CMSC 417 Computer Networks **Spring 2016**

Project 3: Reverse Engineered Chat Server

Assigned: March 22 *Due: April 8th, 11:59:59 PM*

1 Introduction

In this project, we will be providing a chat client and will host a chat server. Clients allow users to communicate with one another by connecting to the server and interacting with it in accordance to an application protocol. Through this protocol, the server allows clients to engage in a group chat and send private messages to one another.

Your task will be to reverse engineer the chat server and its protocol and use this information to write a compatible replacement. You may find it helpful to explore packet capture tools like Wireshark or `tcpdump` to help you capture and parse the messages the reference client and server exchange. You can download Wireshark here: `https://www.wireshark.org/#download`

2 Client (Provided)

The client will be available in the `bin` directory of your project once you clone it.

While running, the client takes commands directly from the user. All commands are preceded by a backslash. Not every command is available in every context. The client supports the following commands:

1. **\connect <IP>:<port>** = Instruct the client to connect to the provided chat server. If the IP and port are omitted, they default to `128.8.126.45:23456`, which is where our hosted server will be running. If you want to use the client to test your sever replace that with the address of your server. *Note: 127.0.0.1 is an alias for localhost, so 127.0.0.1:<port> will connect to that port on the same machine you're running the client.*
2. **\disconnect** = If connected to the server, disconnect.
3. **\list** = List all users connected to the server.
4. **\msg <User> <Message>** = Send a private message to the specified user. User must be less than 256 characters in length, and the Message must be less than 65536 characters in length.
5. **\nick <Name>** = Set your nickname to the specified name. Name must be less than 256 characters in length.
6. **\exit** = Exit the chat client.

All other input is interpreted as a message being sent to the main group chat.

2.1 Valid input

Valid user/nick names are assumed to contain only non-whitespace, printable ASCII characters.

Valid chat messages must start with a non-whitespace, printable ASCII character and then consist of printable ASCII characters after that.

You can find a list of printable ASCII characters here:
`https://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters`

3 Server (Hosted)

We will be running a server at 128.8.126.45 and listening on port 23456.

4 Replica Server Implementation

As long as you comply with the requirements in Section 6, you may write this project in a language of your choosing. We will be installing support for python, ruby, java, C and C++ on the VM. If you feel that you need additional language support and the VM doesn't support it, please let us know and we may (but also may not) accommodate your request. Make sure to test that your code compiles and runs in the VM using `make` and `bash run.sh` so that our tests will also be able to compile and run it.

The URL to accept the assignment is:

`http://ter.ps/b6b`

The process of adding the Java OpenJDK to the VM has made vagrant up take longer than before. Even with the trust64 base box pre-cached, it can take 5–10 minutes or for it to come up.

Once you get it up, the current versions of ruby, python, java, gcc and g++ installed are as follows:

```
vagrant@vagrant-ubuntu-trusty-64: $ ruby -v
ruby 1.9.3p484 (2013-11-22 revision 43786) [x86_64-linux]
vagrant@vagrant-ubuntu-trusty-64: $ python --version
Python 2.7.6
vagrant@vagrant-ubuntu-trusty-64: $ javac -version
javac 1.7.0_95
vagrant@vagrant-ubuntu-trusty-64: $ gcc --version
gcc (Ubuntu 4.8.4-2ubuntu1 14.04.1) 4.8.4
vagrant@vagrant-ubuntu-trusty-64: $ g++ --version
g++ (Ubuntu 4.8.4-2ubuntu1 14.04.1) 4.8.4
```

Your replica server must support a single optional argument specifying the port on which it will run as a decimal integer. Since your sever will be called by your `run.sh`, it can accept that in whatever format you like, but your run.sh must accept the single parameter:

- **<port>** = The port that the server will listen on. Represented as a base-10 integer. If omitted the server can listen on whatever port you like—we will always provide this argument in our tests, but you can have a default value to make your local testing easier.

5 Grading

Your project grade will depend on how much client functionality is maintained when connecting to your server. We will test your server by running clients and directly comparing the output with identical input against your server and the reference server.

We strongly encourage you to write tests against the reference implementation to compare against your own implementation.

6 Additional Requirements

1. Your code must be submitted as a series of commits that are pushed to the origin/master branch of your Git repository. We consider your latest commit prior to the due date/time to represent your submission.

2. You must provide a `Makefile` that is included along with the code that you commit. We will run `make` inside the root of your repository to compile your code. It is your responsibility to write a `Makefile` that compiles your code (if necessary) regardless of the language, e.g., call `javac` appropriately if you're using java. *If your code doesn't need to be compiled, e.g., python or ruby, you can use a makefile with a single line "skip:" that will do nothing when we call make in our tests.*

3. You must provide an executable `run.sh` shell script that calls your sever wherever you generate it. We will call `bash run.sh <port>`, so make sure it works with bash. *Note: a run.sh with the single line "<command> $1" will call that command and pass through the first argument if specified, allowing you pass the port.*

4. You must submit code that compiles and runs in the provided VM, otherwise your assignment will not be graded.

5. You are not allowed to work in teams or to copy code from any source.