Prepared By: Alexander Marquis

Supervisor: Douglas Lundin

Broken Authentication Vulnerability Mitigation

Broken Authentication

Abstract

A broken authentication vulnerability was identified through the use of the password changing software on the company website. This vulnerability stems from the lack of requirements for strong passwords for accounts on the website. This leads to an increased attack surface as malicious individuals have more points of entry into the software. Bad passwords are easily cracked or guessed and this means that the privileges of the accounts with bad passwords can be gained through guessing. This vulnerability can be demonstrated by changing the password of a user to “password” an obviously poor password. In order to demonstrate the ease of mitigating such a problem a demonstration was setup using the HaveIBeenPwned API. This demonstration will ask a user to enter a new password as theirs has expired. It will then check this new password against all of the leaked passwords that HaveIBeenPwned’s API knows about and if it has never been leaked the user’s password will be updated. The user will have five chances to enter in a new password that has not already been leaked until they are logged out of their account. This simple level of password testing and limiting of the number attempts can massively increase security by stopping bots in their tracks and forcing users to pick stronger passwords.

Table of Contents

[**Abstract** 1](#_Toc68616656)

[Broken Authentication Vulnerability Detection 3](#_Toc68616657)

[Vulnerability Confirmation 3](#_Toc68616658)

[Impacts to Company 4](#_Toc68616660)

[Vulnerability Mitigation 4](#_Toc68616661)

[Demonstration of Vulnerability Mitigation 5](#_Toc68616662)

Verification of [Mitigation Code 6](#_Toc68616662)-7

[Conclusion 8](#_Toc68616661)

[**References** 9](#_Toc68616664)

Broken Authentication Vulnerability Detection

A broken authentication vulnerability has been detected on the company website. This vulnerability was noticed by the cyber team during the usage and testing of the password changing software for the company’s website. The broken authentication vulnerability stems from a lack of good password requirements and unlimited password change attempts. A user is currently able to enter in horrible passwords that can be broken easily by password crackers or that has already been leaked.

## Vulnerability Confirmation

Graphical user interface, text

Description automatically generatedThis vulnerability is able to be confirmed as real and present by a simple abuse case of the password changing software on the companies JavaVulnerableLab web page in the production environment in docker. In the case demonstrated below a user is able to change their password to the word “password” which is an awful password that no users should be allowed to have. This is because poorly constructed passwords can serve as entry points for malicious individuals to leak, destroy, or otherwise abuse company data or resources that they otherwise wouldn’t have access to.

Graphical user interface, text

Description automatically generated

# Impacts to Company

Allowing users to have bad passwords is a vulnerability of broken authentication. This kind of broken authentication can be damaging to the company as it provides an entry point for malicious individuals to have access to privileges and resources that they normally wouldn’t have. What the attackers do after gaining access to these privileges and resources is highly dependent upon the account attacked. It can be as small as a single user account having their data leaked or destroyed or it could be as bad as system wide leakage or damage if an admin account is the one to have its password cracked. Broken authentication is currently number seven on the OWASP 2021 Top 10 list. These vulnerabilities, while not as prevalent as they used to be, can be devastating in their effects depending on the level of access that the perpetrators get (*A07:2021 – identification* 2021)

# Vulnerability Mitigation

After receiving direct approval from project supervisor Douglas Lundin on November 12th, 2021 work proceeded to demonstrate that this vulnerability can be mitigated via the use of the HaveIBeenPwned API. This API can check a password against millions of passwords that have been leaked (Hunt). Using this check, one may determine if a password has already been leaked before. This can help to determine whether the password is safe to use. Forcing the user to choose a password that hasn’t already been leaked will help weed out bad passwords and thus limit the points of entry that malicious individuals have. Another action that is conjoined with the use of the HaveIBeenPwned API is limiting the number of password change attempts when the user is supplying bad passwords by logging them out. This helps disincentivize the usage of bad passwords, helps weed out bots trying to set deliberately insecure passwords, and helps ensure proper authentication by having the user login again.

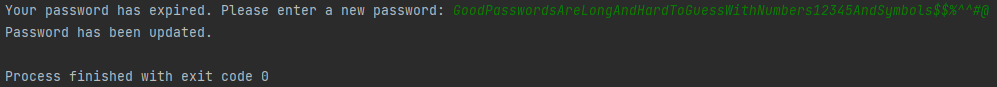
## Demonstration of Vulnerability Mitigation

To demonstrate that the HaveIBeenPwned API used in conjunction with limiting password change attempts can help to mitigate this vulnerability a demonstration using print statements has been setup. As demonstrated in the screenshots below, the program will reject bad passwords such as “password” as they have already been cracked and if the user attempts to change their password to a bad password five times in a row they are then logged out.

Text

Description automatically generated

If the user chooses a good password that has not already been cracked, then the software updates the password as expected.



## Verification of Mitigation Code

This demonstration was setup as its own java project and was imported from the haveibeenpwned4j GitHub repository and then modified. The haveibeenpwned4j library helps setup a client connection and acts as a front end to the HaveIBeenPwned API (Martinspielmann). The modifications took place in the main method of the HaveIBeenPwnedApiClient.java file in the haveibeenpwned4j library to setup a demonstration of how this library and API may be used in order to better the authentication standards of the company. The program starts out by creating a connection to the HaveIBeenPwned API

Text

Description automatically generated

The program then creates a scanner to get user inputs from the console, sets up three loop control variables, and prompts the user to change their password because it’s expired. The “badPassword” variable stores whether or not the user has entered in a good password yet. The “attempts” variable stores the number of attempts that the user has made to login. The “maxAttempts” variable store the maximum number of attempts that the user is allowed to make while changing their password.

Text

Description automatically generated

## Verification of Mitigation Code (Continued)

The program then starts a do while loop where it will get a string input from the user to be interpreted as the new string they wish to set their password to. This new password is then tested against the HaveIBeenPwned API. If this test is successful, meaning the password has not already been cracked, then the program prints a message telling the user that their password has been updated. If the password fails this check then the user is asked to enter a new password and the loop repeats. If the maximum number of attempts has been reached then the user is informed that they have been logged off and the program ends

Text

Description automatically generated

## Conclusion

The simple demonstration presented here shows that implementing such a program into the password changing process would be rather simple and could increase security immensely. Broken authentication is a serious problem and is currently number seven on the OWASP 2021 top 10 list (*A07:2021 – identification* 2021). The effects of a user gaining authentication that they shouldn’t have can be severe and many times irreparable. This mitigation is pulled directly from the OWASP Identification and Authentication Failures prevention methods. By testing new passwords against previously cracked password lists and limiting failed password changes, the security of our users’ passwords will be increased immensely which both increases user satisfaction along with company and user security.

**References**

Hunt, T. (n.d.). *API V3*. Have I Been Pwned. Retrieved November 15, 2021, from <https://haveibeenpwned.com/API/v3> .

Martinspielmann. (n.d.). *Martinspielmann/HAVEIBEENPWNED4J: The Ultimate Java Library for Troy Hunt's ';-- have I been pwned (V3).* GitHub. Retrieved November 15, 2021, from <https://github.com/martinspielmann/haveibeenpwned4j> .

Open Web Application Security Project. (2021, September 24). *A07:2021 – identification and authentication failures*. A07 Identification and Authentication Failures - OWASP Top 10:2021. Retrieved November 15, 2021, from https://owasp.org/Top10/A07\_2021-Identification\_and\_Authentication\_Failures/.