Distributed Systems
COMP90015 2017 SM1
Project 2 - Security and Subscribing

Project 2

The project involves building on Project 1. If you have not satisfactorily completed Project 1 then you'll need to work with your tutor and lecturer to catch up.

You are required to:

- Implement a secure socket port that works along side the existing port. This will include making secure connections during exchange and query operations between servers.
- Implement a SUBSCRIBE/UNSUBSCRIBE command, that will receive published/shared resource descriptions as they are published/shared.

For the security aspect you will make use of certificates and secure sockets as discussed in the lectures.

Secure Sockets

- ~~Every server will start up a secure socket on a designated "secure" port (3781 by default) as well as the existing "unsecured" port. The secure socket will only accept secure connections.~~
- Successful secure connections will be treated in the same way as unsecured connections for the most part. However:
    - Queries that originate from a client on a secure connection will only be relayed to other servers on their secure ports.
- The client and server will both need a certificate signed by a certificate authority. In this project, I will be the certificate authority and your group will need to email me (or potentially the tutors who will be able to handle the request as well) your certificate signing requests. For development purposes you can test with your own certificate authority but eventually before you submit your code you will need to have a signed certificate from me for both your server and client. I will give more details about this on LMS/in class.
- ~~The client will have a `-secure` flag that, when given by the user, means that the client should attempt to make a secure connection rather than a unsecured connection. The port given by the user with -port (if any) will be assumed to be secure.~~
- ~~The server will have a `-sport` flag that takes the secure port number as its argument. The default secure port is 3781.~~

Server rules on secure connections

A server that receives a secure connection from a client should assume that all information and any further processing, e.g. making connections to other server, should be secure as well:

- Any relay connection should be secure if the original client connection is secure.
- ~~The server details in an EXCHANGE message should be considered to be secure ports if the EXCHANGE message was received on a secure connection; and unsecured otherwise.~~

- If a server exchanges known servers with other servers, it should never exchange unsecured host:port pairs over a secure connection. The two types of connections need to be maintained separately in this sense and there needs to be an exchange that happens periodically for the unsecured connections and one that happens periodically for the secure connections. The `-exchangeinterval` parameter and ~~`-connectionintervallimit`~~ parameter can be used for both secure and unsecured connection management.

~~This effectively means that your server will maintain, for every known host:port pair, whether this is a secure connection or not. This might be implemented as two sets of servers: a secure set and an unsecured set.~~

~~Similarly, if a client connects to an unsecured port then all further relaying should be on unsecured ports only. It should not be the case that a query relay could be partially secure and partially unsecured.~~

SUBSCRIPTION command

The SUBSCRIBE command works almost identically to the QUERY command:

```
{
    "command": "SUBSCRIBE",
    "relay": true,
    "id" : "X",
    "resourceTemplate": {
        "name": "",
        "tags": [],
        "description": "",
        "uri": "",
        "channel": "",
        "owner": "",
        "ezserver": null
    }
}
```

The big difference is that the connection is a **peristent connection** and the response is **asynchronous**. As well, the SUBSCRIBE command has a client defined `id` field which should be a string and can be any thing that the client would like to use to refer to this particular subscription request in the future.

The user will give the command line option `-subscribe` to indicate a subscribe command.

SUBSCRIPTION client behavior

The client makes the connection, sends the command and then waits for an unbounded number of responses (in fact this is still quite similar to the query). Responses that match the template come whenever the server becomes aware of them. The connection is not closed until the client sends an UNSUBSCRIBE command and receives a final response from the server.

As well, any number of SUBSCRIBE commands can be sent to the server on the same persistent connection, asynchronously. Each one of them should be stored by the server and a match to any of them should be sent back to the client/server from which the connection was made.

For the client in this project, for simplicity of implementation, the command line will allow only a single SUBSCRIBE command. When the user presses ENTER, i.e. a line is read from standard input, then the client will UNSUBSCRIBE and can then terminate.

SUBSCRIPTION responses from server

Each time the client sends a SUBSCRIBE command, that has no errors, the server will respond with

```
{ "response" : "success",
  "id" : "X"
}
```

where X is the same `id` that was given in the subscribe command. The server is of course also sending back hits, asynchronously, to any subscribed resource templates in the form of:

```
{ RESOURCE }
```

If the resource template in the subscribe command contained incorrect information that could not be recovered from:

```
{ "response" : "error",
  "errorMessage" : "invalid resourceTemplate"
}
```

If the resource template was not given or not of the correct type:

```
{ "response" : "error",
  "errorMessage" : "missing resourceTemplate"
}
```

Same error messages can be sent for the `id` field, if it is missing or not of the correct type.

UNSUBSCRIBE command

```
{
    "command": "UNSUBSCRIBE",
    "id" : "X"
}
```

This will remove the subscription X from the server. Other subscriptions may still remain.

If all subscriptions are stopped and the connection is closed, the server responds with:

```
{ "resultSize" : X }
```

where X is the total number of hits during the entire connection, and the connection is closed by the server.

SUBSCRIBE relay

~~If the `relay` field is true then the server should make subscription requests to other known servers, similar to query, but in this case the subscription is a persistent asynchronous connection from one server to another.~~

~~Subscription commands sent by the client should be relayed on to other servers, and hits coming back from those servers should be forwarded back to the client.~~

As new servers become known (though exchange), existing subscriptions from clients where `relay` is true should be extended to include those servers as well. This can lead to a large number of connections being maintained if it is implemented as a connection per client subscription. Your exact implementation is up to you, so long as the client is able to receive published/shared resources from any server (not just those servers that you implemented yourself) in an asynchronous way.

If the client originally connected via a secure connection then the relay connections for the subscription should be secure as well, i.e. should only be between servers that support secure connection. Vice versa for unsecured subscribe commands: they should only be relayed over unsecured connections.

Technical aspects

- Your programs should start from the command line just as they did in the first project, with the difference being only the new options available for server and client.
- Any additional files that you need, e.g. certificates, should be embedded into the JAR file, rather than being read from the computer's file system.

Report

There are two aspects to discuss in your report, using about 500 to 750 words for each aspect:

- You were asked to implement security without being given a security policy, i.e. without knowing what kinds of things we are trying to be secure against. To what extent do the security mechanisms implemented in the project address known security issues? What kinds of problems are not addressed? On another front, the owner and channel information is sensitive and it is not relayed between servers, nor is owner information shown in the results of user queries. Having an owner and channel provides a certain functionality for users but the current implementation limits the applicability. How could appropriate use of security mechanisms, e.g. like public key cryptography, be used to improve the system? Put forward any ideas you have in a clear way and discuss their merit.
- Servers relay a subscription to other servers by making a similar persistent asynchronous connection. This can lead to a large number of connections at any point in time, depending in part on the implementation. Discuss your implementation of this aspect. Discuss an alternative approach that does not require asynchronous persistent connections between the servers, allowing that from a correctness point of view that the alternative may be correct eventually (i.e. through the eventual consistency property). Critically compare the approaches.

Submission

- You should submit your report plus your modified system similarly to Project 1; instructions will be given on LMS.
- You will have a similar chance in Project 2 to assess your team members using the LMS test submission.
- The due date is Saturday Week 12, 27th May, 11:59pm.