A Simplified BlueTrace Protocol - Report

Sebastian Chua (z5161468)

System Functionality Overview

The system built in this assignment provides a simplified implementation of the BlueTrace protocol developed by the Singaporean government.

Logging In

In this simplified implementation, a list of valid usernames and passwords are provided in a file `credentials.txt`.

TempIDs

In the interest of privacy, a user's contact log will only store TempIDs, a random string of 20 numbers generated by the server. Users will have to manually request TempIDs from the server, after which the server will maintain the mapping between a user's username and the TempID in a file `tempIDs.txt`. A user's TempID will only remain valid for 15 minutes, after which the user will need to manually request a new one from the server.

Beaconing, Contact Logs and Contact Tracing

In this simplified implementation, users will manually send a beacon to a user they come into contact with. This beacon will contain details of the user's TempID and will be stored in a file `<zid>_contactlog.txt`.

When a user tests positive for COVID-19, they will upload their contact log to the server, whereby the server will use `tempIDs.txt` to identify actual usernames in the contact log and perform contact tracing.

Prior to upload or adding beacons to a contact log, the contact log will be cleaned of any beacons that have been in the log for more than 3 minutes.

Program Design

Language: Python 3.7

Client-Side Classes

- **`Client`**
    - Receives user input for commands to perform. Also handles sending and receiving beacons, as well as writing to and cleaning contact logs of expired beacons.

Server-Side Classes

- **`Server`**
  - Parses and handles a client's requests. In charge of creating new threads for new client connections and contact tracing when a client uploads their contact log.
- **`LoginManager`**
  - In charge of logging users in and out of the system. Tracks a user's login attempts and determines whether or not a user should be blocked.
- **`TempIDManager`**
  - In charge of making TempIDs for users and maintaining them in tempIDs.txt. Also in charge of mapping a given TempID to a username.

Helper Classes

- **`DataManager` -** Used to encode and decode JSON objects to UTF-8 to be sent in the packets
- **`TempID` -** Represents a TempID. Can be used to neatly parse and work with TempID entries.
- **`ContactLogEntry` -** Represents an entry in a contact log. Can be used to neatly parse and work with contact log entries.
- **`LoginStatus` -** Enum class to help with handling and printing outcomes of an attempted login.

Application Layer Message Format

User Command to the Server

- Command indicating a client's request to the server. Will be a string that will be encoded into UTF-8 and is either:

```
logout | Download_tempID
```

Login Credentials

- Format of credentials JSON object containing a user's attempt to login to the system:

```
{ 'username': <username_input>, 'password': <password_input> }
```

- The above object is then dumped into a string using Python's **`json.dumps`** before being encoded into UTF-8 to be sent

Beacons

- Format of beacon sent to other clients via UDP:

```
<TempID: str [20 bytes]> <TempIDCreated: Date(str) [19 bytes]> <TempIDExpiry: Date(str) [19 bytes]> <Inserted: Date(str) [19 bytes]> <ProtocolVersion: str [1 byte]>
```

Contact Logs

- User first sends a message containing the length of the contact log to inform the server of how many bytes it should expect to receive. The entire contact log is then read into a string, encoded into UTF-8 and sent

Design Considerations and Tradeoffs

`tempIDs.txt`

- The current implementation appends to `tempIDs.txt` whenever new TempIDs are generated for a user. Expired TempIDs are also not removed from the file.
- As the file grows larger, mapping a TempID to a username will grow linearly with it, resulting in slow lookup times

Keeping Track of Logins

- The `LoginManager` class keeps track of users that are logged in the system in a dictionary as a class property
- Although this works as a temporary solution, utilising a database or using a flat-file could provide better persistence and functionality.

Error Handling

- The current implementation leaves out a lot of error handling in enforcing the message protocol in exchange for a simpler implementation.
- This could lead to a lot of potential security flaws in the system

Possible Improvements and Extensions

User Signup

- New users should be able to sign up to the system
- **Implementation:** Add logic to append users to `credentials.txt` or a user database

Automatic TempID Generation

- Server should automatically assign TempIDs to clients when they log in or when their TempID expires instead of having clients manually requesting them
- **Implementation:** Can have server issue a new TempID if it finds that the user's TempID has expired

Two-Way Beacon Receiving

- When User A comes into contact with User B, User A's TempID should be stored in User B's contact log. Likewise, User B's TempID should also be stored in User A's contact log as both users have come into contact with each other.
- **Implementation:** Each client should be able to maintain its own personal contact log