how you arrived at your ﬁnal exploit code:

This is how the program works: it opens a file, and checks its number of chars. Then, it puts the chars to a buffer and writes it on the Terminal with some lag. The problem is the program just checks the number of the chars at the first of the program, and it does not check the that when it wants to write those chars to the buffer. Furthermore, it writes those characters in the Terminal very slowly. Therefore, it gives some time to the attacker to replace the file during the time the program is writing the chars in the Terminal. Hence, the attacker can change the file and put malicious code in the file; therefore, put malicious code in the stack. Furthermore, he/she can override the buffer and overwrite the return address, and put the address of the malicious code in it. This is how I made my sploit. My sploit runs correctly when I run it on my local fancy\_echo. However, because when I automate my sploit, address of buffer changes, I could not find the address of the buffer to use it and writes it to my return address. The way my sploit works is: I put some nop sleds at the beginning of the buffer, and then put the shellcode at the middle, and then some other nop sleds, until I reach the return address, and overwrites it with the address of the shellcode. I find all the address with gdb tool.

how the vulnerability could be repaired:

This is a TOCTTOU attack. The way you can prevent it is to check everything when you want to use it. For example, in our case, the developer should take a copy from the file, and check the copy with the original file whenever he/she wants to write something on the buffer, hence in the Terminal. Another thing that the developer can do is to restrict the privilege of changing the file during the time the program is trying to write the characters from the file.