Patrick Walsh

Graded Assignment: Homework 3

University of Maryland Global Campus

SDEV 325 – Detecting Software Vulnerabilities

# Executive Summary

Demonstrating insecure interaction between components for CWE-676: Use of Potentially Dangerous Function and CWE-494: Download of Code Without Integrity Check. This document includes detailed explanation and code snippets taken from functional code showing each of these vulnerabilities in practice as well as how to mitigate them.

In each example, the security vulnerabilities are dealt with in the document and in the provided code, and the mitigation technique is explained.

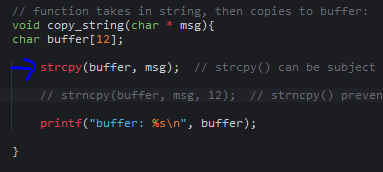
# Example 1 – CWE-676: Use of Potentially Dangerous Function

## Overview

This example highlights the use of a potentially dangerous function in the C language (CWE, 2021a). In this example, the program uses a function which takes in a string and attempts to copy the strong over to allow for data manipulation. The function makes use of the strcpy() function which is vulnerable to exploits if used incorrectly (CWE, 2021a). The function can cause buffer overflows in the program memory, causing the program to crash or other security issues. To mitigate this problem, developers should use the more secure strncpy() function instead.

## Analysis of the Vulnerability

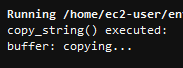
The program function copy\_string() takes in a string parameter, then copies that string to a buffer using the strcpy() function. The strcpy() functions has two parameters, the destination string and source string. In this case, the msg parameter is the source string and buffer parameter is the destination string. The buffer is expecting a string length maximum of 12 characters:



When the copy\_string() function is called, there are no issues:



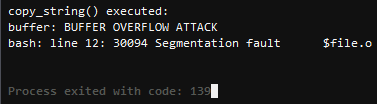
Output:



The vulnerability lies in when the buffer string length is violated and a string of more than 12 characters is inserted into the function:



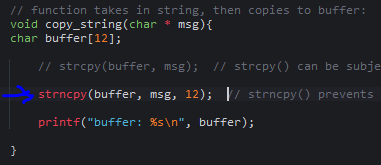
Output:



The buffer is overloaded and the program crashes unexpectedly.

## Mitigation

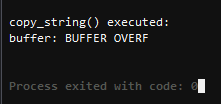
To avoid this vulnerability, developers should use the strncpy() function instead of the strcpy() function. The strncpy() function takes in the same two parameters as before, but it also takes in one additional parameter in the form of a string length limit. This ensures that the string copied over to the buffer is of a certain size. Here, the limit is 12, just as the buffer is limited to size 12:



Now when the function is called and a string longer than the expected limit is inserted, the program does not crash:



Output:



The entire string is not copied over since it is longer than the allowed limit, but the program avoided a buffer overflow and the program terminates without errors.

# Example 2 – CWE-494: Download of Code Without Integrity Check

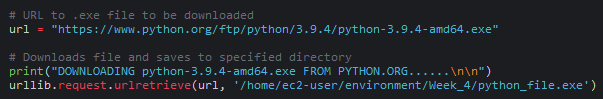
## Overview

This example shows a common software vulnerability of downloading and running executable files without a proper integrity check. This is a potential security threat because a user may download a file without verifying the origin of the file and integrity of its contents (CWE, 2021b). Some files may come from malicious links or may be corrupted in transit, leaving the recipient open to exploits. To avoid this, users should verify the integrity of the file before running or opening it.

This example uses Python to download a file from the internet and an MD5 hash value to verify that the contents of the file have not been altered in transit.

## Analysis of the Vulnerability

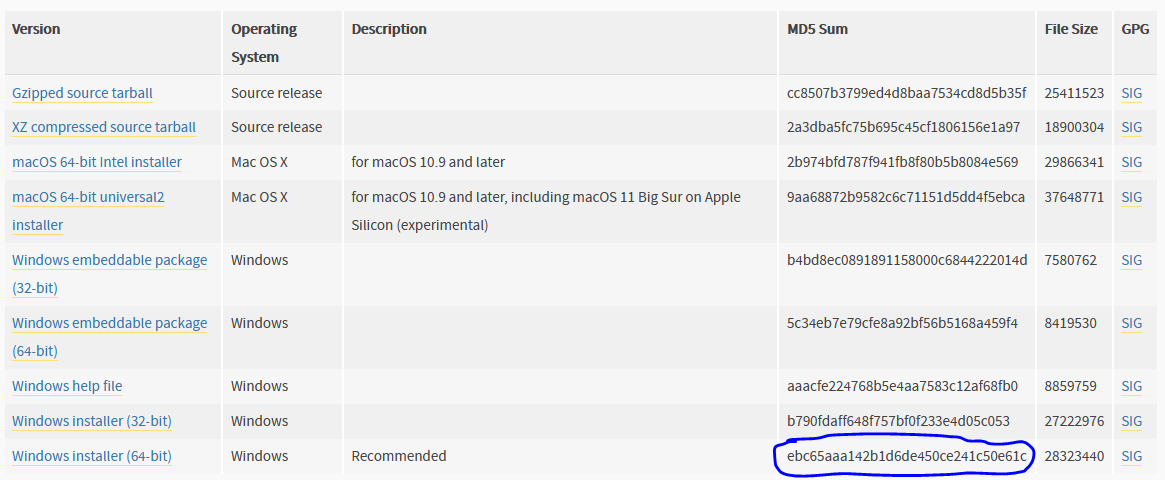
The code snippet below downloads the executable file from python.org:



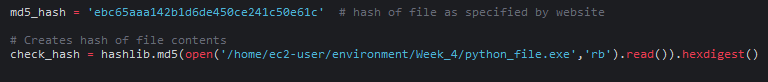
However, this code does not verify the integrity of the file, so it is possible that the file could be altered or otherwise corrupted in transit. If a user were to run this file, it is uncertain if the file could pose a security risk.

## Mitigation

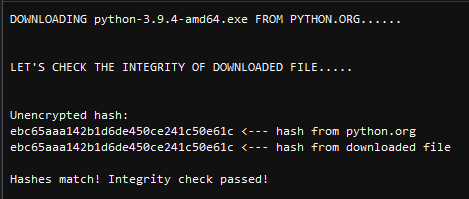
To verify file integrity, we can look at the file hash value from python.org:



We will compare this hash value with the hash value generated from the file contents:

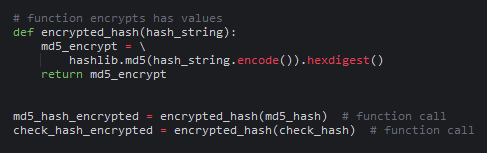


The ‘check\_hash’ call creates a hash of the file contents. We then compare this hash to the hash value from python.org:

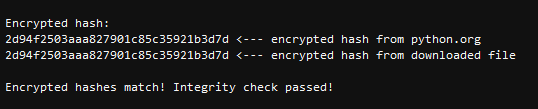


Since the hash values match, we can be confident that the file has not been changed and thus our integrity check passed.

If we want to add an extra layer of security and not expose the value of the hash, we could first encrypt the hash values and then check the result:



The encrypted hash results are seen below:



# References

Common Weakness Enumeration (CWE). (2021a, March 15). CWE - CWE-494: Download of Code Without Integrity Check (4.4). 2020 CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/data/definitions/494.html

Common Weakness Enumeration (CWE). (2021b, March 15). CWE - CWE-676: Use of Potentially Dangerous Function (4.4). 2020 CWE Top 25 Most Dangerous Software Weaknesses. https://cwe.mitre.org/data/definitions/676.html