5

Graded Assignment: Homework 5

University of Maryland Global Campus

SDEV 325 – Detecting Software Vulnerabilities

Kyra Samuel

# Executive Summary

Using AWS Cloud 9 to create two different unique demonstrations of Porous Defenses such as Improper Restriction of Excessive Authentication Attempts and Use of a One-Way Hash without a Salt.

# Example 1 – CWE-759: Use of a One-Way Hash without a Salt

## Overview

This python application allows you to pick between the two vulnerabilities presented. When selecting option **2: Use of a One-Way Hash without a Salt**, the user is prompted with a second menu. For simplicity, the database will be represented by an array called database.

Text

Description automatically generated

## Analysis of the Vulnerability

To view the vulnerability, select the option for “hack” first before salting the passwords. The program will display a scenario of your company getting hacked and what information the hacker stole.

A picture containing text, screenshot, screen

Description automatically generated

The reason why salted passwords are the better use for storing sensitive data is because hashing data creates duplicates. Notice in the picture above, the passwords for [jbailey@outlook.com](mailto:jbailey@outlook.com) and [fbutler@gmail.com](mailto:fbutler@gmail.com) are the same exact password. As a hacker, I have no idea what the password is yet, but I do know that these two emails have the same password and now I know that these passwords are not salted. This means that if I can hack into [jbailey@outlook.com’s](mailto:jbailey@outlook.com’s) account then I can also hack into [fbutler@gmail.com’s](mailto:fbutler@gmail.com’s) account. Fortunately, when I put these passwords into crackstation.net, none of them were cracked.

## Mitigation

To solve this issue, I am creating a random string with a random length as the salt and adding this to the password. Select the option for “salt passwords”:

This option calls get\_salt(), which returns a random string to act as the salt and adds it to the hash of the passwords. The content of the get\_salt() method include:

def get\_salt():

salt = ''

rand\_range = random.randint(5, 15)

salt.join(random.choice(string.ascii\_letters + string.digits) for i in range(rand\_range))

return salt

By creating a random string every time, the method is called, this can provide a way for unique strings to get added onto the passwords. Also, you could create place the salt at any index of the password to make it harder to detect where the salt is placed. For simplicity, the get\_salt() method is called before the encode() method and added to the end password then hashed. This format should be called every time a user registers for an account:

Text

Description automatically generated

The output of “salt passwords” is displayed below demonstrating that there are no duplicates and the vulnerability explained above is resolved:

Text

Description automatically generated

# Example 2 – CWE-307: Improper Restriction of Excessive Authentication Attempts

## Overview

This python application allows you to pick between the two vulnerabilities presented. When selecting option **1: Improper Restriction of Excessive Authentication Attempts**, the user is prompted with a second menu. For simplicity, the database will be represented by an array called database.

## Analysis of the Vulnerability

Hackers can use Brute Force Attacks on websites to crack passwords and steal people’s data. Software that requires some form of authentication yet does not restrict the amount of authentication attempts is not secure. After choosing the vulnerability demo option 1, select option **1: unrestricted** and the following scenario is displayed.

Graphical user interface, text

Description automatically generated

This is evidently an issue because the hacker does not even have to know the password initially. Most brute force attacks are iterating over a dictionary of generated passwords and inputting those passwords into the login form. Some of these dictionaries can have over hundreds of passwords which means that if the website has an unrestricted amount of login attempts – the hacker’s chances of cracking user’s passwords are extremely high.

## Mitigation

By simply adding a limit of attempts to a login form will suffice for this vulnerability. There are a ton of examples of software that use this technique. Apple does this with their iPhones in particular, the user gets three attempts then a 1-minute lock is set on the phone, after 5 attempts – a 30-minute lock is set on the phone, etc. For my example, I allowed the user five chances to enter an incorrect password. By entering option **2: restrict,** the following scenario is displayed:

Text

Description automatically generated

When using a database, a good practice is to store the number of attempts and times of these attempts. On the fifth attempt – check if the time difference between the fifth and first attempt is less or greater than a custom time limit (i.e 1 hour) – if not, lock the account and store the location of computer trying to get access into the username’s account in the database, and IP address. This helps stop brute force attacks because it lessens the odds of the program finding a password match. The brute force program would have to figure out the password in 5 tries. The company can also block IP addresses that are stored during these attempts and look for duplicates.

# References