DD2525 – Lab Assignment 1
Information Flow Control

Deadline: April 17, 2024

1 Introduction

This assignment considers a scenario of a simple online dating service that has the distinguishing feature of allowing its users to specify visibility of their profiles at a fine-grained level. This is achieved by allowing users of the service to provide a piece of code that controls their visibility. In the rest of the assignment, we refer to such code pieces as *discovery agent*. Discovery agents work as follows. Suppose we have a user Bob who wants to control how discoverable his profile is. Bob provides his implementation of a discovery agent that takes as input two profiles – his profile and the profile of another user, say Alice. The discovery agent is run on the server, possibly with side effects. It returns a pair of a boolean – specifying whether Bob wants to be discovered by Alice – and a possibly empty profile with the information about Bob that Alice can see.

Goal of the assignment The goal of this assignment is to implement the dating server and its clients in Troupe, figuring out the necessary security labels and placements for declassifications and authority declarations.

2 Getting Started

The following subsections describe how to install and get started with Troupe. Download the Troupe user guide from `https://canvas.kth.se/courses/46200/files/folder/Labs?preview=7402724` and study it carefully. You can find the examples from the user guide at `$TROUPE/examples/fromuserguide` and `$TROUPE/examples/network` in the Troupe Docker image. This video tutorial `https://canvas.kth.se/courses/46200/external_tools/2427` can be helpful.

2.1 Downloading and Running Troupe in a Docker container

This subsection explains how to download run Troupe inside a Docker container. Please follow the instructions below.

1. Download the Docker image for Troupe using a terminal/shell:

   ```
   $ docker pull ghcr.io/troupelang/troupe:latest
   ```

2. Download the assignment code: `https://canvas.kth.se/courses/46200/assignments/280264`

3. Unzip the code.

4. Execute the following command to run a Troupe container:

   ```
   $ docker run -it -entrypoint=/bin/bash \
   -v "$(pwd)":/code ghcr.io/troupelang/troupe
   ```

   This command opens a bash session inside a Troupe container. When we mount the volume, `-v "$(pwd)"...`, this option tells Docker to mount the current directory into the container's `/code` folder. Thus, we can interact with the code folder inside the bash session using your favourite text editor.

5. Inside the opened session, go to code folder:

   ```
   $ cd ../code
   ```

6. You can now follow the instructions from Section 2.3.1.

2.2 With VSCode

There is an VSCode extension that automates step 4 above for you. You can find more information in `https://code.visualstudio.com/docs/remote/containers#_getting-started` There is also a Troupe boilerplate for VSCode running inside a Docker container, `https://github.com/TroupeLang/example-project`.

2.3 Starting code for the assignment

Download the starting files from Canvas `https://canvas.kth.se/courses/46200/assignments/280264`.

2.3.1 Create a network identifier for your server

Create a network identifier (together with the public/private key pair) for your node:

```
$ cd code/ids/
$ node $TROUPE/rt/built/p2p/mkid.js - -outfile=id-server.json
```

The "id" section of the JSON file is your node identifier. The other parts correspond to the private and public keys – technically, the identifier is the hash of the public key.

2.3.2 Testing your network identifier (optional)

At this step, you can pass the file `id-server.json` when passing it as an argument to Troupe runtime. The provided Makefile already does that for the starting file and you should be able to run the starting file for the server by typing `make dating-server` in the directory `code/dating`.

2.4 Guidelines through the code

The directory `code/dating-server` includes the starting files for the dating server implementation, including the Makefile. The initial code for the server is very minimal but functional: it includes the registration of the server, and a trivial server loop that listens on a message matching “`{NEWPROFILE}`“ string. It subsequently contacts one of our dispatcher servers that spawns a few clients that contact your machine. Upon running the provided code, you should see the output as below.

```
 »> Main thread finished with value: ()@{}%{}
New profile received
New profile received
New profile received
New profile received
...
```

3 Trust relationships

The trust relationships in this assignment are such that every client that operates at level ℓ must trust the server up to level ℓ. For example, a client of Alice, must trust the server at the level up to `‘{alice}‘`. The server trust relationship must also reflect this. To make your server works with our clients, we have provided a sample trustmap file `trustmaps/GoT.json` that includes the levels and the nodes of our machines. Trustmap files are simple JSON files. When developing your own clients you will need to create the corresponding entries in the trustmap file used by your server. A very simple trustfile for client of Alice may look like this.

```
[{"id":"Qm-your-server-identfier","level":"alice"}]
```

This means that the server is trusted to receive information up to level `{alice}`. To generate the identifiers for the clients, use the same process as for creating a network identifier (cf. Section 2.3.1).

4 Protocol and behavior specification

There are only two kinds of messages that we exchange. Messages from the clients to the servers match the pattern `("NEWPROFILE", _)`. Messages from the server to the clients match the pattern `("NEWMATCH",_)`. We describe the specification of the messages below.

4.1 ”NEWPROFILE“ messages

When an incoming message has the form `(”NEWPROFILE”, data)`, we expect that the data is a tuple of the form `(profile, agent, pid)`, where `profile` is further a tuple of the form `(lev, name, year, gender, interests)`. The values in the profile tuple are

- Value `lev` is the security level that you expect the rest of the fields of the profile are labeled with. The label of the `lev` value itself is bottom `{}`.
- Value `name` is a string, containing the name of the profile. The label of this value is `{lev}`.
- Value `year` is a number, containing the year of birth of the profile. The label of this value is `{lev}`.
- Value `gender` is a boolean; it is true if the gender is female, and false otherwise. The label of this value is `{lev}`.
- Value `interests` is a list of strings that contain the interests of the profile. The label of this value is `{lev}`.

Furthermore, the level of the whole profile tuple is `{}`.
Finally, the remaining fields of the `data` tuple are:

- Value `agent` is a function that takes another profile – see Section 4.3 for the details. The label of this value is `{}`.
- Value `pid` is the process identifier of the client. The label of this value is `{}`.

The level of the whole data tuple is `{}`.

4.2 ”NEWMATCH“ messages

Messages from the server to the client should be of the form `(”NEWMATCH”, profile)`, where `profile` is the profile of the new match.

4.3 Agent specification

Each agent is a function that takes a profile and returns a tuple indicating whether they like this profile. Recall that a profile is a tuple of the form `(lev, name, age, gender, interests).`

Structurally, the returned value must be a tuple of the form `(preference, maybeProfile)`, where `preference` is a boolean, and `maybeProfile` is the unit value `()` if the preference is false, or a profile tuple if the preference is true. For example, the following agent unconditionally rejects all matches:

```
   fun rejectAll _ = (false, ())
```

In terms of security labels, the returned preference and the profile must be labeled at the level of `{lev}` of the level of the argument profile. This means that in order to produce such a value the agent must declassify parts of their profile to that level.

4.4 Server specification

When there is a match between two agents, the server should send the profiles returned by the agents to the corresponding clients' processes. For example, if Alice's agent returns a tuple `(true, alicesProfileForBob)`, and Bob's agent returns `(true, bobsProfileForAlice)`, then the server should forward the profiles to the respective processes. In this example, the server will send the message `(″NEWMATCH″, alicesProfileForBob)` to Bob's client process and `(″NEWMATCH″, alicesProfileForBob)` to Alices's client process. Observe that some declassification by the server may be necessary here, but it should be limited to the declassification of the preference booleans and the outer labels on the profile tuples. If any of the agents returns a negative preference, the server should not notify any processes.

5 Requirements and Grading

For this assignment, we ask you to implement two or more clients and one server. Two of your clients should be benign, while the others should be malicious, i.e., trying to leak information back. Do test the implementation of your clients with your server. Additionally, you should test the implementation of your server by using our service that sends out example profiles.

5.1 Report

An zip file to be submitted via Canvas that should include the following:

1. Source code to your solution including the clients and the server.
2. A report describing how you approached the solution, and the design of your server and clients. You should also detail the choice of security labels and the use of declassification operations.
3. Clear statement of contributions of each group member. It is not sufficient to write "All members contributed equally" or similar.

5.2 Grading (Max 22 points)

- Implementation of the server (10 points)
- Implementation of the server and two benign clients (15 points). Note that this task subsumes the previous one.
- Implementation of a malicious client (5 points)
- Implementation of additional malicious clients (max 2 bonus points altogether)

Observe that the points above assume that the report is satisfactory and every group member is able to explain the solution and answer questions during the live presentation of the lab. Failure to do so may result in deduction of points.