SecretStroll

Introduction

In this project, you will develop a location-based application which enables users to search for nearby points of interest (POI). We provide you a skeleton to help with the development.

We use Docker to enforce isolation between the client and server on the virtual machine. Docker is a software that uses the capabilities of the Linux kernel to run processes in a sandboxed environment. As such both the server and the client will "think" they are running in two independent systems.

Both the client and the server provide a command-line interface to run and interact with the server. The underlying location-based service is already implemented in the skeleton, and your task is to add the authentication with attribute-based credentials.

We strongly recommend to use the `petrelic` cryptographic pairing library to implement PS credentials. You can find the project repository in https://github.com/spring-epfl/petrelic/ and you can visit https://petrelic.readthedocs.io for documentation. This library is bundled in the provided Docker container and virtual machines.

To ease your work, the skeleton has already implemented and integrated capabilities to save, manage, and load keys and credentials as byte arrays. You need to implement (de)serialization methods to match the API. We provide `serialization.py` as a `petrelic` extension of `jsonpickle`, a serialization library, to help you with the serialization of cryptographic objects.

Part of the developing a system is testing the system, and you should check both success and failure paths when your working with cryptographic primitives. In this project you **must** use the *pytest* framework to test your system.

A VM is provided to develop this project, this is to facilitate the setup and configuration of the SecretStroll and reduce potential networking problems while running the application in different environments. As we mentioned, the client and server are going to run in two different Docker containers to simulate running them on two different machines.

Files in the skeleton

The skeleton contains the following files:

- `your_code.py` source code that you have to complete.
- `client.py` client CLI calling classes and methods defined in `your_code.py`.
- `server.py` server CLI calling classes and methods defined in `your_code.py`.
- `serialization.py` Extends the library `jsonpickle` to serialize python objects.

- `requirements.txt` Required Python libraries.
- `readme.md` This file…
- `docker/` directory containing Docker configurations for running the client and the server.
- `docker-compose.yaml` *docker compose* configuration describing how to run the Docker containers.

Setting up the development environment

All necessary python components are already installed on the VM and dockers. However, you can set up a virtual environment and install the requirements with `pip`. You can use the following command to install requirements:

```
pip3 install -r requirements.txt
```

You can use git repositories to sync your work with your teammates. However, keep in mind that you are not allowed to use public repositories, so make sure that your repository is **private**.

Feel free to have a look at `client.py` and `server.py` to see how the classes and methods are used.

How to develop

The deployment is handled by Docker and our skeleton. In this section, we introduce the Docker infrastructure and how to use it. Then, we provide a step by step guide of running the client and server. You just need to to develop an attribute-based credential and update the `your_code.py` file to use it. The skeleton takes care of syncing your code between the VM and containers.

Running the Docker infrastructure

Before launching the infrastructure, ensure the `tor` directory in the project skeleton has the correct permissions

```
student@cs523:~/skeleton$ chmod 777 tor
student@cs523:~/skeleton$ ls -ld tor
drwxrwxrwx 2 student student    4096 mar 24 15:31 tor
```

The server and the client run in a Docker infrastructure composed of 2 containers, and a virtual network.

Before setting up the Docker infrastructure for the first time, it is necessary to build images which will be used to run both client and server containers. To do so, run the following command in the skeleton directory, which contains the file `docker-compose.yml`:

```
docker-compose build
```

Note: We have already build the duckers inside the VM.

To set up the Docker infrastructure, run the following command in the directory containing the file `docker-compose.yml`:

```
docker-compose up -d
```

When you stop working with the infrastructure, remember to shut it down by running the following command in the directory containing the file `docker-compose.yml`:

```
docker-compose down
```

If you forget to shut down the Docker infrastructure i.e. before shutting down your computer, you might end up with stopped Docker containers preventing the creation of the new ones when you to re-launch the infrastructure the next time. This can be fixed by removing the network bridge with `docker-compose down` and destroying the stopped Docker containers with `docker container prune -f`.

Accessing the data

The code of the skeleton is shared between your VM and Docker containers, so modifications you make in your VM will also appear in containers. Feel free to read the file `docker-compose.yml` to see how it is done.

If you need to transfer some data between your VM and your host machine, VirtualBox offers the possibility to have shared directories between the VM and your host. For this feature to work correctly, please, install the *Guest Additions* from VirtualBox on the VM by referring to their documentation.

Running the server

It is easier to run the commands in a Docker container by opening a shell, then running the commands inside this shell.

To execute a shell in the container in which the server is to be launched, run the following command:

```
docker exec -it cs523-server /bin/bash
```

In this container, the root directory of the project is mounted on `/server`.

```
cd /server
```

Here is the usage of the server:

The server has two subcommands `gen-ca` and `run`. `gen-ca` is used to generate the public and secret keys, `run` is used to run the server. The server and its subcommands have an embedded help, which can be read with the `-h` command. Here is a short guide how to use it.

```
python3 server.py gen-ca -a 'attribute_string' -s key.sec -p key.pub
```

```
usage: server.py gen-ca [-h] -a ATTRIBUTES -p PUB -s SEC

optional arguments:
  -h, --help            show this help message and exit
  -a ATTRIBUTES, --attributes ATTRIBUTES
                        Valid attributes recognised by the server.
  -p PUB, --pub PUB     Name of the file in which to write the public key.
  -s SEC, --sec SEC     Name of the file in which to write the secret key.
```

```
python3 server.py run -s key.sec -p key.pub
```

```
usage: server.py run [-h] -p PUB -s SEC

optional arguments:
  -h, --help         show this help message and exit
  -p PUB, --pub PUB  Name of the file containing the public key.
  -s SEC, --sec SEC  Name of the file containing the secret key.
```

In the Part 3 of the project, the server is expected to be accessible as a Tor hidden service. The server's Docker container configures Tor to create a hidden service and redirects the traffic to the python server. The server serves local and hidden service requests simulatneasly, and you do not need an argument to determine this behaviour.

The server also contains a database, `fingerprint.db`. This is used in Part 3. The database has a POI table, that contains records for each POI. The server returns the list of POIs associated with a cell ID that is queried by the client, and information about each POI in the list. You should not modify the database.

Running the client

To execute a shell in the container in which the client is to be launched, run the following command:

```
docker exec -it cs523-client /bin/bash
```

In this container, the root directory of the project is mounted on `/client`.

```
cd /client
```

Here is the usage of the client:

The client has four subcommands `get-pk`, `register`, `loc`, and `grid`. As for the server, the client and its subcommands have an embedded help, which can be read with the `-h` command.

Use `get-pk` to retrieve the public key from the server.

```
python3 client.py get-pk -o key-client.pub
```

```
usage: client.py get-pk [-h] [-o OUT] [-t]
```

```
optional arguments:
  -h, --help         show this help message and exit
  -o OUT, --out OUT  Name of the file in which to write the public key.
  -t, --tor          Use Tor to connect to the server.
```

Use `register` to register an account on the server.

```
python3 client.py register -a 'attributes' -p key-client.pub -u your_name -o attr.cred

usage: client.py register [-h] [-p PUB] [-o OUT] -a ATTRIBUTES [-t] -u USER

optional arguments:
  -h, --help            show this help message and exit
  -p PUB, --pub PUB     Name of the file from which to read the public key.
  -o OUT, --out OUT     Name of the file in which to write the attribute-based
                        credential.
  -a ATTRIBUTES, --attributes ATTRIBUTES
                        String representing the attributes.
  -t, --tor             Use Tor to connect to the server.
  -u USER, --user USER  User name.
```

Use `loc` and `grid` to retrieve informations about points of interests using lat/long location (Part 1) and cell identifier (Part 3).

```
python3 client.py loc -p key-client.pub -c attr.cred -r 'revealed_attrs' 46.52345 6.57890

usage: client.py loc [-h] -p PUB -c CRED -r REVEAL [-t] lat lon

positional arguments:
  lat                   Latitude.
  lon                   Longitude.

optional arguments:
  -h, --help            show this help message and exit
  -p PUB, --pub PUB     Name of the file from which to read the public key.
  -c CRED, --cred CRED  Name of the file from which to read the attribute-based
                        credential.
  -r REVEAL, --reveal REVEAL
                        Attributes to reveal. (format: attr1,attr2,attr3).
  -t, --tor             Use Tor to connect to the server.
```

Warning: The database only contains points of interest with latitude in range [46.5, 46.57] and longitudes in range [6.55, 6.65] (Lausanne area). You can make queries outside these values, but you will not find anything interesting.

```
python3 client.py grid -p key-client.pub -c attr.cred -r 'revealed_attrs' 42

usage: client.py grid [-h] -p PUB -c CRED -r REVEAL [-t] cell_id
```

```
positional arguments:
  cell_id               Cell identifier.

optional arguments:
  -h, --help            show this help message and exit
  -p PUB, --pub PUB     Name of the file from which to read the public key.
  -c CRED, --cred CRED  Name of the file from which to read the attribute-based credential
                        credential.
  -r REVEAL, --reveal REVEAL
                        Attributes to reveal.
  -t, --tor             Use Tor to connect to the server.
```

A sample run of Part 1

We show a typical run of the system for Part 1:

Initialization:

```
Open a shell
$ cd skeleton
$ docker-compose build
$ docker-compose up -d
```

Server side:

```
Open a shell
$ cd skeleton
$ docker exec -it cs523-server /bin/bash
(server) $ cd /server
(server) $ python3 server.py gen-ca -a 'attributes' -s key.sec -p key.pub
(server) $ python3 server.py run -s key.sec -p key.pub
```

Client side:

```
Open a shell
$ cd skeleton
$ docker exec -it cs523-client /bin/bash
(client) $ cd /client
(client) $ python3 client.py get-pk -o key-client.pub
(client) $ python3 client.py register -a 'attributes' -p key-client.pub -u your_name -o attr
(client) $ python3 client.py loc -p key-client.pub -c attr.cred -r 'revealed attrs' 46.52345
```

A sample run of Part 3

We show a typical run of the system for Part 1:

Initialization:

```
Open a shell
```

```
$ cd skeleton
$ docker-compose build
$ docker-compose up -d
```

Server side:

You should have already generated the keys in Part 1, so you do not need to repeat that step.

```
Open a shell
$ cd skeleton
$ docker exec -it cs523-server /bin/bash
(server) $ cd /server
(server) $ python3 server.py run -s key.sec -p key.pub
```

Client side:

You should have already performed the registration in Part 1, so you do not need to the repeat the step. Use the grid parameter to query for a particular cell. Set the reveal argument (-r) to an empty value.

```
Open a shell
$ cd skeleton
$ docker exec -it cs523-client /bin/bash
(client) $ cd /client
(client) $ python3 client.py grid -p key-client.pub -c attr.cred -r '[]' 42
```