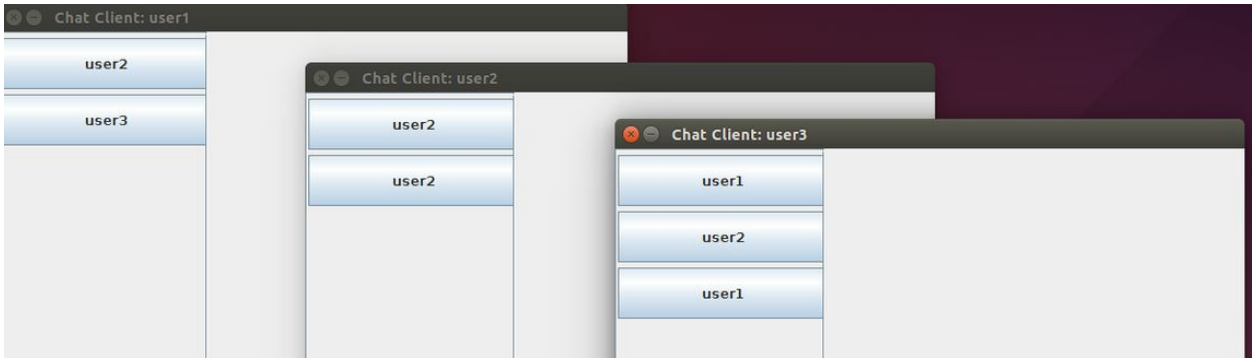
Team 13's

Audit of Group 14

Group 14 created a chat-client, provided via a zip-file. Server and clients are run via shell scripts. During our investigation of Group 14's project, we focused on how the chat service does (or does not) maintain the three principles of Confidentiality, Integrity, and Availability.

We began our investigation by stepping through the functionality of the client, looking for any obvious bugs. We were able to run the server and a pair of clients, and add the users as "friends" successfully. We saw that we were able to initiate a conversation between the two users (but didn't just yet). We discovered that, when the first user (user1) removed the second user (user2) as a friend, user1 disappeared from user2's list of friends as expected; however, user1's own list of friends did not correctly remove user2. We were then unable to remove user2, but were able re-add the users to each others' friends. Neither user was able to initiate a conversation with the other. This constitutes a failure of Availability, as the primary service of the program was non-functional. We killed and restarted user2's client, which did nothing to fix the problem. Upon killing and restarting both clients, users 1 and 2 were able to initiate conversations as expected.

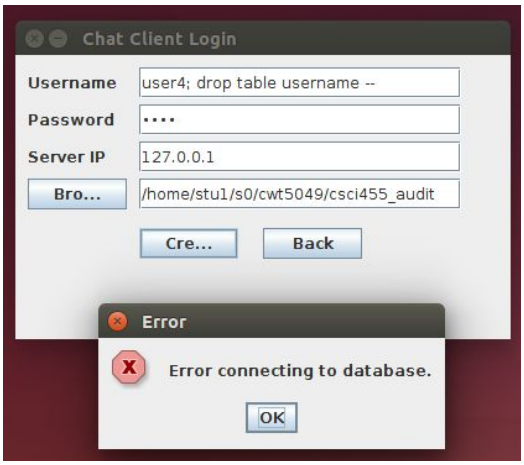


Adding and removing friends is quite buggy (None of those buttons do anything when clicked)

After playing around with the clients some more, we eventually got conversations working as expected (the trick is to never remove your friends). From there, we began investigating potential Availability issues arising from the server. From Group 14's README, we learned that "The server is using a in memory database so if the server goes down all users must regrister" [sic]. So we killed the server. Clients which were mid-conversation when the server fell were able to continue their conversations; however, clients were unable to add or remove friends, and were unable to initiate new conversations. When we tried restarting the clients, sure enough the user complained that they were not registered. Users keep some information in a database locally, so they don't lose too much… but having to re-register after the server crashes for any reason violates the principle of psychological acceptability. Group 14's README mentions that the in-memory database "can be changed to a file database easily," which would rectify this issue.

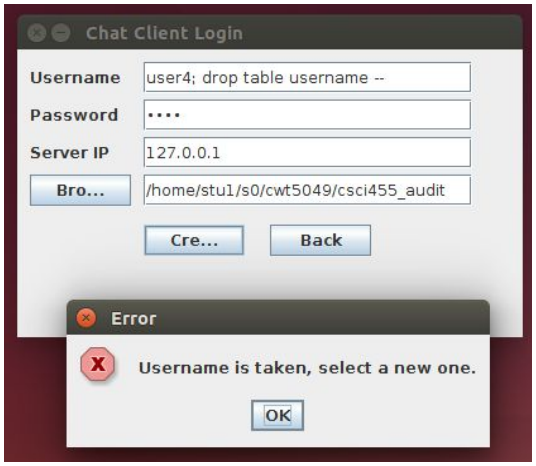
Having fiddled around with Availability some, we decided to move on to Confidentiality and Integrity. We started out with some SQL injection.

We discovered from an error that got printed out sometime during all of our fooling around that the table "USERNAME" was being used somewhere. We decided to mess with it:



First attempt

However, something curious happened after clicking “Cre…” again, with no changes to the form:



Second attempt

Upon investigating the source code, we discovered that Group 14 uses SQL prepared statements to defend against malicious parties like us. Oh well. However, during the malicious registration attempts, we got some threading exceptions from the client:

```
Exception in thread "Thread-0" java.lang.NullPointerException
        at ChatFrame.createAccountResponse(ChatFrame.java:416)
        at ServerConnection$ReaderThread.run(ServerConnection.java:291)
Exception in thread "Thread-2" java.lang.NullPointerException
        at ChatFrame.createAccountResponse(ChatFrame.java:416)
        at ServerConnection$ReaderThread.run(ServerConnection.java:291)
```

Whoops

The threading errors gave us the idea to try logging the same user into multiple coexisting clients. We could not log into more than one client for the same user, however, preventing concurrency issues arising from the use of a shared object. Protecting against SQL injection and concurrency issues uphold the concept of Integrity: unauthorized attempts to modify user data fail.

Next we checked how Group 14 protects Confidentiality. Communication between clients and server goes through SSL sockets. Group 14 mentioned in their README that they felt shaky on their implementation of the SSL keystore, but in the brief time we investigated that, we did not spot any glaring flaws in it.

Finally, Group 14 allows the user to get away with some dangerous acts. No password requirements are enforced, so users can enter very insecure passwords (short ones, ones without uppercase AND lowercase letters, numbers, and special characters, common pass phrases like “password” or simply “pass,” etc). Even if it places a slightly higher burden on the user’s memory, a minimum length and some special character checking would make user passwords much harder to guess or brute-force.

Overall, we found Group 14’s project to have good practices with regards to Integrity. It mostly had good practices with Confidentiality, the possibility for weak passwords aside. The area where this project can show the most improvement is on its availability. It is relatively easy to put in a state where two clients cannot communicate with each other, either through a server crash or one of the clients removing the other as a friend, then attempting to re-establish communication.