CS-412 Grass project report

Alexandre Chambet, Guillaume Michel, Xavier Pantet

May 2019

1 Project documentation

As intended, our project follows a standard client-server architecture. For brevity of this report we will not describe how they work internally, but we rather focus on high level implementation principles. The client is fairly simple. Essentially, it reads commands from the command line or the infile, forwards them to the server and prints the response either on the screen or in the outfile.

1.1 Client-server communication

The client and the server communicate through native c++ sockets. The client first initiates a connection to the server on the specified port. Then the server allocates a socket to handle the client and generate a unique identifier for it. This will be further useful to perform client authentication.

1.2 How files are exchanged

Files are divided into chunks of 2048 bytes. Then those chunks are sent one by one in a sequential manner over the network. The client received them and reassemble them by just merging them together to recreate the files. Also, in order to respect the requirements, threads are opened both on the client and the server to receive or send the file. This permits to asynchronously send files while also sending commands.

1.3 How commands are handled by the server

Each command reaching the server passes through a bunch of layers before being actually executed. All logic related to commands stands in *Commands.cpp*. The entry point is the *exec* method. First commands are sanitized. We check that the command exists and is implemented and that the user connected to the current socket is allowed to run the command. Finally, the method running the command is called, and we check a few more things about the parameters such as: parameter presence/absence and parameter format. All commands, except the vulnerabilities, are run through *execve*, to avoid unwanted command injections.

To handle users and authorization, we designed an *authentication service* as well as an *authorization service*. Authentication works directly with the sockets. Since every one of them is assigned a unique identifier, the authentication service keeps track of users connected to the different sockets. Each user then has an authentication status. The authorization service takes the user and the command as inputs and checks that the user has access to the command.