Prepared By: Brooklynn Silva & Jonathan Rumley

Supervisor: Douglas Lundin

Injection Vulnerability Mitigation

SQL Injection

Abstract

Insecure software attacks are executed in a variety of methods, but according to the OWASP Top 10, the most commonly known attack is an injection attack. This report will be focused on a SQL injection vulnerability attack found on a web application – Java Vulnerable Lab. This report was created to show the process of detecting, analyzing, and mitigating the SQL injection vulnerability while emphasizing the importance of the potential risks or impacts to the company, confirmation that the vulnerability can be mitigated, and to present a timeline starting with the attack and showing the proceeding steps.

Table of Contents

[**Abstract 1**](#_Toc57039627)

[**SQL Injection Vulnerability Detection 3**](#_Toc57039628)

[**Pre-Mitigation Code 4**](#_Toc57039629)

[**Pre-Mitigation URL 5**](#_Toc57039630)

[**Impacts to Company 5**](#_Toc57039631)

[**Confirmation of Vulnerability Mitigation 6**](#_Toc57039632)

[**Post-Mitigation Code 6**](#_Toc57039633)

[**Post-Mitigation URL 7**](#_Toc57039634)

[**Timeline 7**](#_Toc57039635)

[**References 8**](#_Toc57039636)

SQL Injection Vulnerability Detection

After the cyber-team ran the routine scans, a SQL injection vulnerability was detected on the Java Vulnerability Lab web application using the OWASP ZAP vulnerability scanner. Multiple SQL injection vulnerabilities came up in the routine scan, but we’re specifically focused on problem report #JVL13579 because it has been confirmed as an exploit increasing the priority from 7 to 9 out of 10. Any SQL injection attack is high urgency as a data spill could result in a countless number of damages or impacts, but 9 out of 10 requires immediate mitigation to prevent future attacks. Below provides a visual representation of how ZAP delivered us the SQL injection vulnerability details.

A picture containing graphical user interface, text

Description automatically generated

## Pre-Mitigation Code

Text

Description automatically generatedBelow displays the jsp file, download\_id.jsp, from the JVL source code. The cyber-team located the vulnerability at line 18. The code allows for potential attacks because the numeric parameter of the variable, String fileid;, is being treated as string and not an integer. Attackers can sneak into the web application attempting to exploit the URL parameters of certain aspects of the web application by using a variety of SQL injection attacks.

Pre-Mitigation URL

The cyber-team conducted a variety of SQL injection attacks on the vulnerable code to test for potential data spills and exploit the existing vulnerability, one example being shown below. Concatenating ‘and substring(version()1,1)=5’ at the end of the URL fileid parameter resulted in a data spill (file download) as the SQL injection was deemed ‘true’ by the original code in download\_id.jsp.

Graphical user interface, text, application, email, website

Description automatically generated

# Impacts to Company

Cyber security attacks can vary depending on the field of work, but attackers seem to find ways to exploit hardware, networks, and software leading to damages, negative impacts, and risk/posed threats to the companies attacked. These variables can vary drastically from company to company, but according to IBM, the average total cost of a data breach in 2020 is USD 3.86 million (Cost of Data Breach Study, 2020). The screenshot above shows the data spill that occurred after the SQL injection vulnerability was executed.

# Confirmation of Vulnerability Mitigation

Confirmation of the SQL injection vulnerability mitigation starts with successfully mitigating the vulnerability in the JVL source code, jsp file, download\_id.jsp. This vulnerability was mitigated by avoiding the concatenation of string values and converting the string values into integers. Our team corrected the source code by parsing an integer, as this is the safest approach to mitigation in this case. If the parameter entered is not an integer, the program will throw a parameter error message stating that a possible SQL injection attack could be in progress. This feature was added by our cyber-team as part of the mitigation to the vulnerability.

## Post-Mitigation Code

The DEV cyber-team modified the following code in the download\_id.jsp file and sent to PROD cyber-team to test the mitigated code.

Text

Description automatically generated

## Post-Mitigation URL

The SQL injection attack that was originally exploited was concatenated in the URL fileid parameter again, after the mitigation was complete. This resulted in an error message informing the user that a possible SQL injection attack was in progress, proving that the mitigation was a success.

Graphical user interface, application

Description automatically generatedTimeline

* Vulnerability scan of JVL performed in OWASP Zap by PROD cyber-team
* SQL injection identified in JVL source code jsp file, download\_id.jsp on line 18
* SQL injection vulnerability pre-mitigation process started by identifying that concatenation of string values is vulnerable to attacks from manipulation of the URL fileid parameter
* SQL injection vulnerability post-mitigation process was completed by DEV cyber-team to add depth to the jsp file by parsing the integers from the string values and adding an error message that will deploy if SQL injection is attempted
* Future manipulation to URL fileid parameter resulted in error message as expected
* SQL injection vulnerability report written and presented to management in order to bring attention to the vulnerability and prevent future data spills

**References**

Cost of a Data Breach Study. (2020). *IBM.* Retrieved November 11, 2020, from https://www.ibm.com/security/data-breach

OWASP Top 10. (2020). *OWASP.* Retrieved November 23, 2020, from https://www.owasp.org/www-project-top-ten/