SSH Client

ECE 4303 - Spring Semester 2019

Pasha Belgar

Alex Chan

Franklin Cortez

Abel Guzman

Andy Klaib

Priscilla Luu-Villa

Krystal Wang

Due: May 2nd, 2019

**SSH Client Table of Contents**

Contents

[I. Introduction 3](#_Toc7531482)

[II. Design 3](#_Toc7531483)

[III. Implementation 4](#_Toc7531484)

[IV. Primary Constraints 4](#_Toc7531485)

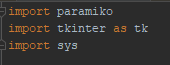
[V. Presentation Summary 5](#_Toc7531486)

[VI. Conclusion 6](#_Toc7531487)

[VII. References 7](#_Toc7531488)

# Introduction

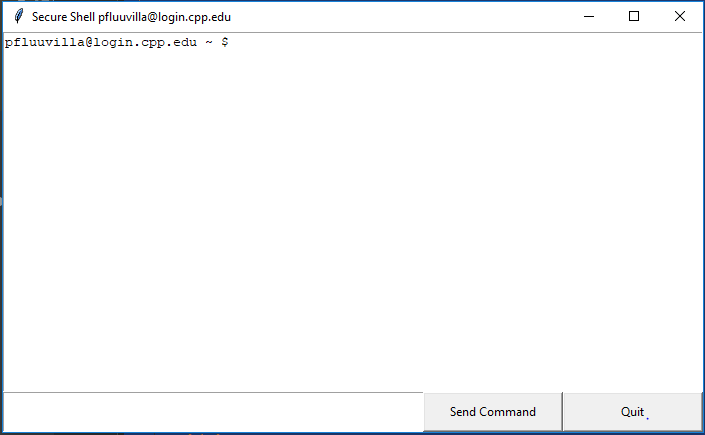
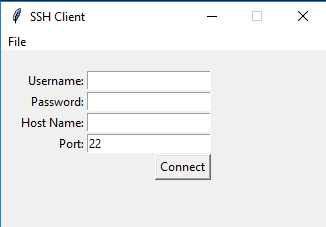
The goal for this project was to create a SSH Client, similar to PuTTY, using any programming language. Our group chose to create a SSH Client using Python 3. To create the SSH connection, we leveraged the Parmiko library and functions that exist in the respective library. To create the GUI, we used the Tkinter library and its respective functions.



# Design

Our design for the GUI of our SSH Client was inspired by PuTTY. We wanted the user to input the hostname they wish to connect to, their username, and their password to create the connection. Additionally, the user could change the port they wish to create the SSH connection on but, by default, the port value was set to 22.

The shell that is created when the connection is made was designed to be similar to the command prompt or terminal. The shell was created with a textbox that appended the output of the command sent to the host while displaying the command sent by the user. Again, this shell was designed with PuTTY in mind and has a similar format.



# Implementation

|  |
| --- |
| def sshConnect(hostname, port, username, password):  client.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())  client.load\_system\_host\_keys()  client.connect(hostname, port, username, password)  def sshCommand(command):  stdin, stdout, stderr = client.exec\_command(command)  shell\_frame.insert('end',stdout.read()) |

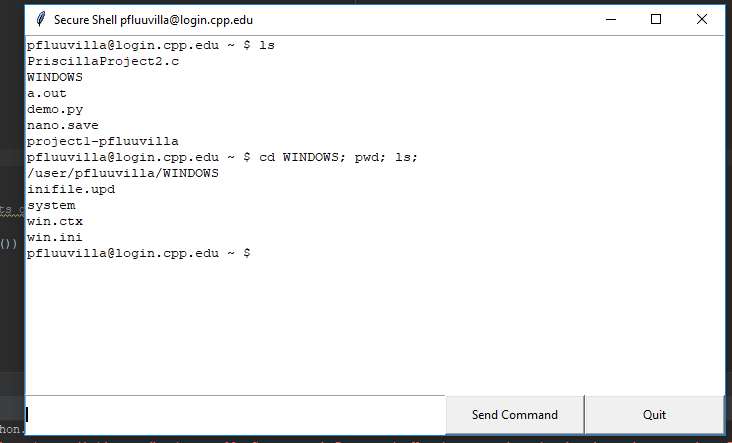
For our SSH Client to function properly, we leveraged two functions. The two functions, sshConnect and sshCommand, would create the SSH connection and send the SSH command, respectively. The sshConnect method created the connection for the user. However, the connection would only be created and there would not be a way to input commands. In our implementation, the “Connect” button used in the first window would create the SSH connection. The sshCommand method would send a command the user would input to the hostname the user connected to. The output of the command would be appended to the end of the textbox in the shell window. In our implementation, the “Send Command” button used in the shell window would use the “send” method we created, which incorporates the sshCommand method.

The remaining code we wrote to create the SSH client was centered around the GUI. Since we had to create each object seen in the UI and place it on a window, we spent a large portion of our time designing the client and ensuring the UI was clean and intuitive to use. Additionally, we added a menu with saved values for SSH. This idea could be furthered by creating save files for the input values, such as the user’s username, password, and host they wish to connect to, to a text file and opening or importing the values through the menu for a faster connections to frequently used hosts.

# Primary Constraints

Although our SSH client was successful in its implementation, there were some constraints with the way we send commands and create a connection. Since we are using the Paramiko library to create the secure shell’s connection, the amount of code that is written was significantly reduce. However, we noticed there were some conflicts with using the “cd” command and navigating the path of the host we connect to. Upon further research and parsing through the Paramiko manual, we learned about some constraints with using the Parmiko. The main constraint we faced with Paramiko is the SSH connection we create is a single instance. This means that the commands we send the the host is a single instance of the connection between the user and the host. Therefore, commands such as ‘cd’ would change directories for the instance the command was sent but not persist for future commands.

The solution we have for this constraint is to send multiple commands on a single line. For example, if a user wanted to change directories and print the new working directory, the user would type “cd [directory]; pwd” and the output would be the path the the directory the user changed to. To improve our client, we could create special cases for these types of instructions. However, for the purpose of this project, we believed our solution was sufficient.



# Presentation Summary

During our presentation, in class, we went over the coding language we chose to create our SSH client with. Additionally, we briefly went over the libraries used in conjunction with Python as well as the main functions used to create the client. One part we briefly touched upon but did not go too into details with was the GUI. Since we believed explaining why we wrote each line to create a button or window was out of the scope of the presentation, we chose to address the GUI portion of the code in this report. After explaining the tools we used to create our client, we proceeded with a live demonstration. In the demonstration, we showed the class how to navigate the first window and the file menu. We showed our pre-configured SSH inputs and used it to speed up the process of inputting our credentials for the SSH connection. After we input our password, we clicked the “Connect” button which opened another window, the shell. In the shell, we illustrated how it was created to mock the shell created by PuTTY when the connection was made. From the shell, we demonstrated how to send commands and showed the output we received from the server. After demonstrating a few commands, we used the “Quit” button we created to terminate the connection and close the client.

# Conclusion

We believe our project was successful in allowing us to learn more about TCP/IP and how SSH works. Since we worked with PuTTY before we began our implementation and researched different ways to implement SSH, we learned the basics behind SSH and how it is used to create a secure remote connection to a host over a network.

Additionally, we familiarize ourselves with some linux shell commands that were used in our demonstration and during testing of our client. With the knowledge and experience we gained from our implementation we understand the difficulty of creating a robust client such as PuTTY. There are many opportunities for us to continue work on this client to improve its functionality.

# References

[1] “Tkinter - Python Interface to Tcl/Tk.” *Tkinter - Python Interface to Tcl/Tk - Python 3.7.3 Documentation*, docs.python.org/3/library/tkinter.html.

[2] “Welcome to Paramiko's Documentation!.” *Welcome to Paramiko's Documentation! - Paramiko Documentation*, docs.paramiko.org/en/2.4/.