Implement mySU for Getting Root Shell

Now we know how to inject our code into the Android system and gain the root privilege, but we have not completely achieved our ultimate goal yet. An important reason for users to root their Android devices is to execute any command of their choice using the root privilege. When building the OTA package, the commands are already decided; if users want to run other commands after the programs in the OTA package is executed, they will not be able to do that, unless they can get a shell that runs with the root privilege. Such a shell is called root shell.

We can launch the root shell using the methods from the previous tasks, but that is problematic: shell programs are interactive, meaning they will not terminate unless users type an exit command; this will stop the system booting process, so the OS will never be able to complete its booting sequence. The interesting question is how to run something non-interactive during the booting process that enables us to get an interactive root shell later on.

If we were on a typical Linux system, we can easily solve the above problem by using the chmod command to turn on the Set-UID bit of any shell program (e.g.bash) that is owned by the root. Later on, when any user runs this shell program, the shell will run with the owner's (i.e. root) privilege. Un-fortunately, for security reasons, Android has removed the Set-UID mechanism from its underlying LinuxOS since version 4.3 (API Level 18). The official document of security updates on Android 4.3 says the following: "No setuid/setgid programs. Added support for filesystem capabilities to Android system files and removed all setuid/setguid programs. This reduces root attack surface and the likelihood of potential security vulnerabilities."

Another approach is to start a root daemon during the booting process, and then use this daemon to help users get a root shell. This is the approach used by some of the popular rooting OTA packages, such as SuperSU developed by Chain fire. In this task, students will write such a daemon and use it to understand how it helps users to get a root shell. The main idea of this approach is quite simple. When users want to get a root shell, they run a client program, which sends a request to the root daemon. Upon receiving the request, the daemon starts a shell process, and "give" it to the client, i.e., allowing users to control the shell process. The tricky part is how to let the user control the shell process that is created by the daemon.

For users to control the daemon-generated shell process, they need to be able to control the standard input and output devices of the shell process. Unfortunately, when the shell process is created, it inherits its standard input and output devices from its parent process, which is owned by root, so they are not controllable by the user's client program. We can find a way to let the client program control these devices, or we can do it in a different way by giving the client program's input and output devices to the shell process, so they also become the input/output

devices for the shell process. This way, the user has a complete control of the shell process: whatever the user types in the input device of the client program will also be fed into the shell process; whatever the shell process prints to its output device will be showing to client program.

Writing the code to implement the above idea is not easy, as we need to have two essential pieces of knowledge: (1) how to send the standard input/output devices (file descriptors) to another process, and(2) once a process receives the file descriptors, how it can use them as its input/output devices. We provide some background knowledge regarding these.