Graded Assignment: Homework 3

Mark Kardash

University of Maryland Global Campus

SDEV 325 – Detecting Software Vulnerabilities

Executive Summary:

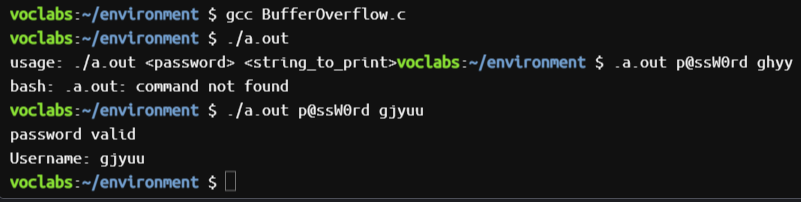
For this assignment, the topic of which was demonstrating and mitigating various software vulnerabilities, I decided to focus on two vulnerabilities that have to do with bytes and functions. Specifically, those are CWE-120: Buffer Copy without Checking Size of Input, and CWE-676: Use of potentially dangerous function. The first one occurs when the message copied is too long, therefore “overflowing” the buffer. The second, as the name implies, involves using a function that makes the code vulnerable to attacks. I believe that I have managed to mitigate both vulnerabilities successfully, even though the second code I wrote did not function properly.

Example 1 - [CWE-120: Buffer Copy without Checking Size of Input]

Overview:

To demonstrate this vulnerability, I created code that takes and stores a password inside an array, and then prints either a validation or an error message for the password, along that the user enters. The code is written in the language C, and its use of the function “strcpy” makes it vulnerable to overflow attacks (More on this below).

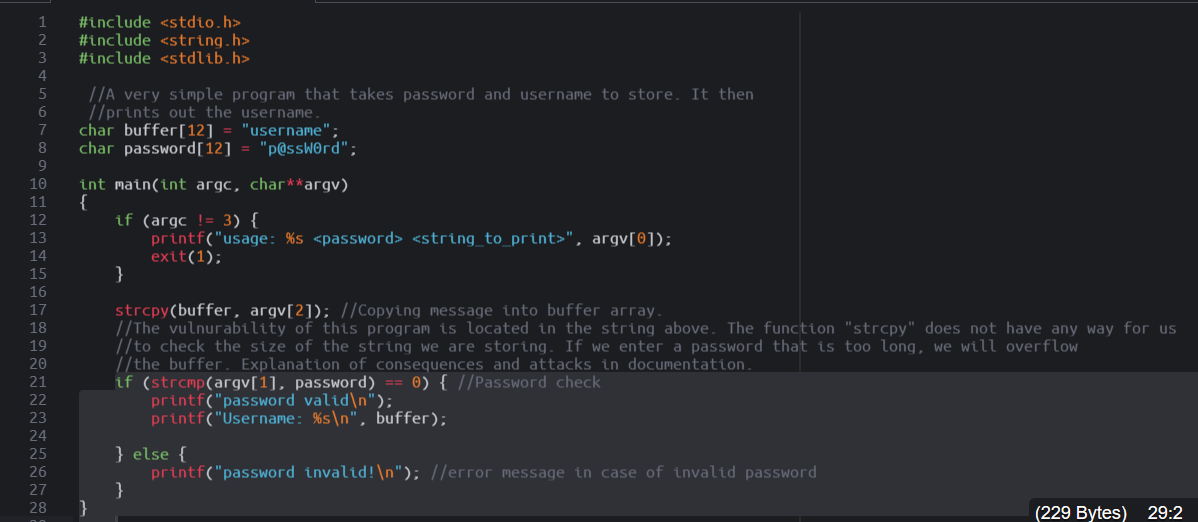
Since I was doing this on the compiler, the screenshot below is the closest thing I had to an app screen. It shows that the code is working as expected:



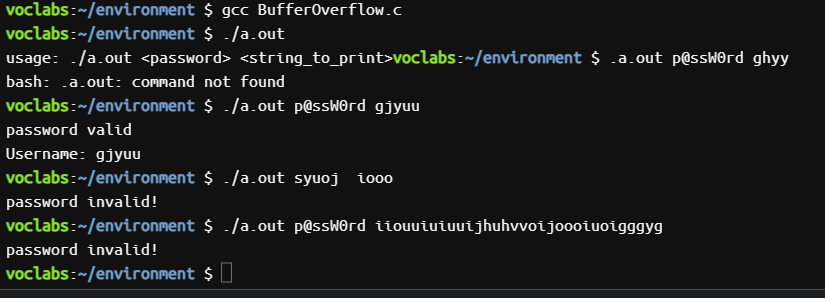
Analysis of the Vulnerability:

Buffer Copy without Checking Size of Input, sometimes referred to as Classic Buffer Overflow, is a vulnerability in which the buffer is given more data than it can take, resulting in the data “spilling” outside its boundaries. In this case, the function “strcpy” used in the original code, like many functions in the C programming language, does not have a length embedded into it for how much a user is allowed to copy and store in a buffer. Because of this, when a message is copied, “strcpy” will continue reading and copying a message until a zero or a null character is reached. If the message copied contains more characters than indicated in the buffer size, an overflow will occur, causing data to “spill” out of the array. Despite us having used a function that compares the two strings to make sure the same password is entered, it is not enough.

Vulnerable Code:



Vulnerable App Result:

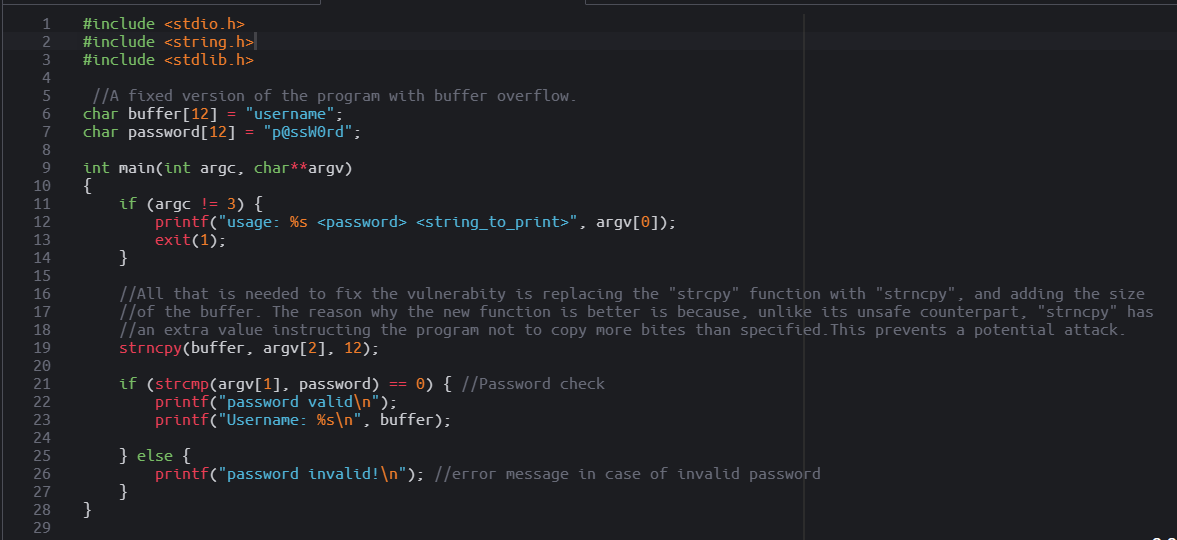


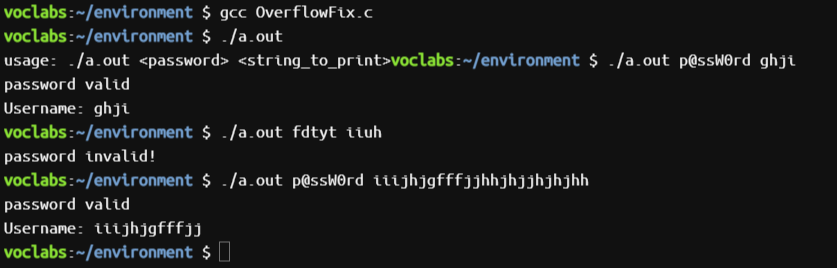
What we see happening above is a classic example of buffer overflow. To test the app, we first enter the correct password, along with a username that has less than 12 characters (our specified buffer size). The program works as expected, validating the password. We then enter an invalid password, and, again, a username less than 12 characters long. The program works as expected, displaying an error message. However, once we enter a username much larger than the number of characters specified, an error message is displayed even if a valid password has been entered. This means that a data “spill” has occurred, and the lost data can now be used by hackers for malicious purposes.

Mitigation:

Perhaps the best and easiest way to mitigate buffer overflow is to replace the “strcpy” function with the much safer “strncpy”. “strncpy” is safer because it has within itself an additional value, which is the number of bytes that can be copied into the buffer. This way, the string does not just copy an infinite number of characters, but stops after reaching the indicated buffer size, avoiding overflow.

Repaired Code:





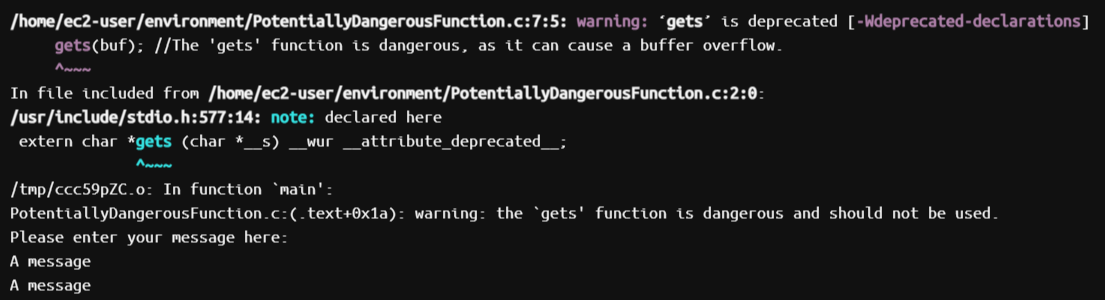
We can now see that even with a very long username, the password is still recognized as valid. But this time, only the first 12 of its characters are printed. From this, we can conclude that the vulnerability has been fixed.

Example 2 - [CWE-676: Use of potentially dangerous function]

Overview:

For this vulnerability, I created a short app that accepts and prints a message. It is once again written in C. Honestly, this second vulnerability is very connected to the first one, so the working principle is behind my app, as well as the vulnerability it leads to, is almost the same. Many times, dangerous functions can lead to many problems, such as the aforementioned buffer overflow. This could result in data being leaked to hackers.

Here is my app screen (compiler):

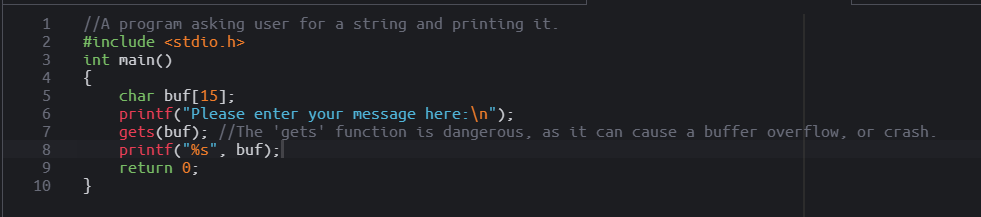


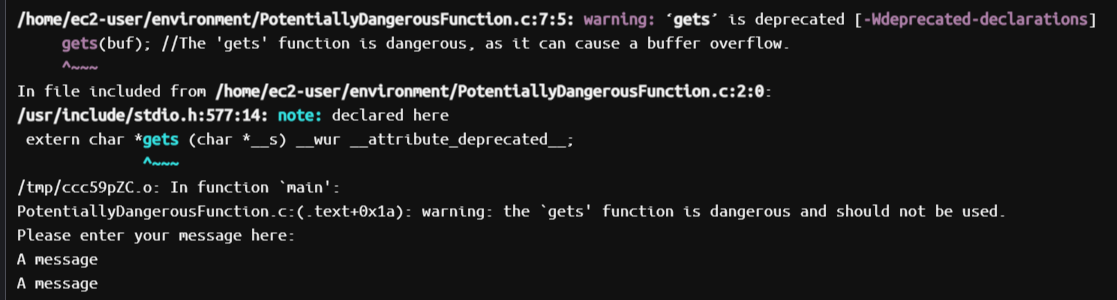
Despite my message (“A message”) being printed two times,the compiler does warn me that the function is dangerous and should not be used.

Analysis of the Vulnerability:

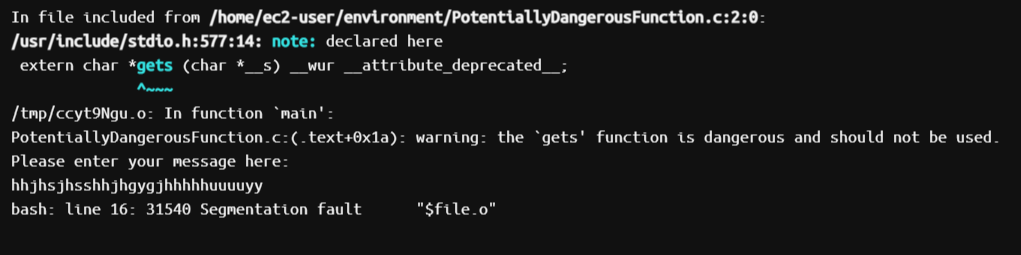
“Use of potentially dangerous function” is the process of using a function that, when utilized incorrectly, could open the door to vulnerabilities. The most common vulnerability introduced by such functions is buffer overflow, already described in great detail in the section above. However, they can also bring more serious problems, such as crashing the entire program, and/or displaying errors. In the case of my code, the app uses the “gets” function to retrieve and display a message the user has entered. Even if the user does not attempt to overflow it, it still displays a warning about the dangers of the “gets” function, erroneously displaying the message twice.

Vulnerable Code:





However, if we actually overflow the application by entering an extremely long message, we will get an even worse result:

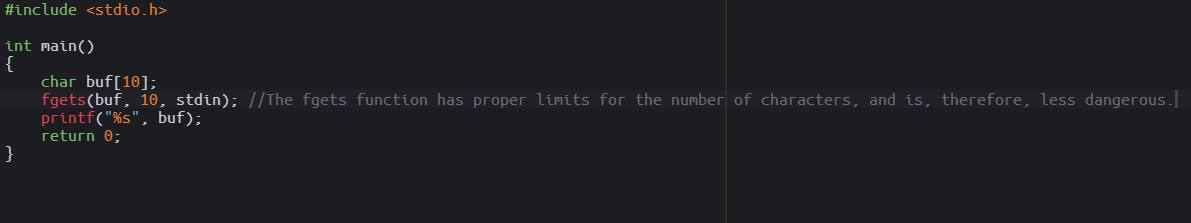


This gives us a segmentation fault error, which means that the user has attempted to access memory they should not have access to. The program prevents this from occurring by ending itself, but our excessively long message has still managed to break it.

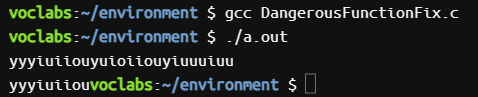
Mitigation:

To mitigate this problem, one is better off replacing the unsafe function with a safer counterpart. In this case, “gets()” has been replaced with “fgets()”, as the latter can restrict the number of characters entered by holding a value of the number of bytes. This way, the program will only print the first few characters of the message, up to the specified buffer size.

Repaired Code:



Now that a new function is implemented, the vulnerability is finally dealt with, and only the first ten characters of the message are printed.



References:

[CWE - CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') (4.10) (mitre.org)](https://cwe.mitre.org/data/definitions/120.html)

[CWE - CWE-676: Use of Potentially Dangerous Function (4.10) (mitre.org)](https://cwe.mitre.org/data/definitions/676.html)