Sicurezza delle Architetture Orientate ai Servizi

Progetto SOASEC

MyCSM

Your Cloud Service Manager

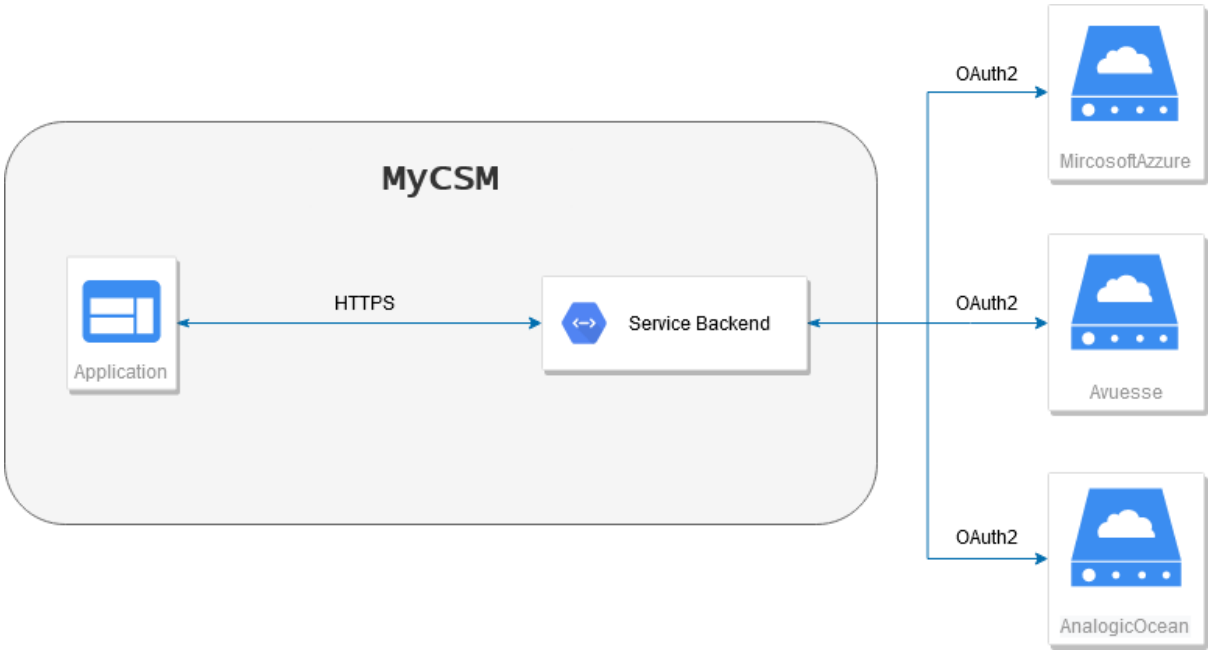
Riccardo Giordano (944397) e Simone Caggese (948590)
UNIVERSITA' DEGLI STUDI DI MILANO

Sommario

Introduction

The project was created to offer the possibility of being able to manage the information and functions of the different cloud providers that offer authentication using the OAuth2 protocol in a single application. Registered users at MyCSM platform can choose between two plans (free or premium) to have limited or full access to the functionalities provided by the application.

Server side, for each call related to a cloud service provider, the application performs an API call to the CSP using the authorization protocol OAuth2.0 to access to the users resource



From a main dashboard, it is possible to have a general overview of the different cloud providers which in turn will have a dedicated page to be able to view the active instances of the different services in more detail, with the possibility of performing some operations restricted to premium profiles.

Cloud Service Providers (CSP) - Resource Server

Each cloud service provider is created according to a model that simulates some features and services that each of them can offer.
For the purpose of emulation of the project, for simplicity it has been chosen to create 3 different CSPs that differ just in graphics, but not in functionality, (clearly it is possible to customize each of these to make them do different things).

To generalize, it was chosen that each service provider can offer 3 different types of services: Management of *Virtual Machines (VM)*, *Databases (DB)* or *Mail servers (MS)*.

Once logged in, the operations available within the platform are:

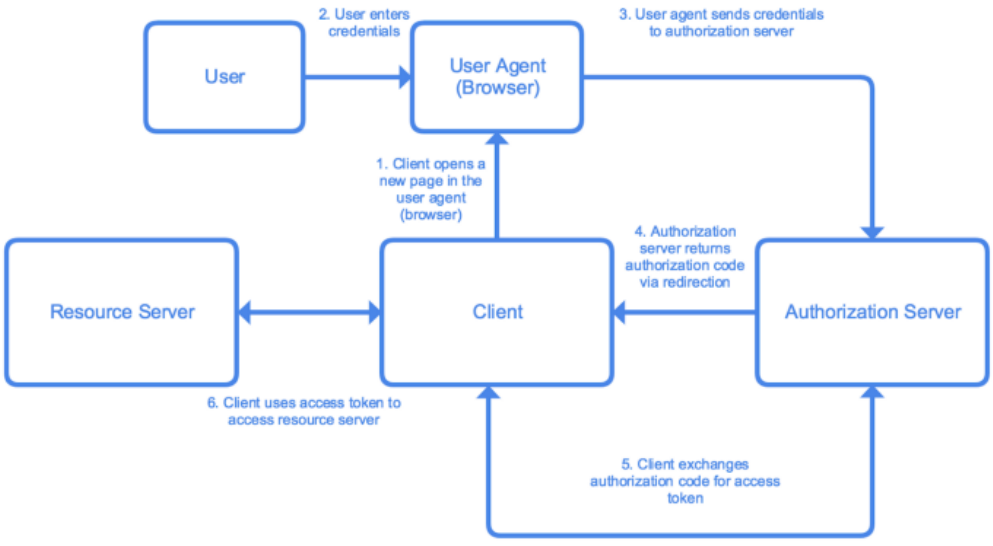
- Creating an instance
- Power on/off an instance
- Possibility to rename the instance
- Possibility to delete the instance

These operations are the same that are then provided to the MyCSM application using the API. Each profile registered in the CSP can have 3 types of associated plan, a "Bronze" plan where the user does not pay anything but can create 3 instances, one "Basic" and one "Premium" where for a monthly expense he can create respectively 10 or infinite instances.

CSP as Authorization Server

Authorization Code flow has the following requirements:

- The client must be able to interact with a user agent (browser) in the environment.



For this requirement each of our CSPs works as an authorization server. This means that the client prepares a link to the **authorization server** and opens the link for the user in an **user agent** (browser). The link includes information that allows the authorization server to identify and respond to the client.

This happens when in MyCSM you try to add a new CSP to the application. The "*/add_csp*" route is called and the client is redirected to the authorization page of the CSP.

On CSP side, the *@route* "*/oauth/authorize*" is called and once credentials are entered and consent is given, the authorization server redirects to the callback address specified during the client registration (that in our project is : */callback/<csp>/<scope>*).

During the callback the client requests the token using the received authorization code, saves it and returns to the CSP page.

From this moment on, a MyCSM client will be able to access the resources via API using the token that has been provided.

MyCSM Application

The heart of the project lies in the MyCSM application. As we previously mentioned, a user can sign up for the application and choose between two plans: a premium and a free plan.

The premium plan provides, for a monthly payment, full access to the operations provided by the CSP, such as the creation / update / deletion of an instance and the possibility to turn on / off the machine.

For a CSP client to be added to the list of available clients within the application, it must be registered.

To register a new CSP client, you need to go to the '*/administration*' route of MyCSM. Here we will find all the fields necessary for OAuth2 to work. It will be necessary to indicate:

- The free / premium plan
- The CSP
- The scope (here we have limited ourselves to distinguishing between 2 scopes, "read" for reading only and therefore access to the API limited to view the consumption values of HDD and possibly RAM and a "write" scope where all the APIs are granted)
- A client ID
- A secret client
- The endpoint for authorization, token, API, and token revoke

Some of this information can be retrieved directly on the administration page of the CSP of interest at the '*/oauth*' route after logging in on the authorization server.

Once a new CSP is linked to your account via the authorization process described in the previous section, it will appear in the menu on the left.
When we go to the page related to that CSP we will find ourselves in front of a dashboard similar to the previous one, but where the returned values are taken from the API through Ajax calls.

Protocols and technologies involved

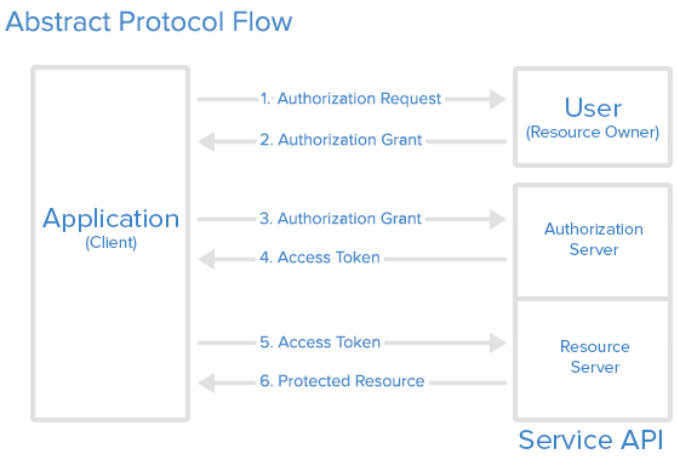
OAuth2

OAuth 2 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. OAuth 2 provides authorization flows for web and desktop applications, and mobile devices.

This protocol defines five roles:

- **Resource Owner:** this is the user who authorize the application to access its private resources, which are limited to the scope of the authorization granted
- **Resource Server:** this is the server which hosts the protected user account
- **Authorization Server:** this is the server that verifies the identity of the user and then issues access tokens to the application
- **Client (or Application):** this is the application that wants to access the private resources of the user (previously authorized by the user and verified by the Authorization Server)

Resource Server and **Authorization Server** are often described together, as a unique server/infrastructure.

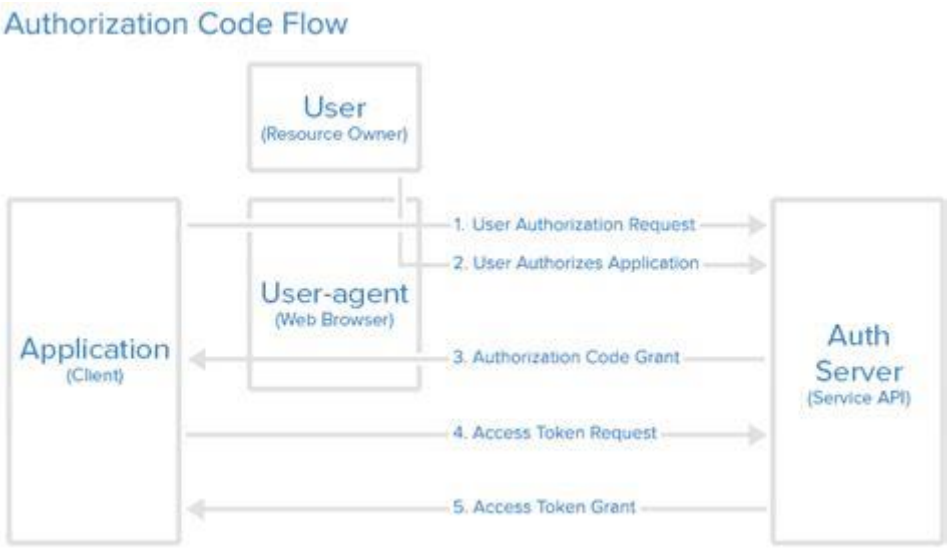


According to the Protocol, the application (which mantains a secret, given from the authorization server) follows an authorization flow, interaction with the other entities (roles), to obtain an authorization grant and then request a token. There are four types of authorization grant:

- **Authorization Code Grant:** this is the most commonly used authorization grant type and it is optimized for server side applications. The application requests an authorization code that is sent after that the user authorize the request through the authorization server. The authorization code is used to request the token that will be used to access the user's resources.
- **Implicit Grant:** the token is sent directly after the authorization request, with no need of other types of credentials
- **Resource Owner Password Credentials Grant:** the application sends the user username and password to obtain the token
- **Client Credentials Grant:** the application sends its client_id and client_secret to obtain the token

Authorization Code Grant Flow

Our csp's Authorization server is designed (and ready) to use every authorization grant type, but, for MyCSM we focused only on Authorization Code Grant type. The flow of the protocol with authorization code grant is the following:



In **1** and **2** the user is redirected to the authorization code link, in which he will authenticate his identity (unless he's already logged in) and then he will be prompted for an authorization form, to decide if he wants to authorize or deny the access to his resources by the application.

In **3**, if the user authorizes the access, the authorization server sends a redirect to the redirect URI, specified during the client registration, in an **authorization code** as a parameter.

In **4** and **5** the application uses the authorization code previously obtained to request (and obtain) the access token, that will be used to access the private resources of the user (via API).

Programming Languages and Server Technologies

Python3

Both MyCSM and the Cloud Service Provider server were built using Python3 and specifically, using the Flask Framework for web applications. The Authorization Server was implemented using the Python3 Authlib library, which provides OAuth1/2 and OpenID primitives implementation and several frameworks (such as Requests, Flask and Django) integrations.

Flask

Flask is a Python Framework designed for building web applications developed by Armin Ronacher. It is built upon Werkzeug, a WSGI (Web Server Gateway Interface) and Jinja (a powerful template engine). Developing web applications in Flask is simple. Routes can be written directly into the main app or via Blueprints (this is useful to separate different routes). A simple route is written using Python decorators and functions declaration:

```
@route('/uri')
def my_route():
      ...
      return render_template('page.html')
```

Every route returns a response, that can be a Jinja template, an HTML page, a redirect or other type of responses.

Flask Talisman

Flask talisman was born as a Flask extension that tries to set HTTP headers in such a way as to protect against some common security problems in web applications.

The default configuration includes:

- Forces all connects to https, unless running with debug enabled.
- Enables HTTP Strict Transport Security.
- Sets Flask's session cookie to secure, so it will never be set if your application is somehow accessed via a non-secure connection.
- Sets Flask's session cookie to *httponly*, preventing JavaScript from being able to access its content. CSRF via Ajax uses a separate cookie and should be unaffected.
- Sets X-Frame-Options to SAMEORIGIN to avoid clickjacking.
- Sets X-XSS-Protection to enable a cross site scripting filter for IE and Safari (note Chrome has removed this and Firefox never supported it).
- Sets X-Content-Type-Options to prevent content type sniffing.
- Sets a strict Content Security Policy of default-src: 'self'. This is intended to almost completely prevent Cross Site Scripting (XSS) attacks. This is probably the only setting that you should reasonably change. See the Content Security Policy section.
- Sets a strict Referrer-Policy of strict-origin-when-cross-origin that governs which referrer information should be included with requests made.

Authlib

Authlib is a Python library designed to build OAuth (1 and 2) and OpenID Connect servers. It is designed from low level specifications implementations to high level frameworks integrations, to meet the needs of everyone. It supports several RFC specifications for OAuth protocols and provides integrations for the most used web framework, such as Django and Flask, but also Requests and httpx. In our project, we used the Flask integration of OAuth2 Authlib client and server. This integration provides basic objects to build clients (from which perform API calls with oauth tokens and the standard OAuth authorization flow) and servers (which implements an easy way to define resource protectors for accessing to the user resources only with specific scopes).

Bootstrap, jQuery, JS, AJAX

As for the FrontEnd part of the application, the Bootstrap framework has been used, this includes design templates based on HTML and CSS for typography, forms, buttons, tables, navigation, modals, image carousels and much more, as well as plugins -in optional JavaScript.
In order to make the experience with the interface of the web pages more interactive, we have chosen to make use of AJAX calls in such a way as to be able to exchange data between client and server without reloading the page: the exchange takes place in the background through an asynchronous call of the data usually using the XMLHttpRequest object. And this data exchange is achieved, as you can guess from the acronym, through functions written with the JavaScript language.
To make the implementation of AJAX easier, it was chosen to use jQuery, a JS library that aims to allow the simplified creation of web applications and versatile dynamic content.

Security properties and solutions

HTTPS

Hypertext transfer protocol secure (HTTPS) is the secure version of HTTP, which is the primary protocol used to send data between a web browser and a website. HTTPS is encrypted in order to increase security of data transfer. This is particularly important when users transmit sensitive data, such as by logging into a bank account, email service, or health insurance provider.

As explained in the RFC 8446, the primary goal of TLS is to provide a secure channel between two communicating peers; the only requirement from the underlying transport is a reliable, in-order data stream. Specifically, the secure channel should provide the following properties:

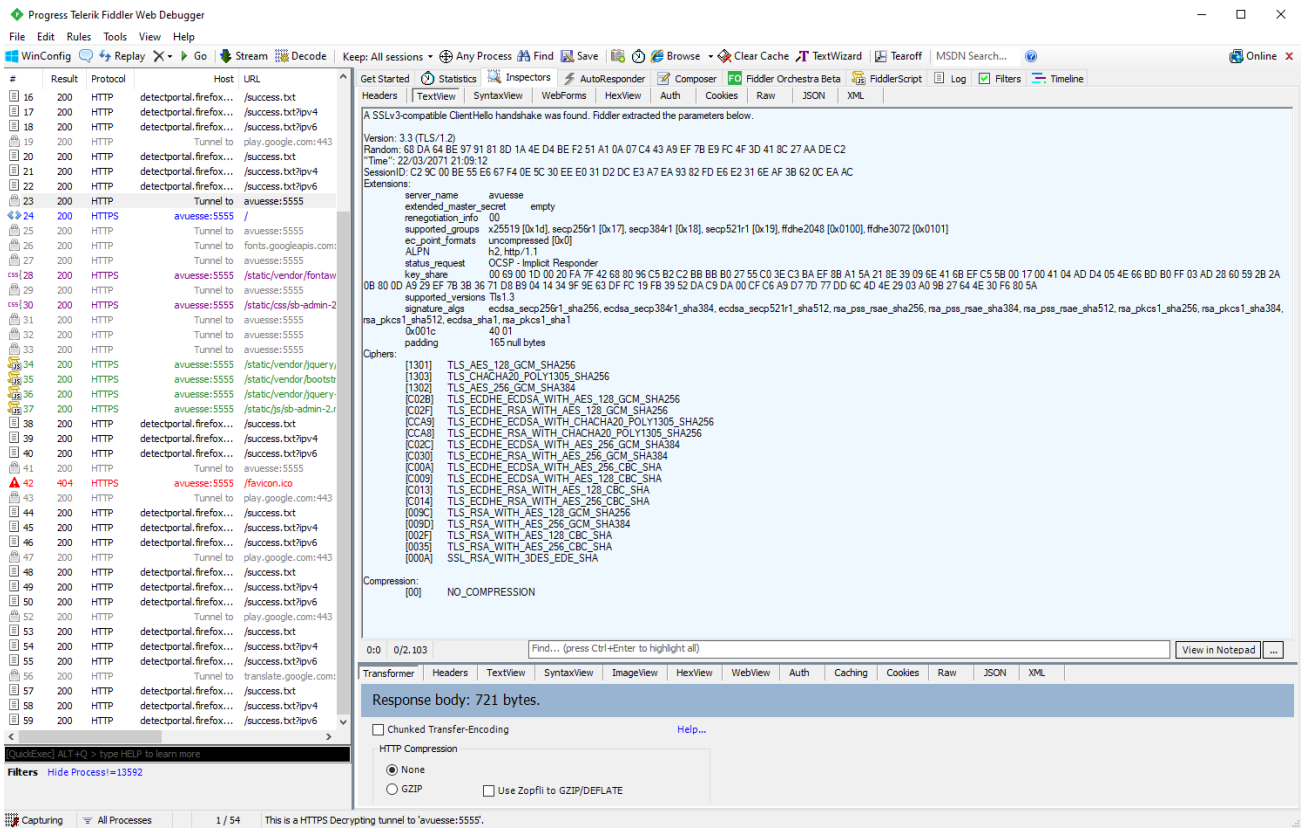
- Authentication: The server side of the channel is always authenticated; the client side is optionally authenticated.
- Integrity: Data sent over the channel after establishment cannot be modified by attackers without detection.
- Confidentiality : Data sent over the channel after establishment is only visible to the endpoints. TLS does not hide the length of the data it transmits, though endpoints are able to pad TLS records in order to obscure lengths and improve protection against traffic analysis techniques.

To verify the properties of confidentiality, integrity, authenticity and availability guaranteed by the use of HTTPS, Fiddler was used Fiddler is a proxy server that acts as an "intermediary" between the remote system with which, for example, the browser is talking (a web server) and the user's client machine. Thanks to its architecture, the program is able to capture all HTTP and HTTPS traffic showing it in an easily readable form to the user who can then analyze it calmly. This interesting utility can also be exploited to dynamically alter the data traffic in transit to and from a certain server: in this way it is possible to verify the behavior of the two systems involved in the communication and carry out in-depth tests.

HTTPS is HTTP inside a SSL/TLS tunnel, it works like a postcard (HTTP) in an envelope (SSL/TLS).

Note: SSL/TLS can be used with a certificate not signed by a trusted authority (but the client will show a warning about that).

TLS ClientHello Request




```
CONNECT avuesse:5555 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0) Gecko/20100101
Firefox/80.0
Connection: keep-alive
Connection: keep-alive
Host: avuesse:5555

A SSLv3-compatible ClientHello handshake was found. Fiddler extracted the
parameters below.

Version: 3.3 (TLS/1.2)
Random: 68 DA 64 BE 97 91 81 8D 1A 4E D4 BE F2 51 A1 0A 07 C4 43 A9 EF 7B E9 FC
4F 3D 41 8C 27 AA DE C2
"Time": 22/03/2071 21:09:12
SessionID: C2 9C 00 BE 55 E6 67 F4 0E 5C 30 EE E0 31 D2 DC E3 A7 EA 93 82 FD E6
E2 31 6E AF 3B 62 0C EA AC
Extensions:
    server_name     avuesse
    extended_master_secret     empty
    renegotiation_info     00
    supported_groups     x25519 [0x1d], secp256r1 [0x17], secp384r1 [0x18],
secp521r1 [0x19],           ffdhe2048 [0x0100], ffdhe3072 [0x0101]
    ec_point_formats     uncompressed [0x0]
    ALPN     h2, http/1.1
    status_request     OCSP - Implicit Responder
    key_share     00 69 00 1D 00 20 FA 7F 42 68 80 96 C5 B2 C2 BB BB B0 27 55 C0
3E C3 BA EF 8B A1 5A 21 8E 39 09 6E 41 6B EF C5 5B 00 17 00 41 04 AD D4 05 4E 66
BD B0 FF 03 AD 28 60 59 2B 2A 0B 80 0D A9 29 EF 7B 3B 36 71 D8 B9 04 14 34 9F 9E
63 DF FC 19 FB 39 52 DA C9 DA 00 CF C6 A9 D7 7D 77 DD 6C 4D 4E 29 03 A0 9B 27 64
```

```
4E 30 F6 80 5A
    supported_versions    Tls1.3
    signature_algs    ecdsa_secp256r1_sha256, ecdsa_secp384r1_sha384,
ecdsa_secp521r1_sha512, rsa_pss_rsae_sha256, rsa_pss_rsae_sha384,
rsa_pss_rsae_sha512, rsa_pkcs1_sha256, rsa_pkcs1_sha384, rsa_pkcs1_sha512,
ecdsa_sha1, rsa_pkcs1_sha1
    0x001c         40 01
    padding        165 null bytes

Ciphers:

    [1301]    TLS_AES_128_GCM_SHA256
    [1303]    TLS_CHACHA20_POLY1305_SHA256
    [1302]    TLS_AES_256_GCM_SHA384
    [C02B]    TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
    [C02F]    TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
    [CCA9]    TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
    [CCA8]    TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
    [C02C]    TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
    [C030]    TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
    [C00A]    TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
    [C009]    TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
    [C013]    TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
    [C014]    TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
    [009C]    TLS_RSA_WITH_AES_128_GCM_SHA256
    [009D]    TLS_RSA_WITH_AES_256_GCM_SHA384
    [002F]    TLS_RSA_WITH_AES_128_CBC_SHA
    [0035]    TLS_RSA_WITH_AES_256_CBC_SHA
    [000A]    SSL_RSA_WITH_3DES_EDE_SHA

Compression:

    [00]    NO_COMPRESSION
```

TLS ClientHello Response

```
HTTP/1.1 200 Connection Established
FiddlerGateway: Direct
StartTime: 14:59:20.074
Connection: close

Encrypted HTTPS traffic flows through this CONNECT tunnel. HTTPS Decryption is
enabled in Fiddler, so decrypted sessions running in this tunnel will be shown
in the Web Sessions list.



Secure Protocol: Tls12
Cipher: Aes256 256bits
Hash Algorithm: Sha384 ?bits
Key Exchange: ECDHE_RSA (0xae06) 255bits

== Server Certificate ==========

[Subject]
  E=info@avuesse.it, CN=avuesse, OU=Avuesse, O=Avuesse, L=Milan, S=Italy, C=IT
[Issuer]
  E=info@avuesse.it, CN=avuesse, OU=Avuesse, O=Avuesse, L=Milan, S=Italy, C=IT

[Serial Number]
  3DC0D131DAA554945ED9534BB736B26A363AA2EC

[Not Before]
  06/09/2020 19:23:21

[Not After]
  06/09/2021 19:23:21

[Thumbprint]
  4F125BCB309B6449EDB0D743546FD116620B3D9F
```

Conclusion

The goal of the project was to build an OAuth2.0 server to provide access to the CSP resources to the MyCSM application, guaranteeing the security property of Authorization, and to keep the exchange of messages between the MyCSM web application and its server.

This was possible thanks to the use of the HTTPS protocol, which guaranteed the security properties of confidentiality, integrity and authentication.

The aim of the realization of the software was to lay the foundation for development and code the core of the system that can be extended by adding new features to the structure created.