Assignment 5

Mercy Ikem

8859960M

PROG8291

Prof. Sajjad Shahid

28 July 2024

Table of Contents

[Assignment 5 2](#_Toc172892776)

[Vulnerability 1 2](#_Toc172892777)

[A screenshot containing the vulnerable code 2](#_Toc172892778)

[which vulnerability class or the kingdom the finding represents? 3](#_Toc172892779)

[a brief discussion on how an attacker would exploit the finding 3](#_Toc172892780)

[a brief discussion on how you would recommend fixing the finding 3](#_Toc172892781)

[Vulnerability 2 4](#_Toc172892782)

[A screenshot containing the vulnerable code 4](#_Toc172892783)

[which vulnerability class or the kingdom the finding represents? 5](#_Toc172892784)

[a brief discussion on how an attacker would exploit the finding 5](#_Toc172892785)

[a brief discussion on how you would recommend fixing the finding 5](#_Toc172892786)

[Vulnerability 3 5](#_Toc172892787)

[A screenshot containing the vulnerable code 5](#_Toc172892788)

[which vulnerability class or the kingdom the finding represents? 6](#_Toc172892789)

[a brief discussion on how an attacker would exploit the finding 6](#_Toc172892790)

[a brief discussion on how you would recommend fixing the finding 6](#_Toc172892791)

[Vulnerability 4 7](#_Toc172892792)

[A screenshot containing the vulnerable code 7](#_Toc172892793)

[which vulnerability class or the kingdom the finding represents? 7](#_Toc172892794)

[a brief discussion on how an attacker would exploit the finding 8](#_Toc172892795)

[a brief discussion on how you would recommend fixing the finding 8](#_Toc172892796)

[Vulnerability 5 8](#_Toc172892797)

[A screenshot containing the vulnerable code 8](#_Toc172892798)

[which vulnerability class or the kingdom the finding represents? 9](#_Toc172892799)

[a brief discussion on how an attacker would exploit the finding 9](#_Toc172892800)

[a brief discussion on how you would recommend fixing the finding 9](#_Toc172892801)

# Assignment 5

# Vulnerability 1

## A screenshot containing the vulnerable code

A screenshot of a computer

Description automatically generated

## which vulnerability class or the kingdom the finding represents?

The vulnerability in question is CWE 788, which belongs to the “Access of Memory location after End of Buffer class of vulnerabilities. This basically means that when using fgets() to read data from a file, there’s chance that memory beyond the allocated buffer will be accessed, which could lead to security issues.

## a brief discussion on how an attacker would exploit the finding

Consider an exploit in which the attacker edits a file by adding data larger than the buffer size specified by fgets(). For instance, if fget() attempts to read a line that is longer than the buffer can hold, it might write over the buffer’s boundaries. This may result in the overwriting of neighboring memory regions, which may have unexpected repercussions such as data corruption, program crashes, or, in the worst-case scenario, the execution of malicious code if the overwritten memory contains executable instructions.

## a brief discussion on how you would recommend fixing the finding

The way of fixing this vulnerability is by making sure the input is appropriately small before copying it into the buffer. One way to do this is to make sure the input is the right length before copying it. To achieve this, measuring the input’s length and use functions like fget() and strlen() to compare it to the buffer size before doing the copy operation. To further minimize risks, limit the number of characters copied into buffer by using safer alternatives like fgets() in addition to strncpy(). This technique reduces the possibility of buffer overflow vulnerabilities.

# Vulnerability 2

## A screenshot containing the vulnerable code

A screenshot of a computer

Description automatically generated

## which vulnerability class or the kingdom the finding represents?

CWE 398, which is categorised as a “Indicator of Poor Code Quality,” is the subject of this conclusion, instead of clearly pointing out a security flaw, identifies areas where the code could be improved to make it easier to read and maintain.

## a brief discussion on how an attacker would exploit the finding

analyzing possible exploits for this discovery by an attacker reveals that there isn’t clear threat currently. However, the real risk lies in the long-term consequences of poor code quality. Although the issue the cppcheck found isn’t currently a security breach, difficult-to-read and maintain code may eventually introduce vulnerabilities. For instance, utilizing the variable “ret” in a big codebase without enough scoping could lead to errors or unexpected behaviors the hostile actors could take advantage of.

## a brief discussion on how you would recommend fixing the finding

it is advisable that the variable “ret” be restricted to the bare minimum needed to fix the cppcheck warning and enhance the code’s quality. This reduces the likelihood that issues will arise and makes the code easier to read and maintain. Declaring “ret” at the level of the specific code block in which it is used is more practical than doing so at a higher level. By guaranteeing that ret is only accessible when required, this improves code readability and lowers the possibility of future vulnerabilities.

# Vulnerability 3

## A screenshot containing the vulnerable code

A screenshot of a computer

Description automatically generated

## which vulnerability class or the kingdom the finding represents?

The outcome is in line with CWE-398, a coded “Indicator of Poor Code Quality.” This classification indicates areas in which the codebase’s readability and maintainability need to be improved. The warning in the provided code specifically calls attention to the authenticate() function’s parameter “pass” and recommends that it be defined as a pointer to const in order to enhance the code’s quality.

## a brief discussion on how an attacker would exploit the finding

analyzing potential applications for this discovery reveals that the emphasis should be on upholding best practices and maintaining code maintainability rather than immediately fixing a security flaw. The data in the ‘pass’ buffer may be altered by the authenticate() function if ‘pass’ is declared as a non-const reference (char\*). However, if that function doesn’t change “pass”. It could lead to misunderstanding or abuse, while this might not directly result in an attack, similar coding techniques could lead to misunderstandings or unintentional changes that introduce additional software vulnerabilities.

## a brief discussion on how you would recommend fixing the finding

to fix this problem and make the code better, it is advised to declare the ‘pass’ parameter as a pointer to const. by doing this, it is made obvious that the authenticate() method does not intend to modify the contents of the ‘pass’ buffer. This clarification enhances the clarity of the function purpose and lessens the chance of unintentional changes. Informing users that modifying the functions’s “pass” buffer is not advised is also a smart idea. Finally, using the least restrictive type qualifiers to accurately represent the code’s intent promotes greater readability, maintainability, and strength in the codebase.

# Vulnerability 4

## A screenshot containing the vulnerable code

A screenshot of a computer

Description automatically generated

## which vulnerability class or the kingdom the finding represents?

The discovery is relevant to CWE-682, a case with a classification of “incorrect calculation”. This vulnerability class describes circumstances where the code’s calculations do not yield the intended or anticipated outcomes, potentially leading to unexpected behaviour or improper memory allocation. The provided code snippet raises a warning due to the use of sizeof() on a constant value, which can lead to unpredictable program behaviour.

## a brief discussion on how an attacker would exploit the finding

there may be risks even if this issue doesn’t instantly result in a security breach. For instance, if a programmer assumes that sizeof (1024) returns the size of the number 1024, they may encounter unexpected outcomes or memory corruption. If the intention was to use memset() to clear a buffer with a size of 1024 bytes, then sizeof(1021) won’t produce the expected result. This can be exploited by an attacker to cause the software to behave strangely.

## a brief discussion on how you would recommend fixing the finding

using the real buffer size is crucial to solving this issue and ensuring optimal functionality instead of relying on constant numbers. The memset() function needs to be changed to use the buffer size correctly instead of sizeof (1021). For example, change the code as follows to make sure the entire buffer that ‘path’ points to is cleared: memeset(path, o, path\_size); a hardcoded value is replaced with sizeof(path), which simplifies maintenance and reduces error-proness. Emphasise the use of sizeof() when working with variables or arrays to make sure that the correct size is computed dynamically at runtime, enhancing the stability of the codebase.

# Vulnerability 5

## A screenshot containing the vulnerable code

A screenshot of a computer

Description automatically generated

## which vulnerability class or the kingdom the finding represents?

The results in line with CWE-563, which is categorized as a “Unused Variable.” This category marks those situations in which variables are declared but not used in the code. The warning explicitly notes that the variable “p” is declared in the given snippet but is not referenced anywhere else in the code.

## a brief discussion on how an attacker would exploit the finding

although unused variables might not instantly lead to security holes, they could be indicators of more significant issues with the maintenance and caliber of the code. Unused variables are typically not directly exploited by attackers. However, security vulnerabilities may quietly emerge if these issues are disregarded. Vulnerabilities or weaknesses maybe overlooked, for instance, if code review and maintenance procedure are disregarded due to unutilized variable. In addition to making code harder to understand and maintain, unused variables that clog codebases may even unintentionally introduce security vulnerabilities elsewhere in the code.

## a brief discussion on how you would recommend fixing the finding

The following are two recommended approaches to address this warning and enhance the code quality. Initially, the variables “p” should be completely removed from the code if it is not needed to clean up and simplify the codebase. On the other hand, it can help prevent misunderstandings if notes are made clear that ‘p’ is intended to be used for debugging or future use. Regular code reviews and static analysis tools can also be used to identify and remove these unnecessary variables. This makes the code more maintainable and less likely for bugs or vulnerabilities to slowly creep in. codebase management produces a more stable and secure software environment by prioritizing cleanliness and clarity.