Graded Assignment: Homework 4

Mark Kardash

University of Maryland Global Campus

SDEV 325 – Detecting Software Vulnerabilities

Executive Summary:

Even though I initially had immense difficulties completing this assignment, I finally managed to create programs for two vulnerabilities on the list, CWE-798: Use of Hard-Coded Credentials, and CWE-732: Incorrect Permission for critical function. The first one happens when credentials are written directly into the code, presenting the danger that the attacker will access passwords and usernames by simply hacking into the source code. The second involves giving permission of performing certain actions to the people who should not have permission to perform them. For the second vulnerability, I have used the “umask” function to create permissions, but have only managed to use it from the compiler rather than directly from the source code.

Example 1 – [CWE-798: Use of Hard-Coded Credentials]

Overview:

For this vulnerability, I created a simple security question verifier, that could be used when someone is logging into their account. It asks the user what their favorite fruit is, and has the answer, “Apples”, written as a string of code. It is written in Java.

Code working as expected:

Text

Description automatically generated

Analysis of the vulnerability:

Use of Hard-Coded Credentials happens when sensitive information is included in the source code for an application. In this case, the answer to the security question is written into the code as a string. A breach, such as a brute-force attack is very possible under these conditions. Moreover, an attacker can simply guess the answer through password-guessing (or answer-guessing) attacks. This is especially true in a situation like this, where the attacker will simply test the most common fruit names to break into the app.

Vulnerable Code:

Text

Description automatically generated

Vulnerable App Result:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

This is a very small-scale demonstration of a guessing attack. After a few guesses, the hacker will finally manage to login.

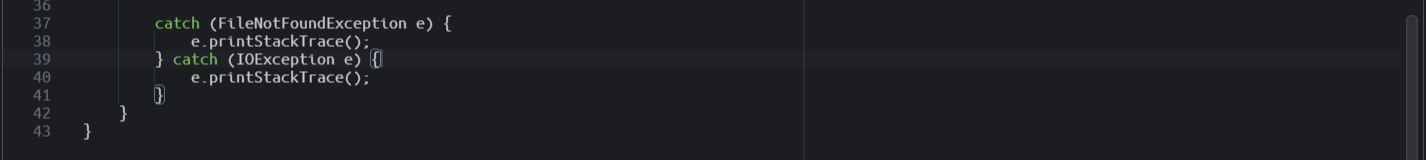
Mitigation:

One way to mitigate this vulnerability is to store the credentials or any sensitive data in external files. The guessing attacks could still happen, but the code would be much safer, since the attacker would not have direct access to data through the source code.

Repaired Code:

Text

Description automatically generated



The answer is now stored in the file “SecurityQuestionAnswer.txt” located in the same project, with the program using “while” and “if” loops to compare the user input with the file contents and print out the corresponding message.

Example 2 – [CWE-732: Incorrect Permission Assignment for Critical Resource]

Overview:

For this example, I created a program that stores a person’s credit card number in a newly created file. It then uses the umask function to set the permission level to what is required. A permission set to 0 would make it vulnerable to attacks. This program is written in C.

Text

Description automatically generated

Analysis of the Vulnerability:

Incorrect Permission Assignment for Critical Resource occurs when certain actors are allowed access to parts of application that they should not have access to. Here, this is done by using the “umask” function. For some reason, I could not use umask directly in the code, so I manipulated it from the compiler instead. Initially, the umask value is set to 0, which means that practically anyone has the permission to read and write inside the code. This means that just about any kind of attack can be perpetrated, from injection of malicious code (or viruses), to direct code modification, to sensitive data leaks.

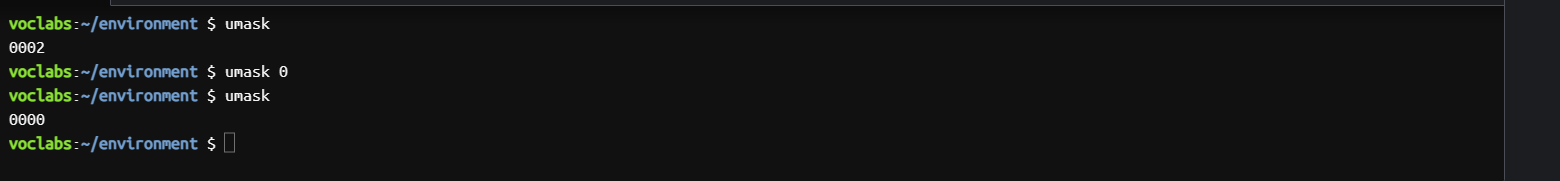
Vulnerable Code:

Text

Description automatically generated

Graphical user interface, text

Description automatically generated



I first called umask in the compiler, to see its initial value, which was “0002”. I then set umask to 0, and verified it by calling it again. Now that it is 0, anyone can modify the code. Unfortunately, due to the nature of this vulnerability, an attack cannot be explicitly demonstrated, and it remains only to be said that the code can be directly modified.

Mitigation:

Mitigation of this issue is quite easy, as all one has to do is change the umask value to “0077”, which will give reading and writing permission only to the user of the file. This, obviously, makes it safer, with third party access restricted.

Repaired Code:

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Once again, we check the initial umask value, and modify it to the one needed. Our program is now safe.

References:

[CWE - CWE-732: Incorrect Permission Assignment for Critical Resource (4.11) (mitre.org)](https://cwe.mitre.org/data/definitions/732.html)

[CWE - CWE-798: Use of Hard-coded Credentials (4.11) (mitre.org)](https://cwe.mitre.org/data/definitions/798.html)