Table of Contents

[Task 1 1](#_Toc170697764)

[SQL Injection Vulnerability 1](#_Toc170697765)

[Overview 1](#_Toc170697766)

[Important Components 1](#_Toc170697767)

[Detailed Component Description 2](#_Toc170697768)

[1. Simple Task 2](#_Toc170697769)

[2. Database Layout 2](#_Toc170697770)

[3. SQL Injection Vulnerability 2](#_Toc170697771)

[Exploitation Example: 2](#_Toc170697772)

[4. SQL Injection Mitigation 4](#_Toc170697773)

[SQL Injection Incident Analysis: The 2014 Sony Pictures Hack 5](#_Toc170697774)

[Conclusion 7](#_Toc170697775)

[References 7](#_Toc170697776)

# Task 1

# SQL Injection Vulnerability

## Overview

This project demonstrates the impact of SQL injection vulnerabilities through a simple web application designed with both insecure and secure implementations. The application allows users to log in using a username and password, interacting with a MySQL database via a Flask backend.

## Important Components

Backend (Flask Application):

* Insecure Version: Contains code vulnerable to SQL injection.
* Secure Version: Implements mitigations to prevent SQL injection.

Frontend (HTML):

* Simple login form for user input.

Database (MySQL):

* Stores user credentials.

Setup Scripts:

* SQL script for database creation and initial data population.
* Requirements file for installing necessary Python packages.

Component Connections

* Frontend to Backend: The HTML form submits user credentials to the Flask backend.
* Backend to Database: The Flask application connects to the MySQL database to validate user credentials.

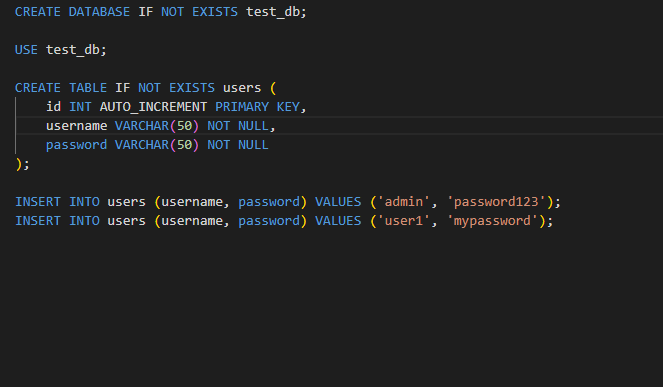
## Detailed Component Description

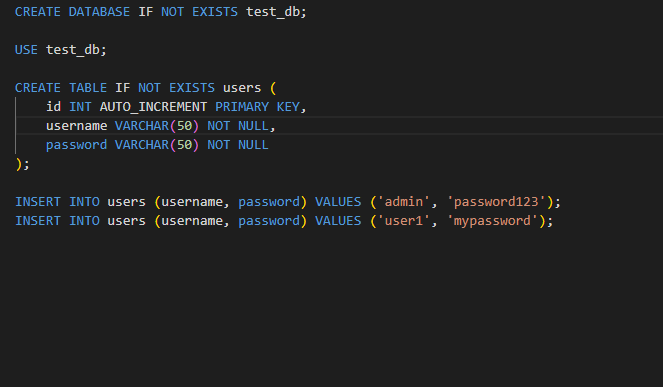
### 1. Simple Task

The task is to create a login form where users input their username and password. The backend verifies these credentials against the database. If the credentials match, the user is logged in; otherwise, the login fails.

### 2. Database Layout

The database has a single table named users with the following schema:

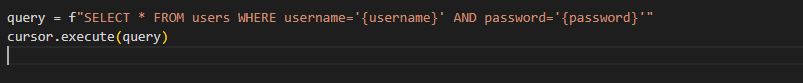
Initial data includes:



### 3. SQL Injection Vulnerability

In the insecure version, user inputs are directly embedded into the SQL query string, making it vulnerable to SQL injection.

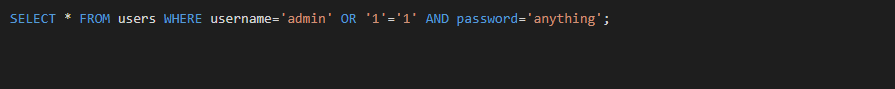
Insecure Code Example:

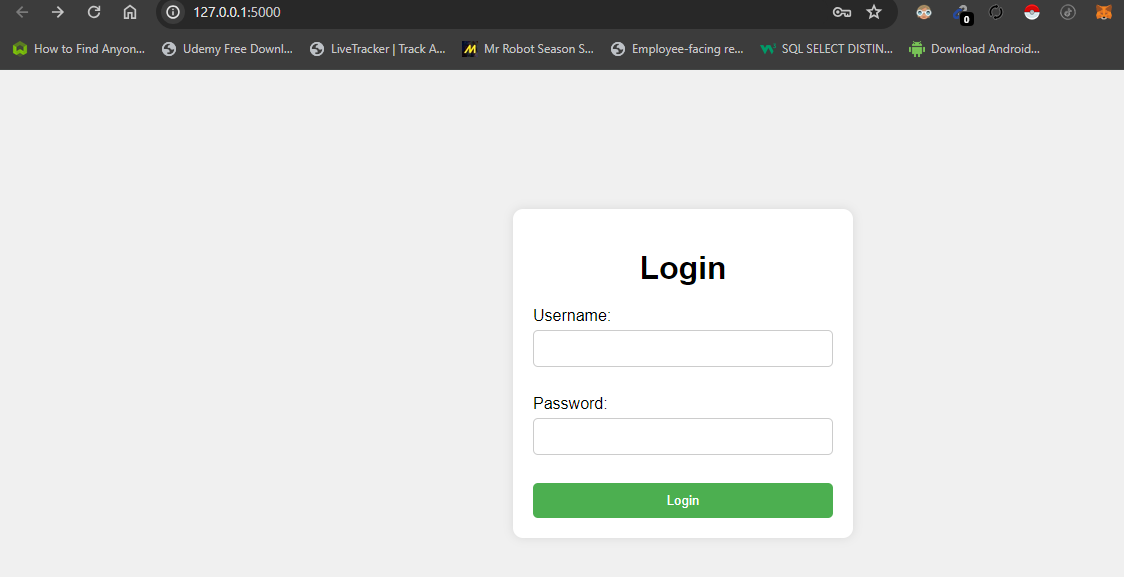


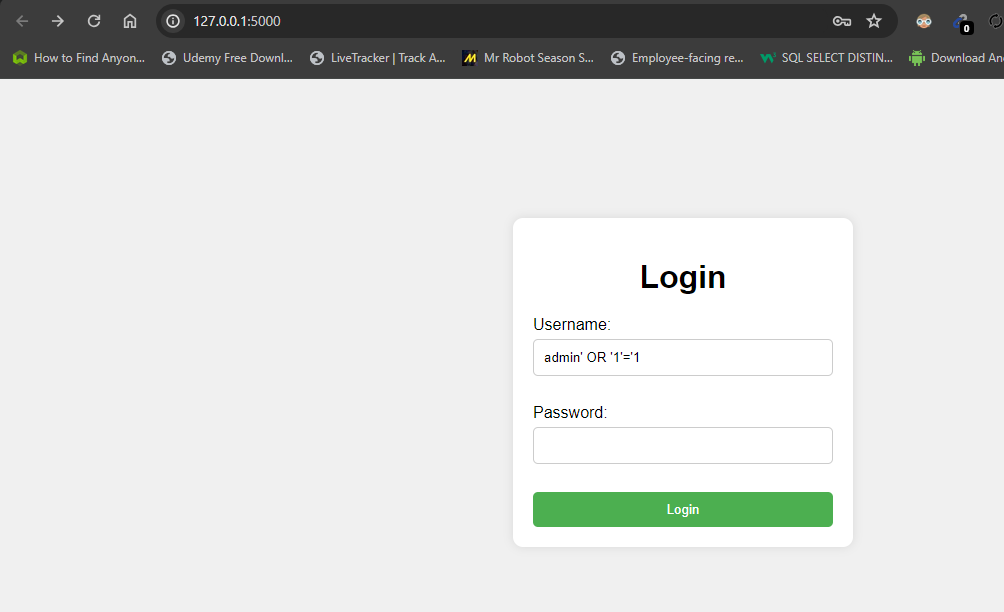
This approach does not sanitize or validate the inputs, allowing an attacker to manipulate the SQL query.

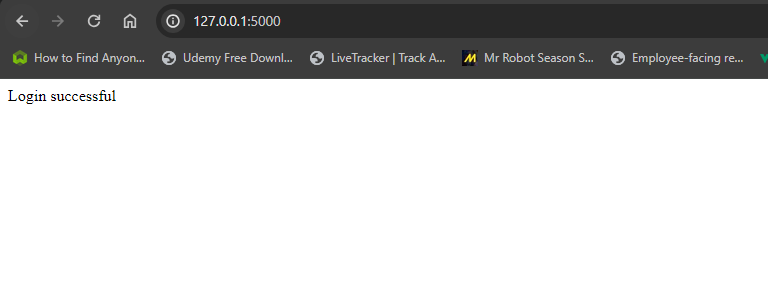
### Exploitation Example:

If an attacker inputs admin' OR '1'='1 as the username, the resulting query becomes:

This query always evaluates to true, granting unauthorized access.





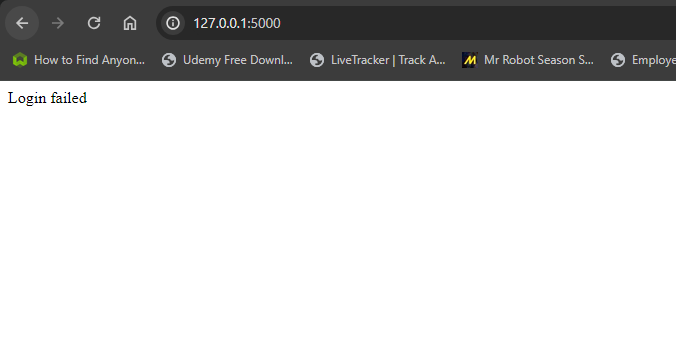


### 4. SQL Injection Mitigation

The secure version uses parameterized queries to prevent SQL injection. Parameterized queries ensure that user inputs are treated as data rather than executable code.

Secure Code Example:





cursor.execute(query, (username, password))

Here, the user inputs are safely passed as parameters to the query.

Why Parameterized Queries Fix the Vulnerability

Parameterized queries separate SQL logic from user input. The database engine distinguishes between code and data, making it impossible for an attacker to alter the intended query structure. This ensures that user inputs cannot interfere with SQL commands, effectively mitigating SQL injection risks.

## SQL Injection Incident Analysis: The 2014 Sony Pictures Hack

Background

Cyber-attack on Sony Pictures Entertainment in 2014 is one of the famous examples of SQL injection attack. Based on the incident, the hackers called themselves ‘Guardians of Peace’ (GOP), the attack was very damaging to Sony Pictures Entertainment (SPE) because large amount of data was stolen, the company incurred major costs, and was also humiliated (engadget, 2014).

Incident Overview

The hackers applied a Trojan virus, SQL injection to breach into the network of Sony and copy out tons of confidential information. The breach included employees’ record, unpublished movies, strategic plans, and private correspondence (engadget, 2014). .

SQL Injection Mechanism

For its part, SQL injection or SQLi is a code injection technique where the attacker takes advantage of the application’s software weaknesses to include a malicious SQL statement in a query. If an application directly includes users’ inputs in SQL queries without encoding them, it opens the door to SQL injection.

It was identified that in the Sony hack, the attackers targeted a weak web application in the organization’s network (engadget, 2014). By exploiting the SQL injection vulnerability, they were able to:

* Gain unauthorized access to the database.
* Extract sensitive information, such as user credentials.
* Escalate their privileges within the network.
* Deploy malware to further compromise systems and exfiltrate data.

**Detailed Analysis**

**Vulnerability Discovery:**

The hackers most certainly employed automated instruments to seek for SQL injection flaws in Sony’s Web applications. Other tools like the sqlmap can also help in detecting the sql injection points and also exploit by yielding data.

**Exploitation:**

When an endpoint was identified as weak, the hackers customised SQL injection queries to manipulate the database (technology & gadgets, 14-11-2014).

The given query is designed to avoid user authentication by rewriting it as ‘1=1’ and providing access to the attacker in the form of an admin.

**Privilege Escalation**:

While having the access to the database, the attackers stole the administrative login details, and then used them to spread across the network (engadget, 2014). . They probably used other weaknesses and poor security measures to take over more systems on the network.

**Data Exfiltration:**

They were able to transfer large volumes of data such as emails, payment records, and business information out of the targeted organizations. They employed the contaminated network connection to steal data from Sony’s network without being noticed.

**Impact and Consequences**

* Many costs such as those encountered in the identification, assessment, communication and rectification of incidents, attorney fees, and loss of revenue due to delays in film release were observed on this company.
* It revealed communications that the public had no business knowing about and impounded the company’s relationship with employers and other affiliates.
* The attack disrupted daily course of business, employees’ performance, and current projects.
* Sony also received legal actions and unfavorable regulation regarding inadequacy of the protection of such important information..

**Mitigation and Prevention**

The most effective way to prevent SQL injection is by using parameterized queries or prepared statements, which separate SQL code from data inputs. This ensures that user inputs are treated strictly as data, not executable code.

**Example of Parameterized Query:**

query = "SELECT \* FROM users WHERE username=%s AND password=%s"

cursor.execute(query, (username, password))

Ensure daily/weekly security audit on the websites as well as also periodic code audits to address security risks in the web applications.

Convey to the company employees the correct measures in the protection of the data and cybersecurity.

IDS as a tool should be put in place to monitor Network traffic activity and take action in instance of threats.

One of the most recent attacks that were highly devastating is the Sony Pictures hack in 2014 (engadget, 2014). . To sum up, it is possible to conclude that, knowing the techniques of the SQL injection attacks and applying strict access security measures, organizations may prevent similar threats. Due to the uncertainties associated with attacks, proper input validation, security audits, and a proper security plan are critical in ensuring the safety of critical and customer information.

## Conclusion

This project show the necessity of protecting web applications against the SQL injection. From the comparison between the insecure and secure implementation of the webpages, it is made clear why parameterized queries must be used. This way, the inputs from the user are properly handled and the application shielded from any attack.

# References

Engadget is part of the Yahoo family of brands,” Dec. 10, 2014.

https://www.engadget.com/2014-12-10-sony-pictures-hack-the-whole-story.html

K. Young, “Cyber Case Study: Sony Pictures Entertainment Hack,” *CoverLink Insurance - Ohio Insurance Agency*, Nov. 01, 2021. https://coverlink.com/case-study/sony-pictures-entertainment-hack/