CS447 : Networks and Data Communications
Programming Assignment #2
Total Points: 150

Assigned Date : Wednesday, April 05, 2023
Due Date : Wednesday, April 19, 2023 →HARD DEADLINE←

Overview

For your second project, you will extend your IRC application from P1 and improve it's functionality. Specifically, P2 adds the following new capabilities:

- **PASS** message to set a connection password.
- `Server-to-Server` communication following Internet Relay Chat: Server Protocol (RFC #2813) https://tools.ietf.org/html/rfc2813 specifications.
- Authentication support.

Technical Requirements

ALL P1 technical requirements fully apply to the P2. Additionally, following new requirements should be met.

1. Support the **PASS** message listed on RFC #2812 Sec. 3.1.1. Note the recommended client registration order **PASS** → **NICK** → **USER** listed under Sec. 3.1.
2. Support `server-to-server` communication specified in RFC #2813. At minimum, it's advised to read Sections 3 and 4 of the RFC found at https://tools.ietf.org/html/rfc2813.
   - Support the following Connection Registration (RFC #2813 Sec 4.1) messages: `PASS`, `SERVER`, `NICK`, and `SQUIT`.
   - Support the following Channel Operation (RFC #2813 Sec 4.2) messages: `JOIN`, and `NJOIN`.
   - Any applicable numeric replies for the above.

   Note:

   The `PASS`, `NICK`, and `JOIN` messages have `server-to-server` communication specific parametric changes. In other words, your PR1 implementations will not work out-the-box for `server-to-server` communication.

3. Authentication and Password Management: RFCs state that passwords are exchanged in plaintext (which is not good!). Moreover, there isn't a clear strategy on user authentication either. The following simplified model is proposed as a solution.
   (a) The same `PASS` value is used for both client and server registration. For servers, use the configuration file to set `PASS`. This acts as a form of **access-key** into the IRC network but does not provide user authentication. For now, we'll use it as a viable `server-to-server` authentication (only).
   (b) For client authentication, modify the `NICK` command to add a user-specific password as follows. `NICK <nickname> :<password>` (e.g., `NICK Wiz :wizpassword`).

(c) Upon successful registration, the client will receive an RPL_WELCOME (as per RFC). **NOTE**: During testing, we will use `QUIT` and force the client the authenticate before proceeding with anything else.
(d) Passwords are encoded in Base64 before they are sent from a `client` to a `server` or exchanged between `servers`.
(e) The receiving `server` will decode the received password → suffix `S23447P2` to it (a.k.a salting) → encode the salted password in base64 → store it in hidden password file. Each server will maintain it's own hidden password file named "`.usr_pass`", and store each nickname against a password; we are loosely mimicking how the `/etc/shadow` file works in Linux for password storage.
(f) On all successive connections, the receiving `server` verifies the received password against the stored salted password by repeating the process mentioned above in step 3e and comparing the values.
(g) In the interest of time, we'll bypass `server-to-server` authentication. However, if you propose and implement a viable method, we'll consider it for extra credit.

4. Only successfully authenticated `clients` and/or `servers` are allowed to proceed to use IRC features.
5. Each client only knows about and connects to one server and one server only.
6. Finally, use `tcpdump` https://www.tcpdump.org/manpages/tcpdump.1.html to capture server->server traffic. Provide screenshots in your report.

Logistics

All P1 Logistics requirements applies to P2. The following additional requirements are listed mostly for clarification purposes.

1. Address any pending issues from P1 before starting P2 implementation.
2. Base64 encoding/decoding, hidden file creation, and log file creation should be done programmatically.
3. Print all reply codes on all interactions to the standard output.

4. At the end of your implementation, you should be able to:
   - Compile and run your code on a typical Linux machine(s). Include a `readme` file with clear compilation instructions and any additional software the grader might have to install.
   - Run your `server` programs first.
   - Run two or more `clients` and register with different `servers`.
   - Properly authentication and/or re-authenticate upon each connection initiation.
   - Create channels. Engage with other users through IRC.
   - Have private chats between clients through server relays.
   - Exit the `client`(s) gracefully.
   - Exit `servers` gracefully through `SQUIT`.

5. It is recommended to test your system with at least 3 servers. Bare minimum, you should test your implementation with 2 servers and more than 2 clients. Also, verify the behavior of the overall IRC when a `server` exits using `SQUIT`. Read Sections 3 and 5 of RFC #2810 (https://tools.ietf.org/html/rfc2810) and incorporate some of the case studies listed therein in your testing.
6. Runtime arguments are fed using the appropriate config files (just like in P1). NOTE: Given the nature of the assignment, each server will have it's own configuration file with different values. Specifically, there will **`SOCK_ADDR`** line for each known remote server. As an example, consider a particular server which is connected to two other servers. The corresponding config file will look like the following:

server_1.conf

```
NICK=babbage
PASS=supersecretpassword
PORT=45345
SERVERS=2
SOCK_ADDR=192.168.0.14:1245
SOCK_ADDR=192.168.0.19:62012
```

Instructions

- **Start early!!**. This is a fairly loaded assignment for 2 weeks.
- **Take backups of your code often!!**. Practice good version control habits
- Follow a good coding standard. Use one of Google's coding standard found here `https://google.github.io/styleguide/`, if you don't already follow one.
- In addition to sections explicitly listed above, it is recommended to read other relevant sections of the RFC #2810, #2812, #2813 before resorting to ask the instructor.
- This assignment is fully intended to be implemented with just the `socket` package and the `base64` package. Use of any other package, except basic I/O, regex, or string parsing packages, without the instructor's explicit permission, will nullify your submission.
- Absolutely do not include executables, folders created by your programs, hidden files, version control repositories, or any irrelevant files in this tarball. All project relevant file formatting standards (PDF, README, .txt, .tgz) will be strictly monitored and are subject to penalties.
- The due date of this assignment is **Wednesday, April 19, 2023 (HARD DEADLINE)**. A Moodle dropbox will be opened for submission on Moodle.

Deliverables

A complete solution comprises of:

- A short report of the design and implementation of your system. The report should be **`PDF`** format. At a minimum, your report should include the following sections:
  - Introduction: Your objective and what you hope to gain from the assignment.
  - Overall design, specific design choices, and reply codes used.
  - The output of a sample run (including screenshots where applicable).
  - Summary and Issues encountered. What you were able to achieve from your own objectives (from the introduction) as well as project specifications. Make sure to explicitly list functionality you failed to implement (or buggy).
- A compressed tarball that contains:
  - a directory containing (only) your source code and config files. **Do not** include executables, folders created by your programs, or any other files not specifically listed here as required.
  - A short `readme` file with compilation instructions.
  - A `makefile` to automate compilation.

To create a compressed tarball of the directory source, use the following command:
`tar -zcvf siue-id-pr2.tar.gz source/`.
e.g. `tar -zcvf tgamage-pr2.tar.gz P2/`

Collaborating on ideas or answering each other's questions is always encouraged. Most times, I find that you learn a lot from your peers. However, do not share/copy/duplicate code from others, including online sources. The exercise is meant for you to learn network programming, not to test your googling abilities. Issues related to academic integrity and plagiarism have **ZERO** tolerance.

Useful Resources

- Linux Man pages – found in all linux distributions
- Beej's Guide to Network Programming – A pretty thorough free online tutorial on basic network programming `https://beej.us/guide/bgnet/pdf/bgnet_usl_bw_1.pdf`
- Internet Relay Chat: Client Protocol RFC #2812 `https://tools.ietf.org/html/rfc2812`

- Internet Relay Chat: Architecture RFC #2810 https://tools.ietf.org/html/rfc2810
- Internet Relay Chat: Channel Management RFC #2811 https://tools.ietf.org/html/rfc2811
- Internet Relay Chat: Server Protocol RFC #2813 https://tools.ietf.org/html/rfc2813
- The University of Chicago χ-Project http://chi.cs.uchicago.edu/chirc/irc.html
- Base64 encoding/decoding:
  - C/C++
    - openssl http://fm4dd.com/openssl/manual-crypto/BIO_f_base64.htm
    - glib – https://developer.gnome.org/glib/stable/glib-Base64-Encoding.html
    - GNU coreutils http://www.gnu.org/software/coreutils/coreutils.html
  - Additional Resource – http://rosettacode.org/wiki/Base64_encode_data#C

  If you use any other resources, make sure to cite those in your report. Using online resources does not mean you are allowed to copy and use someone else's code for your purpose. Such incidents, if detected, will be treated as academic dishonesty.