**Fixing C code with Vulnerabilities**

Benjamin M. Brandhorst

University of Maryland Global Campus

SDEV 425 7980 Mitigating Software Vulnerabilities

Doctor Ronald McFarland

April 26th, 2020

**Part 1: Sample C code Application**

One of the first changes I made to the application was adding fflush to flush the buffer so that when getchar was used, the buffer is ready to accept user input rather than using the ‘\n’ stored in the buffer from the previous menu selection. This can be seen in Figure 1.

A screenshot of a cell phone

Description automatically generated

**Figure 1 – Demonstrating changes to Main**

They second change made to the application was creating a confirmExit function to pause the application while waiting for the user to input a ‘Y’ or ‘y’ rather than exiting automatically. Figure 2 shows the code used for this.

A picture containing laptop, table, computer

Description automatically generated

**Figure 2 – confirmExit function**

With these changes made, you can see in Figure 3 that the program operates in the same manner as the demonstration in the assignment instructions.

A screenshot of text

Description automatically generated

**Figure 3 – Program output after changes**

**Part 2: C Code Rules and Recommendations**

**STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator.**

Because we want to avoid the possibility of copying data to a buffer that isn’t large enough to hold it, which would cause a buffer overflow, I have made a simple modification to the application. This change ensures that there will be enough room for both the data and the null termination character. Figure 4 shows the addition of N-1 to the loop.

A screenshot of a cell phone

Description automatically generated

**Figure 4 – Changes to loop within fillPassword function**

**MSC24-C. Do not use deprecated or obsolescent functions.**

Our application includes the **printf** function which according to MSC24-C is an obsolescent function. I tried to implement the **printf\_s** function but using MACOS, I had issues with my compiler recognizing it as a valid function. Instead, I decided to change the application by creating **char \*text** to hold strings and using **asprintf.** This can be seen in Figure 5 and Figure 6.

A picture containing drawing

Description automatically generated

**Figure 5 - \*text variable**

**A picture containing meter

Description automatically generated**

**Figure 6 – First implementation of asprintf function**

Using **asprintf** is safer than **printf** because it dynamically allocates the string which the output is sent to. This means that the string will never overflow.

**FIO34-C. Distinguish between characters read from a file and EOF or WEOF.**

FIO34-C tells us that the EOF macro is a negative value used to indicate that the end of a file has been reached and no data remains. It is an in-band error indicator which are problematic to work with. With EOF returning a negative value, it shouldn’t match any unsigned character values. However, if the implementation has int and char with the same width, a character reading function could read and return a valid character with the same bit pattern as EOF. This leaves the application vulnerable to attack if someone were to enter a value that looked like EOF into the file or data stream. I’ve modified the application by changing the **cont** variable from a char to an int and adding a **feof()** function to test for end of file and a **ferror()** function to test for errors. The results can be seen in Figure 7.

A screenshot of a cell phone

Description automatically generated

**Figure 7 – Compliant EOF Solution**

**MSC17-C. Finish every set of statements associated with a case label with a break statement**

MSC17-C tells us that if a break statement is omitted from a selection in a switch statement, control flow will fall through to the next case in the switch statement block. However, because C does not require a break statement, no errors will be thrown if it is somehow omitted from the code. It may, however, produce unexpected control flow. Our application is missing a break statement from the ‘F’ case in our switch statement. It has been added to bring things into compliance as can be seen in Figure 8.

A screenshot of a cell phone

Description automatically generated

**Figure 8 – Break statement insertion**

**DCL20-C. Explicitly specify void when a function accepts no arguments.**

DCL20-C tells us that using function declarators with empty parentheses is an obsolescent feature. This is because when a function is defined with no arguments, the compiler does not check whether the function is called with parameters. This can lead to unwanted errors and possibly leave the application open to exploitation if unwanted arguments are allowed to pass during a functions call. Our application did not need any changes to make existing functions specify void. However, I did write an additional function to get the program to perform as it was expected to. Figure 9 shows the additional function including the void specification in its definition.

A screenshot of a cell phone

Description automatically generated

**Figure 9 – Specifying void in a newly defined function**

**Final Application Output Demonstration.**

Now that these changes have been made to the application code, you can see in Figure 10 that the program continues to perform as expected.

A screenshot of a cell phone

Description automatically generated

**Figure 10 – Final Application Output**

References

DCL20-C. Explicitly specify void when a function accepts no arguments - Sei cert C coding standard - Confluence. (n.d.). Retrieved from https://wiki.sei.cmu.edu/confluence/display/c/DCL20-C.+Explicitly+specify+void+when+a+function+accepts+no+arguments

FIO34-C. Distinguish between characters read from a file and eof or weof - Sei cert C coding standard - Confluence. (n.d.). Retrieved from https://wiki.sei.cmu.edu/confluence/display/c/FIO34-C.+Distinguish+between+characters+read+from+a+file+and+EOF+or+WEOF

The GNU C programming tutorial. (n.d.). Retrieved from https://crasseux.com/books/ctutorial/asprintf.html

MEM30-C. Do not access freed memory - Sei cert C coding standard - Confluence. (n.d.). Retrieved from https://wiki.sei.cmu.edu/confluence/display/c/MEM30-C.+Do+not+access+freed+memory

MSC17-C. Finish every set of statements associated with a case label with a break statement - Sei cert C coding standard - Confluence. (n.d.). Retrieved from https://wiki.sei.cmu.edu/confluence/display/c/MSC17-C.+Finish+every+set+of+statements+associated+with+a+case+label+with+a+break+statement

Necco, D. (2019, December 2). STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator - Sei cert C coding standard - Confluence. Retrieved from https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator