LangSec IFC

Noak Jönsson, Felipe Andrés José San Martín Irarrázabal

April 2024

1 Approach

In solving this project we spent the majority of our time figuring out the nuances of Troupe and how the security labels functioned. Initially this was done in a trial and error approach where the sample server was modified to only accept two clients which we were developing. Through numerous iterations, and with a large number of *debugpc()* and *printWithLabel* calls we were able to figure out how to structure the *agent* and *matchProfiles* functions as well as how to use the various *declassify* methods in order to have two clients be matched on the server. After we had functioning clients, we moved on to extend the server in order to have it be able to handle multiple clients.

2 Design

2.1 Server

The main *server* loop functions as follows. The *server* loop takes as argument a list of profiles, which initially is empty. When a new client sends a *NEW-PROFILE* message, the profile is compared to all the profiles in the argument profiles list using the *matchProfiles* function. The new profile is then appended to the profiles list. The server loop then calls itself with the updated profiles list.

The *matchProfiles* function takes as argument the profile, agent, and pid, of the two profiles it will compare. The two agent functions are applied with the opposite profile to produce two sets of boolean and potential profiles to be returned to the other client. If both the boolean values are true, ie both clients want to match each other, then the profiles returned from the agents, declassified so that they can be read by the matched client, are sent to the opposite clients. If one or more of the boolean values are false, then nothing happens.

2.1.1 Security Labels

As the security labels allowing the server to handle data a client has raised to '{clientName}' are set using trustmaps, the only use of declassification on the server side is when the boolean values are compared in the *matchProfile* function. As this function creates multiple branches based on secret variables, the **Blocking Label** is raised to the level of both the clients being compared. This prevents data being sent to the clients if there is a match, as the **BL** is '{client1, client2}' while the target security label of each client is only '{client1}' or '{client2}'. As the **BL** is persistent, without appropriate declassification, the **BL** would be raised for each new client added, and data to be sent to *client3* would be prevented due to **BL** being '{client1, client2, client3}'. To mitigate this, we used the *declassify_with_block* function when evaluating the condition in the *if then else* statement.

2.2 Clients

We have only created two benign clients, Alice and Bob, that are different in their profile and how their agent decides how to match other profiles. We have also created one malicious client, Malarkey, that leaks the trustlevel of other clients. The structure of the clients are the same. The *main* function sends their (profile, agent, pid) to the dating server and then starts a loop which waits for matches.

The client agent takes another profile as argument and compares it to itself and its preferences. If the argument profile meets the agents standards, the agent returns (true, profileToArgumentClient).

The agent of the malicious client will also send the trustlevel of the argument profile to itself, leaking not only the trustlevel but also the number of clients using the service.

2.2.1 Security Labels

To send data to another client the trust level on the data must be '{otherClient}' or lower. For Alice to send her profile to Bob, her agent must declassify her profile to the level '{bob}'. This is done using the *declassifydeep* function. If any branch is created using an *if* statement where Alice's data is used, this must be declassified as otherwise the **BL** will be raised to {alice, bob} which prevents Alice's profile being sent to Bob's client which only has security level {bob}. Whenever the agents makes a comparison using their own data the comparison is wrapped in a *declassify_with_block* function

3 Contributions

During the initial exploratory learning phase of the project, where we developed the barebones clients and the simple server, we both worked together for the entire process. This was where the majority of manhours was spent, and this phase was concluded once we had two functioning clients, and a server which could handle two clients. The extension of the server to handle multiple clients was done primarily by Noak.

In the report, the approach statement as well as the server design was written by Noak, while the client design section was written by Felipe.