Lab 5: Mitigation of SQL Injection

Thangamuthu Balaji

I. Introduction

This lab report examines SQL Injection mitigation techniques and other injection flaws, specifically command injection, log spoofing, and XPath injection, using Injection vulnerabilities are a big security issue because they let attackers mess with a web app by adding harmful code. This lab showed how these attacks work and taught us how to write code safely to stop them. We used WebGoat versions 2023.8 and 7.1 to practice these skills..

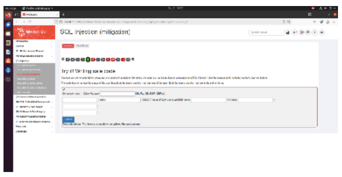
II. WebGoat 8 - 5. Writing a Safe Code

The exercise gave us a Java code example with an SQL query that used placeholders (?) instead of putting user input directly into the query. By using prepared statements and parameterized queries, we can stop SQL injection because it treats user input as data, not code. I updated the code to make it safe from SQL injection by setting up a PreparedStatement with parameters instead of using fixed values.
getConnection
PreparedStatement statement
prepareStatement
?
?
setString,
setString

III. WebGoat 8 - 6. Writing Safe Code

This exercise built on the last one by using JDBC to connect to a database safely and run queries that can't be tricked by SQL injection. The code needed a PreparedStatement with parameters to get user data securely. Just like in Part 1, I used PreparedStatement to connect to the database and added parameters instead of using direct input. This way, the query stays safe even if someone tries to enter harmful SQL commands.

statement.executeUpdate();
catch (Exception e) System.out.println(”Oops. Something went wrong!”);



(a) 5. Writing Safe Code

(b) 6. Writing Safe Code

IV. WebGoat 8 - 9. Input Validation Alone is Not Enough

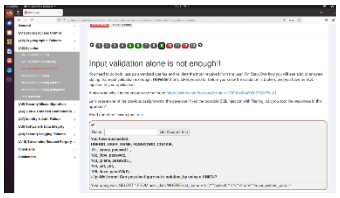
The exercise demonstrated that input validation alone is inadequate for preventing SQL injection attacks. It presented a scenario where filtered but non-parameterized input still permitted injection. I tested various SQL injection payloads and found that I could bypass the input validation by injecting carefully crafted SQL syntax, allowing me to execute unauthorized queries. While the developer had implemented filtering mechanisms, these were insufficient without the use of parameterized queries. The exercise emphasized that parameterization is essential, alongside input validation, to effectively mitigate SQL injection risks. Input validation can reduce the risk, but it cannot fully prevent SQL injection on its own. Code used: a’/**/union/**/select/**/user system data.*,’1’,’1’,1/**/from/
*/user_system_data;–

V. WebGoat 8 - 10. Input Validation Alone is Not Enough

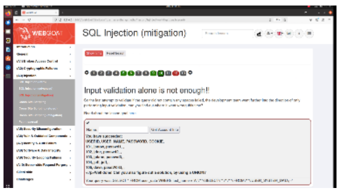
This exercise built on the earlier example to show that if validation is done incorrectly, it won't stop SQL injection attacks. The task demonstrated how clever payloads can still get past even strong input validation. I tested different types of injections using changed syntax and formats, like encoded characters, to find weaknesses. In the end, the exercise highlighted that using parameterized queries is a much better way to defend against these attacks. To effectively prevent SQL injection, it’s important to combine input validation with the right use of parameterized queries to keep applications safe.Code used:
a’;/**/seselectlect/**/*/**/frfromom/**/user_system_data;–

VI. WebGoat 7 - Command Injection

In this exercise, I interrupted traffic and modified a parameter to inject commands. By replacing the HelpFile parameter with ”; ls;#” in the HTTP


Input validation alone is not enough!!

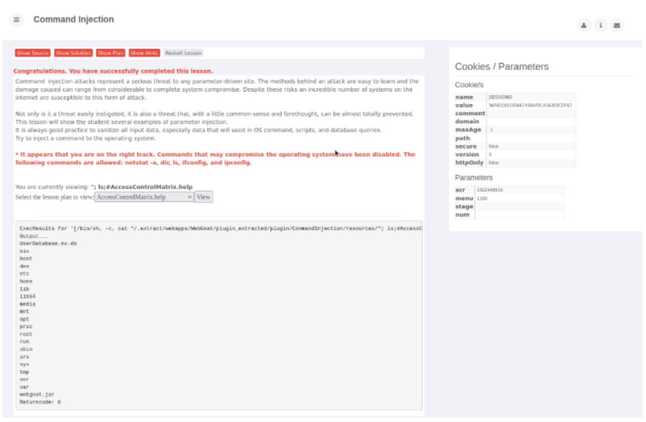
(a) 9. Input Validation Alone is Not Enough


SQL Injection (mitigation)

Input validation alone is not enough!!

(b) 10. Input Validation Alone is Not Enough

I ran an ls command to show the files on the server. This showed how not checking inputs in system calls can let commands run when they shouldn't. To stop command injection, it's important to clean up inputs, especially in areas that work with commands from the operating system.


Command Injection

Congratulations. You have successfully completed this lesson.

Command injection attacks represent a serious threat to any parameter-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can be almost totally prevented. This lesson will show the student several examples of parameter injection.
It is always good practice to sanitize all input data, especially data that will used in OS command, scripts, and database queries.
Try to inject a command to the operating system.

* It appears that you are on the right track. Commands that may compromise the operating system have been disabled. The following commands are allowed: netstat -a, dir, ls, ifconfig, and ipconfig.

You are currently viewing: "; ls;#AccessControlMatrix.help
Select the lesson plan to view: AccessControlMatrix.help View

```
ExecResults for '[/bin/sh, -c, cat "/.extract/webapps/WebGoat/plugin_extracted/plugin/CommandInjection/resources/"; ls;#AccessC
Output...
UserDatabase.mv.db
bin
boot
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
webgoat.jar
Returncode: 0
```

Cookies / Parameters

name	JSESSIONID
value	96F6018518566745 8AF81A562E8CDF67
comment	
domain	
maxAge	1
path	
secure	false
version	0
httpOnly	false

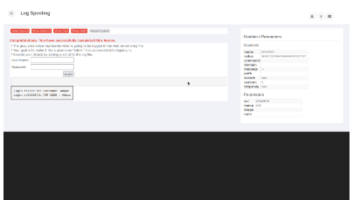
Parameters

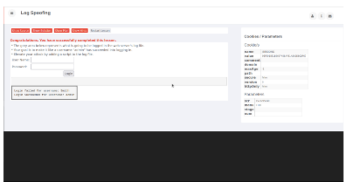
scr	1922448916
menu	1100
stage	
num	

Fig. 3: Command Injection

VII. WebGoat 7 - Log Spoofing

Log spoofing attacks change log files to hide or fake activities. In this exercise, I learned how to add new lines to log files using special characters. By entering %0d%0a in the username field, I created a new log entry that showed a successful login for another user, an Admin. This example illustrates how someone could add false information to logs. To prevent log injection attacks, it's important to properly encode and clean log inputs. Logs should be set up to identify and block escape sequences.. Code used: test%0aLogin succeeded for username: ¡script¿alert('xss')¡/script¿admin



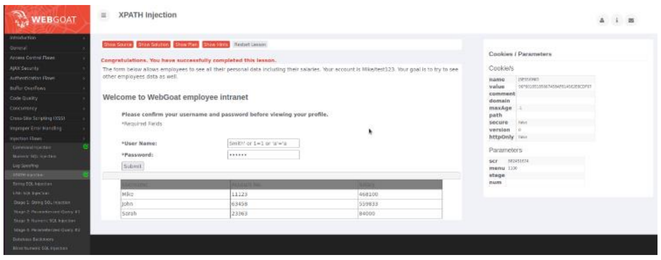

Log Spoofing

(b) Log Spoofing

VIII. WebGoat 7 - XPath Injection

XPath injection is like SQL injection but affects XML-based data. I used XPath injection to get unauthorized access to XML data by getting around the login system. By entering Smith' or 1=1 or 'a'='a in the username field, I bypassed the security check and got access to all employee records. This shows why it's risky not to validate XPath queries. To stop XPath injection, it's important to use parameterized XPath expressions and check inputs carefully. Without these protections, sensitive XML data can be exposed. Code used:
Mike' or 1=1 or 'a'='a
test123 (password)

Fig. 5: XPath Injection



XPATH Injection

Congratulations. You have successfully completed this lesson.

The form below allows employees to see all their personal data including their salaries. Your account is Mike/test123. Your goal is to try to see other employees data as well.

Welcome to WebGoat employee intranet

Please confirm your username and password before viewing your profile.
*Required Fields

*User Name: Smith' or 1=1 or 'a'='a
*Password: ******

Username	Account No.	Salary
Mike	11123	468100
John	63458	559833
Sarah	23363	84000

IX. Conclusion

This lab showed different types of injection flaws and ways to prevent them, like stopping SQL injection and other injection attacks. Each exercise highlighted the importance of strong input validation, using parameterized queries, and cleaning up inputs. Secure coding practices are key to stopping injection attacks, and just relying on input validation isn't enough. By keeping data separate from code, developers can help protect applications from injection risks.

References

[1] OWASP SQL Injection Prevention Cheat Sheet-https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
[2] OWASP Command Injection - https://owasp.org/www.community/attacks/Command_Injection
[3] OWASP Log Injection - https://owasp.org/www-community/attacks/Log_Injection
[4] OWASP XPath Injection - https://owasp.org/www-community/attacks/XPATH_ Injection