI Node Server*: Server implements all basic O-MI operations and maintains a database where the information about O-DF data model, consisting of all object(s) and InfoItem(s) is stored. *Web-client*: It provides graphical user interface which allows the users or developers to perform the desired operation under the standards by just clicking the defined buttons instead of writing all the XML messages and HTTP queries. *Agent*: The agents are the actors that interact with core of O-MI node programmatically and act as an intermediate between the hardware and O-MI node Since for a application to be fully developed it would certainly require security aspects which can defend against different cyber-crimes and attacks, so there is need to discuss the reference implementation for security sub-modules.

## 3.3 Security of O-MI/O-DF Standards

Though the reference implementation clearly ensures robust development and data standardization, security was always a concern for fully developed application. So as mentioned in (Saeed, 2018), initially the reference implementation had only IP white list mechanism for security, i.e, the trusted IP addresses or their ranges are listed and only these trusted users can access the domain and perform "write" functionality while the "read" functionality was open for all users. Thus, in order to have an appropriately designed security mechanism, a dedicated security model was required and created (the version 1 of security model) which became obsolete since it does not provide any support for O-MI read request permission and it only had specific authentication method implementations. Then the second version (which is the existing version) provides a better solution where Authentication and Authorization module are separated into two individual processes working concurrently. The Authentication sub-module basically registers, authenticates and logins the users for session handling. Three login options are available for the user to authenticate

---

[2]O-MI reference implementation developed by Aalto University, Available: https://github.com/AaltoAsia/O-MI

namely local authentication, Facebook login and login with LDAP (Lightweight Directory Access Protocol) credentials. The authorization sub-module has two parts namely superuser console and the access control module. An administrator console has a user interface which can only be accessed by a superuser. This console handles addition of users to group and manages access policies on the database. On the other hand,the Access control module communicates with the O-MI node, processes requests made by users and authorizes them. However, the security model of O-MI/O-DF standard is required to be updated with more recent security techniques in order to prevent cyber attacks and ensure data security.

The implementation details for the above mentioned sub-modules are stated in (Saeed, 2018) and the existing security module is developed using Python and Scala. Django web framework was used for developing the authentication module and Akka HTTP was also used for construction of security model. The module supports SQL databases and LDAP directory services. Furthermore, JSON Web Tokens were used for the token mechanism. New Authentication and Authorization APIs are flexible and adaptable. These modules were necessary to make the O-MI Node secure which makes the APIs more secure and add extra functionalities which can be beneficial to any organization in terms of security, user identity, analysis and monitoring, better access control, from a variety of other methods discussed in section 3.4.

## 3.4 Comparison between Various Security Standards

A variety of methods to increase the security of O-MI-Authentication sub-module are being analysed in this sub section. The most common web security protocols are OpenID, OAuth 2.0 and SAML. These are standardized protocols and there are generalised implementations and libraries existing in multiple languages to implement these protocols. A comparison between these protocols has been shown in Table 1. These web security protocols support the idea of Single sign-on (SSO) which allows a user to enter one username and password in order to access multiple applications. The three security standards (Lightfoot, 2018) (Onl, e) are discussed along with their usage as mentioned in the following:

**OpenID.** OpenID is an open standard for authentication. The user is authenticated using a third-party service called identity provider in this standard. If authentication is a major concern for security implementation then OpenID is highly recommended than any other security methods. In 2014, OpenID connect was

introduced where OpenID is used for authentication along with OAuth 2.0 being used as authorization.

**SAML.** SAML stands for Security Assertion Markup Language. It is an XML-based markup language for security assertions and is the oldest standard of these all. It's an open standard that provides both authentication and authorization services. This protocol is much more complex to use than other protocols and is vulnerable towards XML signature wrapping to impersonate any user.

**OAuth 2.0.** This is an authorization protocol that allows third party applications to get protected access to user accounts on an HTTP service like Facebook, GitHub, and Google. It is mainly used for accessing delegation via token-based authentication. This is a quite flexible and adaptable mechanism for protected resource view and authorization but lacks a good mechanism for authentication of the user. Phishing is a common security risk to which OAuth 2.0 is vulnerable. Apart from that OAuth 2.0 relies completely on TLS for confidentiality.

# 4 OAuth 2.0 PROTOCOL

## 4.1 OAuth 2.0 Roles

OAuth 2.0 basically defines four major roles as specified in (Onl, d) which are:

**Resource Owner.** An entity that provides access to the protected resources. It can be a person or a machine. According to (Buyya and Dastjerdi, 2016), when the resource owner is a person, it is referred as an end-user. The Resource owner must give authorization to third party applications to access its data.

**Resource Server.** It is the server that hosts protected resources and it has the ability to accept and send responses to protected resources requests using access token. The resource server might be the same server as the authorization server. In O-MI-Authentication module, we used same server to function as resource server and authorization server.

**Client.** Client is an application that makes secured resource requests on behalf of the resource owner and with its authorization specifications. Each client is defined by a unique client_id and a client_secret. These credentials are registered with an authorization server inserted into an OAuth 2.0 Access token and then validated by the resource server which provides additional layer of security. The client can be hosted on a server, desktop, mobile or any other device.

**Authorization Server.** It is the server that maintains the list of registered clients, manages and issues the access token needed for all authorization flows. These

are supported by OAuth 2.0 spec to the client after authenticating the resource owner and obtaining authorization. Usually the same application that offers resources through OAuth 2.0 protected API also behaves like an authorization server.

## 4.2 OAuth 2.0 Grant Types

OAuth 2.0 is a flexible authorization framework which provides a number of methods for client application to obtain access token which is further used for authenticated access to the API endpoint. There are basically four authorization grant types, as mentioned in (Onl, d), namely:

**Authorization Code.** This is the most common grant type used. Here, the authorization code is obtained by an authorization server which acts as an intermediary between the client and the resource owner. Here the client directs the resource owner to an authorization server which then directs the resource owner back to the client with the authorization code which can be further used to obtain a long-lived access token and a refresh token. So, the Authorization server authenticates the resource owner and after authentication, it directs the resource owner back to the client thus adding an additional layer of security. It prevents direct transmission of access token to the client unlike implicit grant.

**Implicit.** The implicit grant is a simplified authorization code flow developed for clients implemented in a browser using scripting language like JavaScript. The grant type is implicit so there are no intermediate credentials and access token is directly issued to the client. Unlike authorization code grant type, client receives the access token in the front-end as a result of the authorization request. In some cases, redirection URL used to deliver the access token to the client are used to verify the client identity. Although, implicit grant improves the responsiveness and efficiency of some clients by removing extra steps of authentication before obtaining the access token, still it is the least secure grant type because the access token is directly exposed on the client side.

**Resource Owner Credentials.** Resource owner password credentials can be used directly to obtain an access token. This method is only used when there is a high level of trust between the client and the resource owner and when other authorization grant types are not viable. By exchanging the credentials with a long-lived access token or refresh token, this grant type can eliminate the need for the resource owner credentials to be stored by the client for future use. This is for what OAuth was created to prevent in the first place, so it is not a recommended method to

Table 1: Comparison between web security protocols.

| Parameters | Web Security Protocols | | |
| --- | --- | --- | --- |
| | *OpenID Connect* | *SAML* | *OAuth 2.0* |
| Token format | JSON, JWT | XML | JSON, XML, JWT |
| Protocols used | JSON, HTTP, REST | XML, HTTP, SOAP | XML, HTTP, REST |
| Main purpose | Single sign-on for consumer applications | Single sign-on for enterprise users | API authorization between applications |
| Provides Authorization | No | Yes | Yes |
| Provides Authentication | Yes | Yes | Pseudo-Authentication |
| User Consent | It collects user consent before sharing attributes | It does not collects users consent | It collects users consents before sharing attribute |
| Security Risks | Phishing since Identity provides have a database of OpenID logins. | XML Signature wrapping to impersonate any user | Phising, OAuth 2.0 does not support encryption & also it relies completely on TLS for confidentiality. |

allow third-party apps to use this grant, but this methods finds its use mainly when the client is developed or works under the same authority as the authorization server i.e it is entitled to great degree of trust.
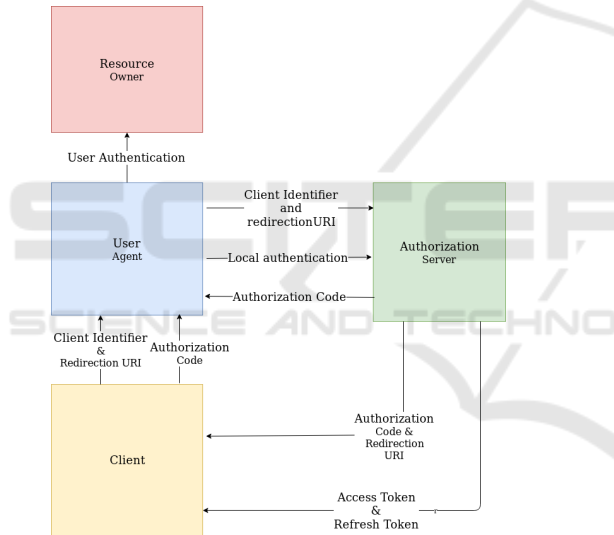


Figure 1: Authorization code grant flow.

**Client Credentials.** The client credentials grant type can be used as an authorization grant when the authorization scope is confined to protected resources under the control of client or under the authorization server. This grant type is used by clients to acquire an access token outside user's context and used by clients to access resources about themselves rather than to access user's resources. Hence the end-user does not have to give its authorization for accessing the resource server.

## 4.3 Advantages of using Authorization Code in O-MI-Authentication rather than Other Grant Types

According to (Onl, d) the authorization code grant flow can be shown in Fig. 1. The authorization code flow is also known as the three-legged OAuth flow since it has three legged scenarios of the application, user and the service provider. So this flow has an extra authorization step as compared to generic 2-legged approach. As stated in (Onl, a) since Authorization code is short lived and that can be further used to obtain a unique access token and a refresh token. When the request for access token is made by the application, the user is authenticated with the client secret, which diminishes the risk of attacker intercepting authorization code.

## 5 OAuth 2.0 SECURITY MODULE FOR O-MI/O-DF STANDARDS

### 5.1 Architecture

The architecture for OAuth 2.0 integration with the O-MI authentication module is shown in the Fig. 2 depicting all the steps undertaken for the proposed model. Fig. 3 illustrates our security module implementation with OAuth 2.0. The implementation steps can be listed as follows:

1. The OAuth 2.0 module authenticates the client and generates a unique token/session for secure communication.

2. The client sends O-MI request with the authentication token/session. This request is then parsed in terms of TTL and request type.
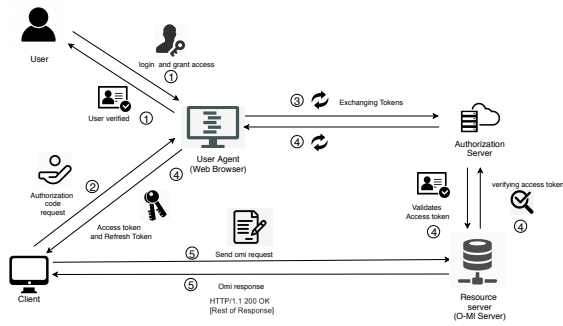
Figure 2: The proposed architecture for OAuth 2.0 integration with O-MI Node.
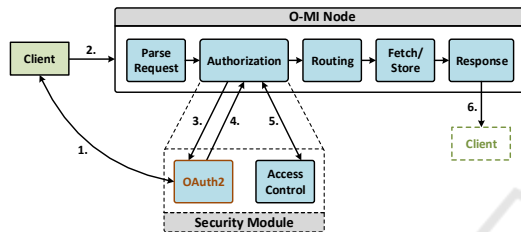


Figure 3: Security module with OAuth 2.0 implementation.

3. The authorization block validates the token with OAuth 2.0 to confirm that the user is authenticated.

4. Once the token has been validated, OAuth 2.0 returns the user ID.

5. The authorization block gets access control rules with user ID and request type. The O-DF request is also filtered based on these rules.

6. The routing block then split the O-DF request to the corresponding handlers. The database is queried in the fetch/store block based on the read or write request. Finally, the results are sent to the client.

OAuth 2.0 implementation and integration with the O-MI authentication module was done using open-source python framework Django. This decision to use Django framework has been made mostly due to availability of libraries to complete the whole OAuth 2.0 flow. Django-oauth-toolkit is used to set up our own Authorization server to issue access tokens to client applications for a certain API, to set up a resource server to protect the views of the application and to authenticate the access token and ultimately the client. The implementation of security module can be found on GitHub[3]. To complete the OAuth 2.0

---

[3]O-MI security module implementation, Available: https://github.com/AaltoAsia/O-MI-Authentication/tree/OAuth2-Service-Provider-Functionality

flow, we used the authorization code grant flow type due to various advantages it offers. Django-allauth library was included to establish the client or the middleware server for the third party application/client side integration for the OAuth flow to be completed and to store and manage the information about access tokens, social accounts, applications, user and group information.

## 5.2 Added Security Features

OAuth 2.0 flow was found the most suitable one to be integrated with the O-MI/O-DF standards. Following reasons were also considered to implement OAuth 2.0 in the O-MI authentication module:

- *API Security:* To make an API secure; one way is to use the basic authentication method similar as the O-MI-Authentication method where the username and password are sent using Base64 encoding. The credentials are not encoded or hashed and the user has to send username and password over the wire. To improve this flow, we use OAuth 2.0 to secure the API, so instead of username and password, an access token is sent which makes the flow more secure. Since in case of any attack if the client application starts misbehaving or using data improperly, access token for that application can be revoked any time thus making all the future requests invalid.

- *Provision for limited access:* Limited access to the application can be also be provided by defining limited scopes in the access token. So when using access tokens, a client application may not have full API access. In contrast, if the credentials are stolen then whole API can be misused.

- *Federal identity:* As discussed in (Onl, b) with OAuth 2.0, an effective mechanism can be applied where once user logs in with username and password (the OAuth 2.0 service provider), he is redirected to the provider in all other applications. He has to confirm that he wants to be authorized and need not to enter his credentials/sign up for other applications.

- *Service analysis and monitoring:* According to (Onl, b) the organizations can monitor their application usage by the consumers and can track, monitor more easily that which access token is making requests. It helps in gaining better insights about customer's behavior and can make optimizations.

## 5.3 Steps of Implementation

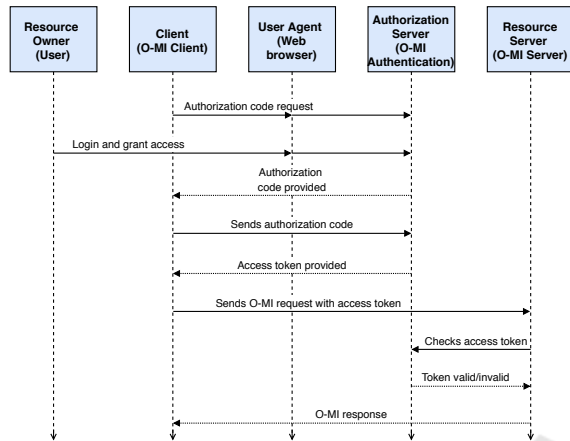The implementation steps adopted for the OAuth integration with O-MI/O-DF standards is depicted in Fig. 4.



Figure 4: The implemented OAuth 2.0 flow for O-MI Node.

**Accessing the O-MI Authentication Module.** The first step of the flow requires the user to provide appropriate credentials to the O-MI Authentication module. If valid user credentials are provided, then the user is redirected to the authentication module interface where the user can obtain his/her own signed and decoded token which can be further used to authenticate the user in the O-MI-Node. If the user does not have valid credentials, he/she can register as a user or super user by using filling up the sign-up form provided on the homepage of authentication module.



Figure 5: Client Registration Interface.

**Application Registration.** Now the user should tell the authorization server which application is allowed to use the OAuth 2.0 Authentication by registering the application on the server. Here the user has to provide information about the application like client type, grant type and the redirect URL. To make this module secure and as per the needs of authentication module, we used the configuration of having client type as confidential client, grant type as authorization code and the redirect URL as the callback address of the client application on running on different server (or the middle-ware server). After registration the user is provided with a client_id and a client_secret which is used to authenticate the application while obtaining the authorization codes, access token and the refresh token. Fig. 5 shows the application interface along with the desired application code flow. Here the user is assigned a unique client ID and secret, the client type is made confidential so that only trusted clients could access the resources of the O-MI node. As discussed in subsection 4.3 due to the mentioned advantages of authorization code flow, the grant type is set to authorization code here. The redirect URI includes the callback URI of the client application which is bound to run on a different domain/server than that of the O-MI authentication module. The following details about the application can be edited/deleted according to the user's need.

**Obtaining Authorization Code.** The authorization server validates the client and continues to complete the flow according to the grant type and the client credentials provided. After validation, it asks the user's permission to provide the demanded scopes to the application. After getting user's permission, authorization server redirects the user agent back to client using the redirection URI provided at the time of application registration. Now after redirection, the redirection URI automatically includes an authorization code and any local state provided earlier by the client as shown in Fig. 6.

**Exchanging Tokens.** After a short lived Authorization code is obtained from the authorization server, now the client requests an access token from the authorization server's token endpoint by exchanging the authorization code and client information with the authorization server. After verification of client and the authorization code, an HTTP response is passed by the authorization server including the access token, its validity period and a refresh token.

**Obtaining the Protected Resource View and Authenticating the Access Token with O-MI Node.** After the access token and refresh tokens are obtained as shown in Fig. 7, the client is OAuth protected i.e

http://127.0.0.1:8888/accounts/customprovider/login/callback/?code=5xpHtIOIXSj725X1sfmvhmC2KkdCi&state=random_state_string

←————————————Authorization code————————————→

Figure 6: Authorization code included in the URL after redirection.

the user can access the protected resource view of the client by providing the access token. Now to complete the whole process, user has to provide the access token to the O-MI-Node which checks with the O-MI-Authentication module and ultimately results with a 200 OK code if the access token provided is valid.

**User Interface.** Our Django social application which runs on a different server must be told about the credentials to be allowed to use the server as an authentication back end and information about the provider must be provided. We used the django-allauth library for developing the client application which provides the flow for sign up of both local and social accounts, allows connection of more than one social account to a local account and supports consistent storage of access token. For the purpose of developing a generalised reference implementation we used our own custom provider, in case we use any external OAuth 2.0 provider like Github, Facebook,etc. We would need the unique client id and secret key which is obtained from respective OAuth 2.0 providers. Fig. 8 shows the interface for social application registration.

{"access_token": "krX8kDLV3gtPeaKbfoZYrRZ2
fCcEDA", "expires_in": 36000, "token_type"
: "Bearer", "scope": "read write", "refres
h_token": "WDMThAAk3VQmeBIg4WkSB4LMGdp59w"

Figure 7: Access token,refresh token and token details obtained from token endpoint.
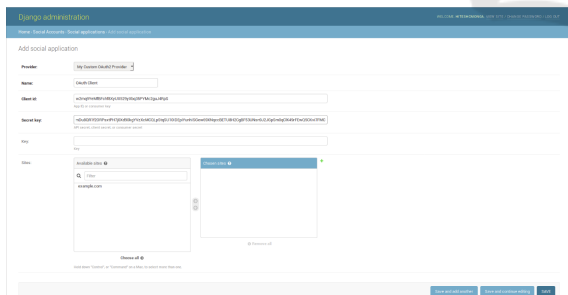


Figure 8: Page for creating/editing django social application.

**Django Site Administration.** The django client application also manages multiple social applications, social accounts and the social tokens. Basic features such as addition/deletion of accounts and tokens are also included within the library, multiple sites can also be added or deleted. The user can keep a track of the recent changes implemented under the *My actions* section. Figure 9 shows the django site administration panel.
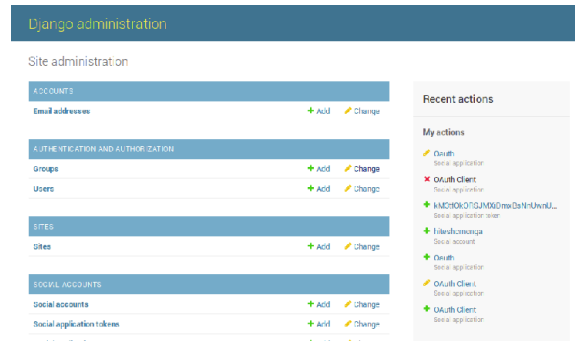


Figure 9: Django site administration panel on client side.

## 6 BUSINESS USE CASES

The O-MI/O-DF standards are used for unification of diverse IoT systems and data streams. It can enable a range of diverse IoT applications, more specifically, three use cases (proof-of-concept) have been developed and showcased[4].

1. Greater Lyon Use case: There are two case studies implemented for Lyon which are associated with essential elements of public policies of metropolitan; heat wave mitigation case study deals with the urban monitoring and the climate change adaptation and bottle banks management case study deals with the optimization of daily public services.

2. The Brussels city use case: This use case is for smart mobility which covers three general pilot cases; Safety Around School use case, waterbus services use case and smart mobility for disabled people.

3. The Helsinki Smart City use case: It is being designed for contribution in creation of an electric vehicles' charging ecosystem. The proposed ecosystem helps in integrating all charging possibilities by providing interoperability between the charging service provider companies and inclusion of other non-specialized possibilities, such as house-holds and private parking places.

Apart from the business use case of the O-MI/O-DF standards, the recently added OAuth integration also has specific use cases (Onl, c) which can used according to the user's need.

---

[4]bIoTope Project, Available: https://biotope-project.eu

Table 2: Comparison between different versions of O-MI security module.

| Functionalities and properties | Version 1 | Version 2 | Modified Version 2 |
|---|---|---|---|
| Request type in O-MI | Write | Read, write, call, delete | Read, write, call, delete |
| Token types | Base64 token | Base64 token | Base64 token and OAuth access token |
| Local user and password | no | yes | yes |
| OAuth 2.0 user login | no | no | yes |
| External Client registration | no | no | yes |
| Protected page or API | no | yes | yes |
| External OAuth 2.0 provider | no | yes | yes |
| OAuth 2.0 service provider | no | yes | yes |
| Http Framework | Jetty | Django, Akka HTTP | Django, Akka HTTP |

1. Multiple Access tokens: This use case is particularly useful for native applications since a web browser needs to be launched only once. And as discussed in 5.2, multiple access tokens provide the functionality of federal identity, which eases the login process for the user.

2. Data monitoring: The user/organisation can give limited access to another user or organisation and can manage the data to be shared. The organisation can monitor the application usage by the consumers and monitor the data usage based on access token requests.

3. Support for native applications: Since OAuth2.0 supports native applications, which makes it easier for the user to log-in and the user does not need to authenticate and authorize access on every execution of the app. This use case can be implied in various ways including in-app-payment methods, data sharing applications, etc.

## 7 DISCUSSION

In the IoT security, data privacy and confidentiality are important aspects to be concerned about. As of now, the security module v2 used so far has two security sub-modules namely authentication and authorization modules for security purposes which had minimalist security features required for an Internet of things network. Though these modules are fully functional modules, still some major functionalities are required to improve the performance and the security of reference implementation is missing. Adding OAuth 2.0 security feature was one of the major steps taken to improve the security modules. Table 2 also compares the newly developing modules with the previous versions.

### Limitations and Vulnerabilities of OAuth

As mentioned in (Emerson et al., 2015) (Lodderstedt et al., 2013) following limitations and threats are being observed for the OAuth 2.0 process flow:

- Addition of extra extensions at the ends in the specification produce a wide range of non-inter operable implementations.

- By eavesdropping the communication between the service provider and user attacks like replay attacks and impersonation attacks are quite common. By eavesdropping, any intruder can steal the authorization code or access token and pretend as a user.

- There are chances that the data can be mishandled by the registered third party application.

- According to (Mangal, 2019), the spec recommends the use of SSL/TLS while transmitting the issued tokens in plain text over the wire, so if the authorization endpoints or the redirection URL is not secure, then the attacker can easily attack the security module.

- In (Chae et al., 2015) authors have discussed about the replay attack as the authorization code is not for single use. Any attacker who captures the authorization code within its lifetime can resend the request to access the resources. Authors have also mentioned the impersonation, phishing and the replay vulnerabilities.

## 8 CONCLUSION AND FUTURE SCOPE

The main focus of this research is to develop an enhanced security model for the Open Messaging Interface (O-MI) and Open Data Format (O-DF) standards by integration with OAuth 2.0 authentication protocol. The integrated security module helps in making the APIs more secure, providing provision for limited access, federal identities and helps in service analysis as well as monitoring. The design and implementation principles of authentication module are explained and the integration with the existing O-MI security

module is presented. Adding OAuth 2.0 security feature was one of the major steps taken to improve the existing security module. However, the requirements, the core design decisions, and the code structure are conceived to be generally applicable to other systems, providing a solid foundation for a further abstraction and generality of the used approach.

Future work will be to implement JSON Web Token (JWT) in the O-MI reference implementation to achieve good performance and portability. With JWT, there is no session to manage as the security information is digitally signed and self-contained in the token, which makes the system stateless. Further, the work will be extended to reduce network round trip time by sharing user ID between OAuth 2.0 and Access Control module, instead of retrieving the ID from authorization module.

# REFERENCES

Authorization Code Grant - OAuth2.0 Servers. https://www.oauth.com/oauth2-servers/server-side-apps/authorization-code/. [Online]; accessed October 2019.

Benefits of OAuth 2.0. https://subscription.packtpub.com/book/application_development/9781783285594/1/ch01lvl1sec09/benefits-of-oauth-2-0. [Online]; accessed October 2019.

OAuth Use Cases. https://tools.ietf.org/html/draft-ietf-oauth-use-cases-01. [Online]; accessed December 2019.

The OAuth 2.0 Authorization Framework. https://tools.ietf.org/html/rfc6749. [Online]; accessed October 2019.

User Authentication with OAuth 2.0. https://oauth.net/articles/authentication/. [Online]; accessed October 2019.

Buyya, R. and Dastjerdi, A. V. (2016). *Internet of Things: Principles and paradigms*. Elsevier.

Chae, C.-J., Choi, K.-N., Choi, K., Yae, Y.-H., and Shin, Y. (2015). The Extended Authentication Protocol using E-mail Authentication in OAuth 2.0 Protocol for Secure Granting of User Access. *Journal of Internet Computing and Services (JICS)*, 16(1):21–28.

Cirani, S., Picone, M., Gonizzi, P., Veltri, L., and Ferrari, G. (2014). IoT-OAS: An OAuth-based Authorization Service Architecture for Secure Services in IoT Scenarios. *IEEE sensors journal*, 15(2):1224–1234.

Emerson, S., Choi, Y., Hwang, D., Kim, K., and Kim, K. (2015). An OAuth based authentication mechanism for IoT networks. In *International Conference on Information and Communication Technology Convergence, ICTC 2015, Jeju Island, South Korea, October 28-30, 2015*, pages 1072–1074.

Fremantle, P. and Aziz, B. (2016). OAuthing: Privacy-enhancing federation for the Internet of Things. In *2016 Cloudification of the Internet of Things, CIoT 2016, Paris, France, November 23-25, 2016*, pages 1–6.

Fremantle, P., Aziz, B., Kopecký, J., and Scott, P. (2014). Federated Identity and Access Management for the Internet of Things. In *2014 International Workshop on Secure Internet of Things, SIoT 2014, Wroclaw, Poland, September 10, 2014*, pages 10–17.

Javed, A., Yousefnezhad, N., Robert, J., Heljanko, K., and Främling, K. (2019). Access Time Improvement Framework for Standardized IoT Gateways. In *IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2019, Kyoto, Japan, March 11-15, 2019*, pages 220–226.

Khan, J., ping Li, J., Ali, I., Parveen, S., ahmad Khan, G., Khalil, M., Khan, A., Haq, A. U., and Shahid, M. (2018). An Authentication Technique Based on Oauth 2.0 Protocol for Internet of Things (IoT) Network. In *2018 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 160–165. IEEE.

Kubler, S., Madhikermi, M., Buda, A., and Främling, K. (2014). QLM Messaging Standards: Introduction and Comparison with Existing Messaging Protocols. In *Service Orientation in Holonic and Multi-Agent Manufacturing and Robotics*, pages 237–256.

Lightfoot, J. (2018). Authentication and authorization: Openid vs oauth2 vs saml. https://spin.atomicobject.com/2016/05/30/openid-oauth-saml/. [Online]; accessed October 2019.

Lodderstedt, T., Mcgloin, M., and Hunt, P. (2013). OAuth 2.0 Threat Model and Security Considerations. *RFC*, 6819:1–71.

Madhikermi, M., Yousefnezhad, N., and Främling, K. (2018). Data Exchange Standard for Industrial Internet of Things. In *2018 3rd International Conference on System Reliability and Safety (ICSRS)*, pages 53–61. IEEE.

Mangal, A. (2019). OAuth 2.0 - The Good, The Bad & The Ugly. https://code.tutsplus.com/articles/oauth-20-the-good-the-bad-the-ugly--net-33216. [Online]; accessed October 2019.

Saeed, A. (2018). Authentication and Authorization Modules for Open Messaging Interface (O-MI). Master's Thesis, Aalto University.

Sciancalepore, S., Piro, G., Caldarola, D., Boggia, G., and Bianchi, G. (2017). OAuth-IoT: An access control framework for the Internet of Things based on open standards. In *2017 IEEE Symposium on Computers and Communications, ISCC 2017, Heraklion, Greece, July 3-6, 2017*, pages 676–681.

Yao, X., Han, X., Du, X., and Zhou, X. (2013). A lightweight multicast authentication mechanism for small scale IoT applications. *IEEE Sensors Journal*, 13(10):3693–3701.

Yousefnezhad, N., Filippov, R., Javed, A., Buda, A., Madhikermi, M., and Främling, K. (2017). Authentication and Access Control for Open Messaging Interface Standard. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Melbourne, Australia, November 7-10, 2017.*, pages 20–27.