Computer Networks Project: MySSH

Ionuț Roșca

Facultatea de Informatică,
Universitatea "Alexandru Ioan Cuza",
General Berthelot, 16, IAŞI 700483, ROMANIA
https://www.info.uaic.ro/en
ionut.rosca@info.uaic.ro

Abstract. `MySSH` is a simple implementation of the SSH program and has the ability to handle pseudo-terminals over network and encrypt its traffic. It also has an optional feature to use an One-Time-Password (OTP) for connecting to the server. It is composed of two components: the client and the server. The server is concurrent and accepts multiple remote clients for which it allocates a pseudoterminal and encrypts their communications.

1 Introduction

`MySSH` is a not-so-sophisticated *SSH*-like implementation. It has the ability to make pseudo-terminal connections with remote clients, upgrading from a basic shell to a powerful one, just how *pty.spawn*() [1] does in Python. It also encrypts its traffic between the server and client(s) using the OpenSSL library. As an optional feature, enabled by default, both the server and the client can use an One-Time-Password (OTP)[7]. Being implemented concurrently, it can handle multiple clients at the same time. In the *Technologies Used* section will be presented to the reader the decisions made regarding network communication, whether to use a communication-oriented or connectionless communication model. A digram that will try to hide the implementation details and give a high overview of both components of the program: the client and the server, and how they interact with each other, will be shown to the reader in *Program Architecture* section. Having seen the high picture, the reader will become aware of the implementation details of the components of the project, the server and the client, each on its own subsection in the *Implementation Details* section. Last but not least, the conclusions will be reflected in the *Conclusions* section.

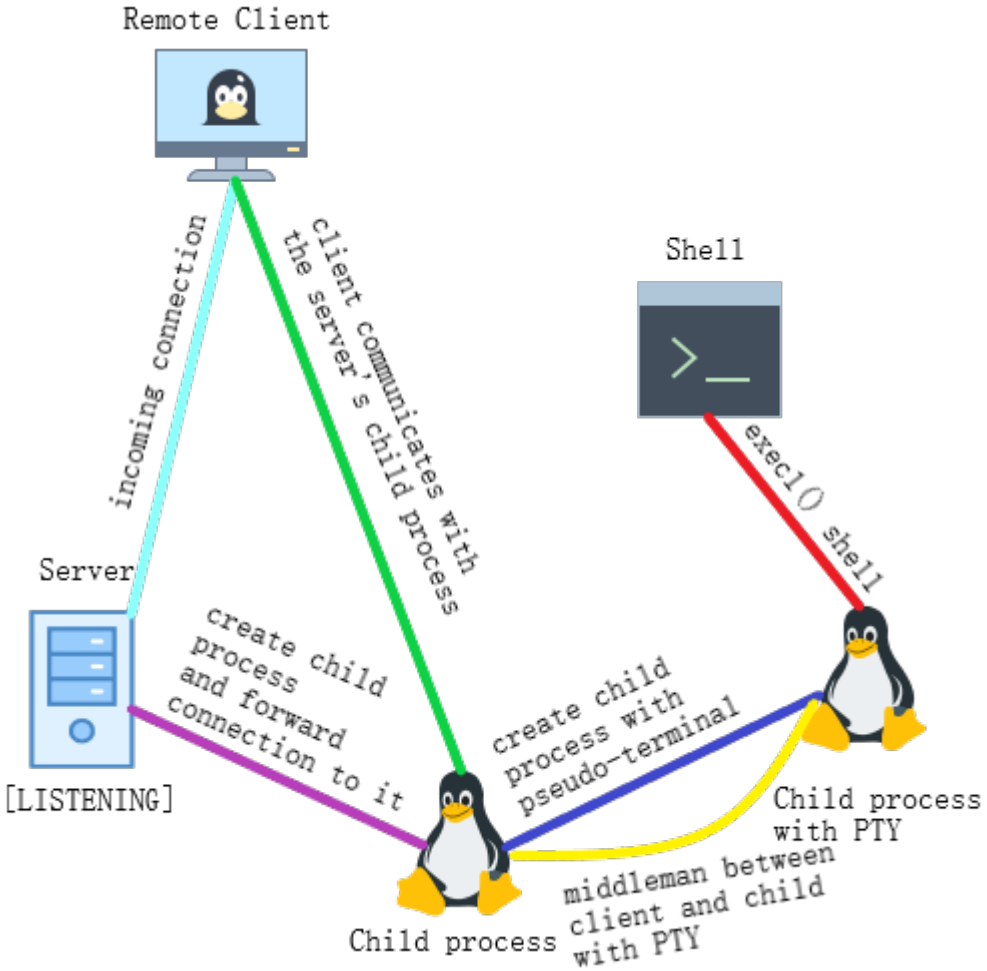
2 Technologies Used

For transmitting data between the server and the remote client(s), the procol used is the Transmission Control Protocol (TCP). Thus, `MySSH` uses a *connection-*

[1] Documentation can be found here: https://docs.python.org/3/library/pty.html#pty.spawn

oriented model for communication over the network. The Transmission Constrol Protocol (TCP) was choosed because of the necessity of transfering *every single byte* from the server to the remote client(s) and vice-versa. Neither the server, nor the client(s) could afford to lose data. Other than that, regarding the inner workings of the programs, the client program uses just a single thread (the main thread) alongside with I/O Multiplexing for communicating with the remote server. The server program uses multiprocessing for handling connections with multiple clients, signal handling for signalling client disconnects in order to deallocate resources and I/O Multiplexing for communicating with the clients. In both the server and the client programs, the communications are encrypted using functions from the OpenSSL API.

3 Program Architecture



4 Implementation Details

At the heart of this project stand the two executable binary programs: the server and the client. In the next two subsections, the inner-workings of the aforementioned programs will be analyzed.

4.1 The Server

The server is reponsible for dealing with the remote clients. The server employs the following mechanisms for doing its job:

- *Multiprocessing* for delegating the responsability of handling the connection with a remote client to a children process.
- *Pseudo-terminals* for providing a "pipeline" which can be used to remotely write/read on the server's terminal. For this to work, the *forkpty()* function call is used in the server's child process (so, this process will be the server's nephew) and on the child side, it will *execl()* a shell and on the parent side the *master* side of the pseudo-terminal will be used to communicate with the pseudo-terminal.
- *Signal Handling* for resource management. After each remote client disconnects, the resources must be freed up. This consists of "reaping" two processes: a) the children process responsible with handling the connection with the client and the children process of process a) who is responsible with providing pseudo-terminal access to the remote client. After each of the two children dies, they will sent a SIGCHLD signal which will be handled in the following way: the parent process with handle reaping of child process a) and the child process a) will handle reaping the children process b). This mechanism is achieving by setting two different signal handlers in the parent and children.
- *I/O Multiplexing* for monitoring the descriptors associated with the remote client and with the forked process running the pseudo-terminal.
- *Encryption* using the OpenSSL library's API, which provides secure data transmission over the network between the server and its clients.
- Optional *One-Time-Password (OTP)* which can be used for authenticating remote clients to the server. For each client which connects to the remote server, the server will generate an *OTP* which should be delivered to the corresponding client. If the client fails to enter the correct *OTP*, the server will terminate the condition. The *OTPs* are generated using high-quality random data provided by the Linux kernel.

4.2 The Client

The client is responsible with the initiation of the connection with the remote server and terminal multiplexing. The client employs the following mechanisms for doing its job:

- *I/O Multiplexing* for monitoring descriptors associated with the remote server and standard input.
- *Low-level terminal I/O* by using the *termios* library, whose API provides methods of changing low-level terminal behavior such as disabling echoing or sending characters byte-by-byte instead of line-by-line.
- *Encryption* using the OpenSSL library's API, which provides secure data transmission over the network between the client and the server.

4.3 Pro tip

When connected to the remote server using the client, the reader may observe that the terminal content type is lower than it should actually be.

Many devices assume 80x24 and a terminal type of VT100, but if you're connected to a Linux (or other Unix) based machine, you can make use of larger window sizes and use more advanced terminal features.[10]

For a more complete experience, the users of `MySSH` can do the following:

```
# Go to another terminal and type (without $)
$ stty -a
# Now, take a note of 'rows' and 'cols' values
# Then, go back to the client and type the following,
# replacing the <...> fields accordingly:
$ stty rows <rows from stty> cols <cols from stty>
```

5 Conclusions

`MySSH` provides the basic features needed for securely communicating remote, by providing strong encryption mechanisms provided by the OpenSSL library and also verification methods by providing a method to identify remote servers using certificates. Also, `MySSH` allocates for each remote client connected to it a pseudo-terminal which facilitates the use of invaluable terminal features like *TAB completion*, generation of signals like SIGINT (CTRL + C) using keyboard keys combination, colors and also provides specific shell features like suggestion of commands (available, for instance, in the *fish* or *zsh* shells).

References

1. Made of Bugs
   A Brief Introduction to termios. https://blog.nelhage.com/2009/12/a-brief-introduction-to-termios/
2. Coding Freak
   Blocking signals. https://codingfreak.blogspot.com/2009/10/signals-in-linux-blocking-signals.html
3. Stackoverflow
   Different signal handlers for parent and child https://stackoverflow.com/a/49404439/10604490

4. Linux man page
forkpty(3). https://linux.die.net/man/3/forkpty
5. Stackoverflow
Checking the status of child process in C. https://stackoverflow.com/a/5278627/10604490
6. Doomsday Vault
Beyond pty.spawn. https://x-c3ll.github.io/posts/forkpty-dnscat2/
7. okta
What is a One-Time Password (OTP)? https://www.okta.com/blog/2020/06/what-is-a-one-time-password-otp/
8. OpenSource
Getting started with OpenSSL: Cryptography basics. https://opensource.com/article/19/6/cryptography-basics-openssl-part-1
9. Support Microfocus
How to create a .pem file for SSL Certificate Installations https://support.microfocus.com/kb/doc.php?id=7013103
10. Decisive Tactics
Setting the Window Size and Terminal Type https://www.decisivetactics.com/support/view?article=setting-the-window-size-and-terminal-type