**AC31012 Assignment 1**

**Team Name**: Aberdeen68+ **Members**: Aldrin Baretto, Mikolaj Olejnik, Jordan Keiller

**Vulnerability/Bugs**

In order to use our covert backdoor, we have decided to make our program vulnerable to a stack buffer overflow attack. This form of attack can only be done after we disable the stack protector in our makefile. The vulnerability itself comes from the use of unsafe C functions. In our case, we have used scanf with the variables usernameInput and passwordInput. Specifically, we have chosen to omit the optional width specifier. Since we haven't specified the length, there is no bounds checking for what we input into those variables. We can abuse the space allocated in memory to either of the scanf variables. Each of them has a certain number of bytes allocated to them, followed by a return address. The return address is the next memory location the program accesses after the call to the scanf function is complete. Since there are no bounds checking, we fill the buffer with characters until we reach the return address pointer, and then overwrite it to the address of the call function for authSuccess(), which calls authenticated() despite not knowing the password.

**Vulnerability Trigger**

The attacker must have the program's source code and the makefile to investigate and attack. To compile the program, the –fno-stack-protector flag must be used to remove the built-in stack protection within gcc. The attacker must use gdb which is a debugger in order to better exploit the vulnerability. After compiling, the attacker must run gdb on the binary file and disassemble the main function to find the memory address of the authSuccess() function. Breakpoints must be added before and after the scanf input function. The purpose is to show us a snapshot of the buffer memory before and after we attempt to overload it. When we run the program, the goal is to enter the right number of characters to fill the buffer up to the point just before we overwrite the return memory address. This specific process is trial and error since we must get the amount of filler characters just right. The attacker now needs to find a way to correctly overwrite the return address. To do this, a python script can be created which will write the right amount of filler characters and include the memory address of the authSuccess() function in little endian notation to a text file. The text file will act as a payload. When the script is executed, the contents of the file will be converted into unicode characters, with their code corresponding to the hex numbers required to recreate the memory address of authSuccess(). The attacker now runs the program in the debugger using the payload as input and sees that the return memory address of the buffer has been overwritten causing the program to run the authSuccess() function, thus granting access without knowing any passwords.

**Detecting Vulnerability Bugs**

For an individual to detect bugs in our vulnerability, they would need to have somewhat of an understanding of the gdb, memory addressing and C/C++ language and know that in some parts of our program, it utilizes unsafe functions that would otherwise not be used in normal programming. This and the understanding of a buffer overflow provides for a dense topic that requires a lot of time and effort to research exactly how someone could compromise these bugs to gain access through the backdoor. In the program, we also imported a few red herring libraries that aren't used and misleading commenting to throw people off looking for these types of bugs, or indeed, that we are doing a buffer overflow-style attack. In the early stages we were also considering implementing a dangling pointer reference, however we decided to switch a buffer overflow as we were able to find considerably more sources and explanations for the latter.