Missing Encryption and User of Hard Coded Credentials Vulnerability Report

Hiren Shah

4/12/2021

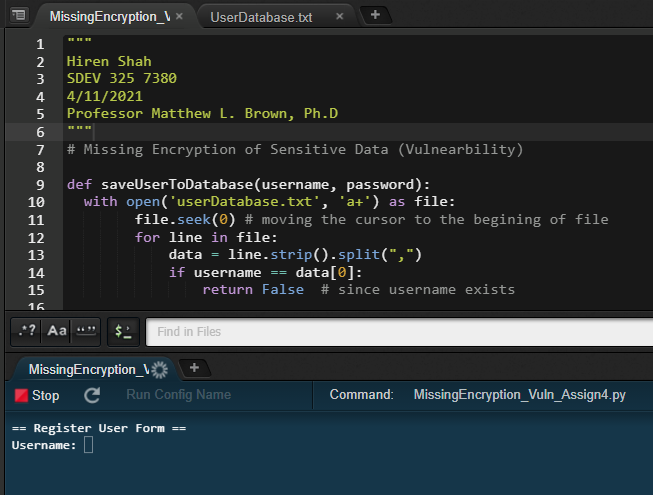
# Executive Summary

This topic demonstrates the topic of “Demonstrating Porous Defenses.” “Out of the CWE/SANS Top 25 types of security vulnerabilities, 11 involve porous defenses.” (Handova, D, 2019). Defensive techniques such as encryption, authentication and authorisation are essential to application security, however when they are misued or implemented incorrectly, they become application vulnerabilities. The unique vulnerabilities selected were the CWE-311 : Missing Encryption for Sensitive Data, and CWE-798 : Use of Hard-coded Credentials. Both the vulnerabilities were mitigated.

# Example 1 – **CWE-311: Missing Encryption of Sensitive Data**

## Overview

CWE-311 is when a software program does not encrypt sensitive or critical information before storage of transmission. In my example I have a register user form which saves the username and password to a database.txt file. The application is writte in Python.



## Analysis of the Vulnerability

In my example the user enters there name and password, which stores that information in the UserDatabase.txt file but does not encrypt the password. If a malicious user wanted access to the password file they could potentially use a ‘CWE-22 : Improper Limitiation of a Pathname’ or another vulnerability to gain access to the passwords.

Vulnerable Code:

*def saveUserToDatabase(username, password):*

*with open('userDatabase.txt', 'a+') as file:*

*file.seek(0) # moving the cursor to the begining of file*

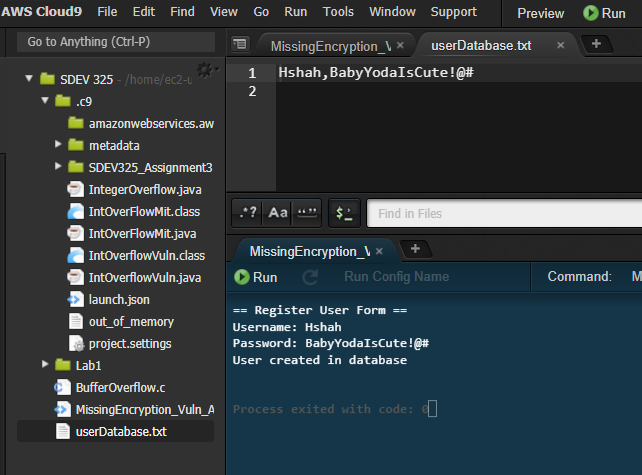
*for line in file:*

*data = line.strip().split(",")*

*if username == data[0]:*

*return False # since username exists*

*file.write(username + "," + password + "\r")*



## Mitigation

For the mitigation I encrypted the given password by using sha256\_crypt. This aglorithm generates an almost unique 256-hash. Even if a malicious user manages to view the passwords through some other vulnerability, they would be encrypted and they wouldn’t be able to use it unethically. In the screenshot below, I had to use repl.it because the aws environment would not let me install the Sha256 encrypt.

Code:

*from passlib.hash import sha256\_crypt*

*def saveUserToDatabase(username, password):*

*with open('userDatabaseEncrypted.txt', 'a+') as file:*

*file.seek(0) # moving the cursor to the begining of file*

*for line in file:*

*data = line.strip().split(",")*

*if username == data[0]:*

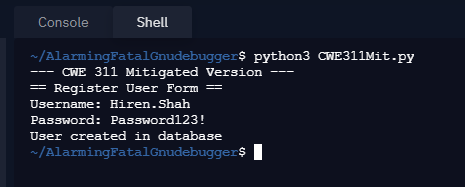
*return False # since username exists*

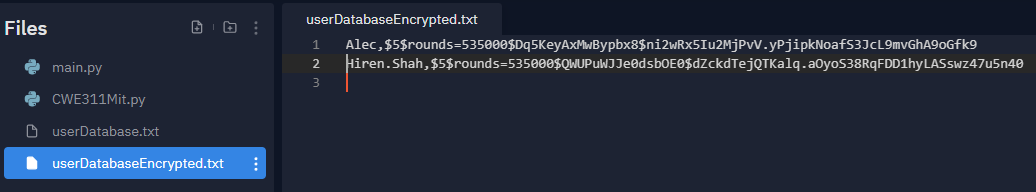
*hash\_pass = sha256\_crypt.hash(password)*

*# write to end of file since username is unique*

*file.write(username + "," + hash\_pass + "\r")*

*return True # success*

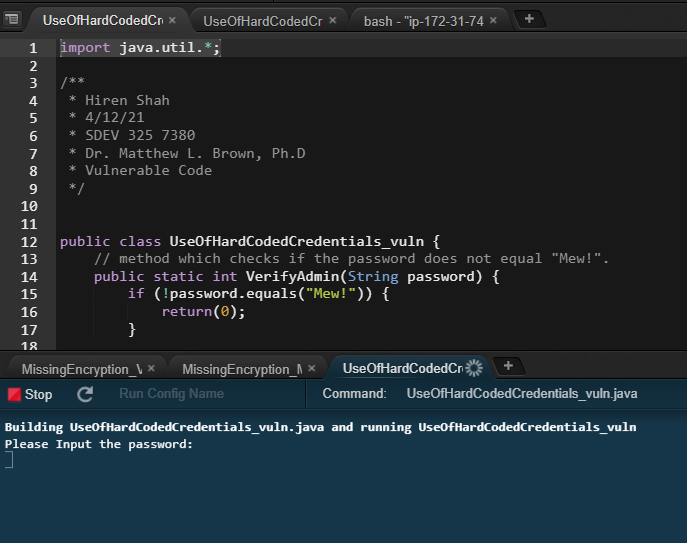




# Example 2 – **CWE 798:** **Use of Hard-coded Credentials**

## Overview

In my example the user inputs a password and the program does a password verification check. It is vulnerable because the real password is hardcoded in the program, hence an attacker can possibly disassemble it’s bytecode to reveal sensitive information. The program is written in Java.



## Analysis of the Vulnerability

In the vulnerable code, the password is hardcoded into the method VerifyAdmin. This makes it so the attacker can decode the bytcode to reveal password information.

**Code**

// method which checks if the password does not equal "Mew!".

public static int VerifyAdmin(String password) {

if (!password.equals("Mew!")) {

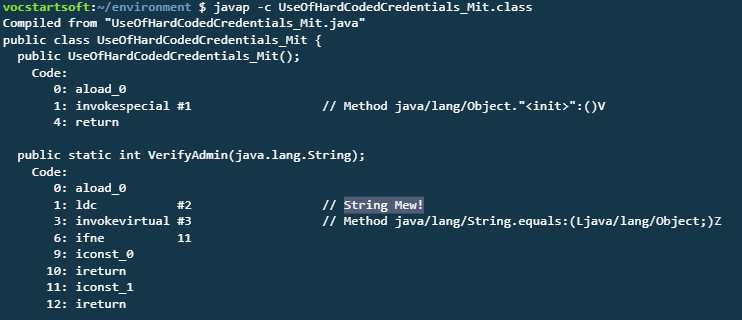
return(0);

}

//Diagnostic Mode

return(1);

}

Showing screenshot of vulnerable code

## Mitigation

In the mitigation example, we use the method CheckingPasswordService() to simulate how the server-side verifies if the password is correct or not. Since it is doing it in the server side, the real password is never revealed on the client-side, therefore it is secured. First we send a request to the remote server, then the remote-side is doing the password verification, then it sends the result back to the client-side, then the client-side decides what to do next based on the given information.

**Code**

public static int checkingPasswordService(String password) {

System.out.println("Send password to server");

System.out.println("Server side is performing password verification");

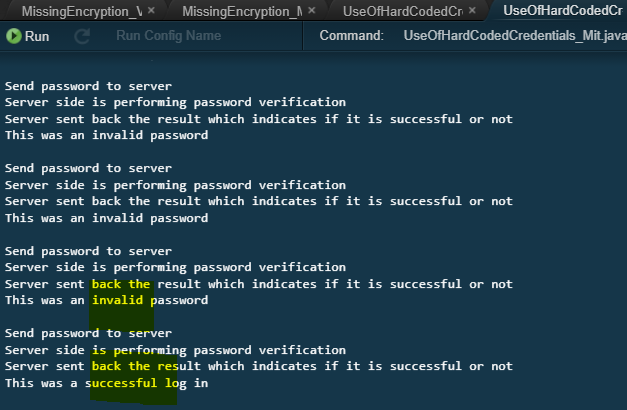
System.out.println("Server sent back the result which indicates if it is successful or not");

Random rn = new Random();

int result = rn.nextInt(2);

return result;

}

Showing successful and unsuccessful log ins.

# References

Handova, D., 2019.

<https://sme.asia/what-are-the-different-types-of-security-vulnerabilities/#:~:text=Porous%20defense%20vulnerabilities&text=Three%20of%20these%20vulnerabilities%20point,Missing%20Authorisation%2C%20and%20Missing%20Encryption>