Mitigating SQL Injection Vulnerabilities in a Login Page

Muhammad Shahmir 21K-3563
Khizer Nadeem 21K-4768
Danial 21K-3577

Abstract: This report presents a detailed analysis of a project aimed at mitigating SQL injection vulnerabilities in web applications. SQL injection remains one of the most prevalent and damaging security threats to web applications, allowing attackers to manipulate databases and potentially access sensitive information. The project employs various techniques and best practices to prevent different types of SQL injection attacks, including classic, blind, time-based, second-order, out-of-band, and union-based attacks. Through the implementation of parameterized queries, input validation, query time limits, and other security measures, the project effectively safeguards against SQL injection vulnerabilities.

Introduction: SQL injection vulnerabilities pose a significant threat to the security of web applications, allowing attackers to execute malicious SQL queries and potentially compromise databases. This report outlines a project designed to address various types of SQL injection attacks by implementing robust security measures and best practices. The project focuses on preventing classic SQL injection as well as more advanced techniques such as blind SQL injection and time-based SQL injection.

Background: SQL injection attacks occur when malicious users exploit vulnerabilities in web applications to inject malicious SQL code into input fields. This injected SQL code can then be executed by the application's database server, leading to unauthorized access, data manipulation, or other malicious activities. Traditional methods of SQL injection prevention include input validation, sanitization, and the use of parameterized queries or prepared statements.

Project Overview: The project consists of developing a login portal for a web application, incorporating various security measures to prevent SQL injection vulnerabilities. Key components of the project include:

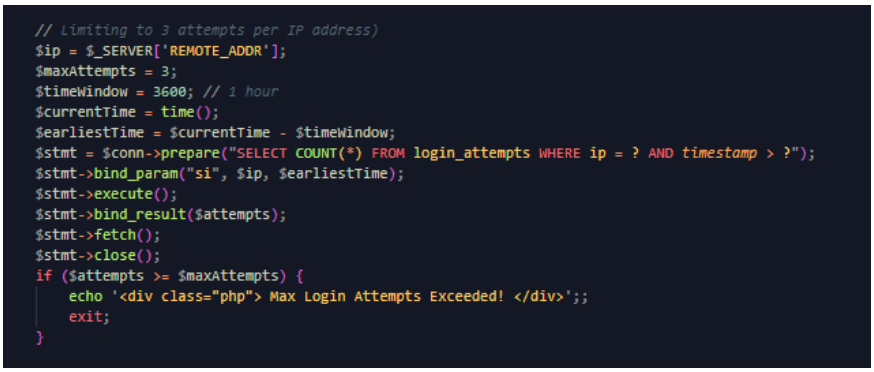
1. **Input Validation:** Stringent validation of user input to prevent the injection of malicious characters or SQL commands.

```php
// Input Validation
if (!preg_match("/^[a-zA-Z0-9]*$/", $uname)) {
    echo '<div class="php"> Invalid Characters Detected! </div>';
    exit;
}
```

2. **Parameterized Queries:** Adoption of parameterized queries or prepared statements to separate SQL code from user input, preventing direct execution of injected SQL commands.

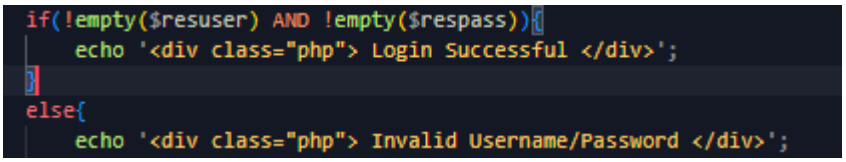
```php
// SQL Injection Prevention
$sql = "SELECT username,password FROM users WHERE username = ? AND password = ?";
$stmt = mysqli_prepare($conn, $sql);
if($stmt){
    mysqli_stmt_bind_param($stmt,"ss", $uname, $password);
    mysqli_stmt_execute($stmt);
    mysqli_stmt_bind_result($stmt, $resuser, $respass);
    mysqli_stmt_fetch($stmt);
    if(!empty($resuser) AND !empty($respass)){
        echo '<div class="php"> Login Successful </div>';
    }
    else{
        echo '<div class="php"> Invalid Username/Password </div>';
        // Record failed login attempt
        $stmt2 = $conn->prepare("INSERT INTO login_attempts (ip, timestamp) VALUES (?, ?)");
        $stmt2->bind_param("si", $ip, $currentTime);
        $stmt2->execute();
        $stmt2->close();
    }
    mysqli_stmt_close($stmt);
}
}
?>
```

3. **Query Time Limits:** Implementation of query time limits to thwart time-based SQL injection attacks by restricting the execution time of SQL queries.



```php
// Limiting to 3 attempts per IP address)
$ip = $_SERVER['REMOTE_ADDR'];
$maxAttempts = 3;
$timeWindow = 3600; // 1 hour
$currentTime = time();
$earliestTime = $currentTime - $timeWindow;
$stmt = $conn->prepare("SELECT COUNT(*) FROM login_attempts WHERE ip = ? AND timestamp > ?");
$stmt->bind_param("si", $ip, $earliestTime);
$stmt->execute();
$stmt->bind_result($attempts);
$stmt->fetch();
$stmt->close();
if ($attempts >= $maxAttempts) {
    echo '<div class="php"> Max Login Attempts Exceeded! </div>';;
    exit;
}
```

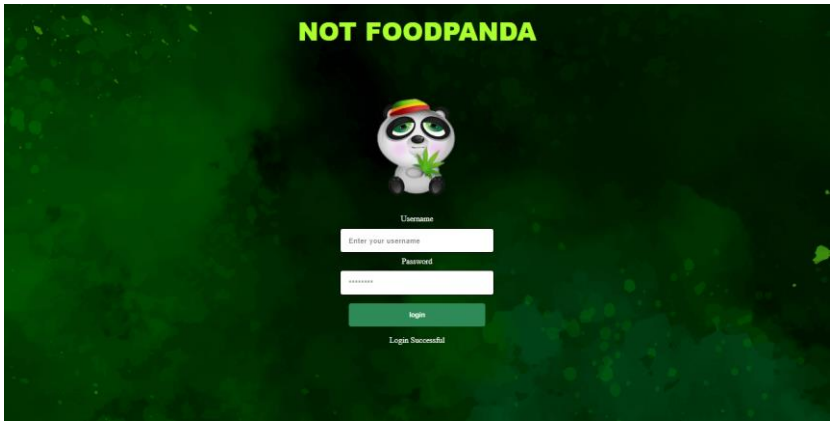
4. **Error Handling:** Minimization of verbose error messages to prevent disclosure of sensitive information that could aid attackers in exploiting SQL injection vulnerabilities.



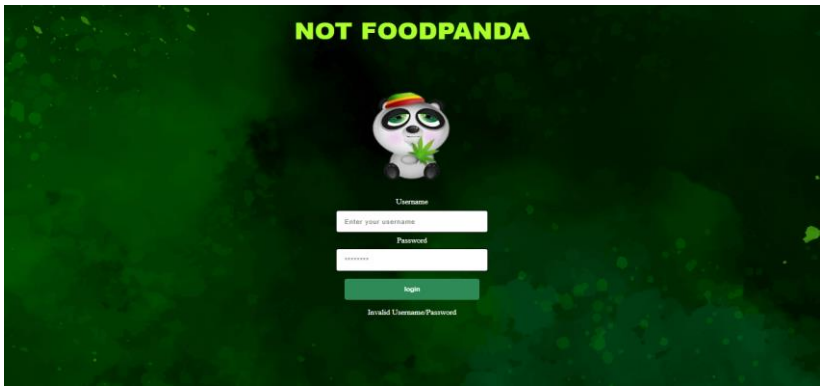
```php
if(!empty($resuser) AND !empty($respass)){
    echo '<div class="php"> Login Successful </div>';
}
else{
    echo '<div class="php"> Invalid Username/Password </div>';
```

Implementation: The project's implementation involves the creation of a login portal using HTML, CSS, and PHP. The PHP code utilizes prepared statements to execute SQL queries securely and incorporates input validation to sanitize user input effectively. Additionally, the project includes measures to limit the number of login attempts per IP address and record failed login attempts to prevent brute-force attacks.

Successful Login:

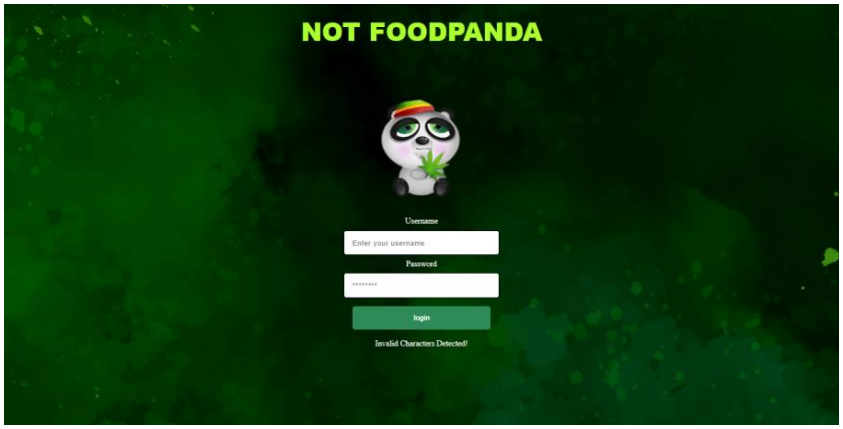


Unsuccessful Login:

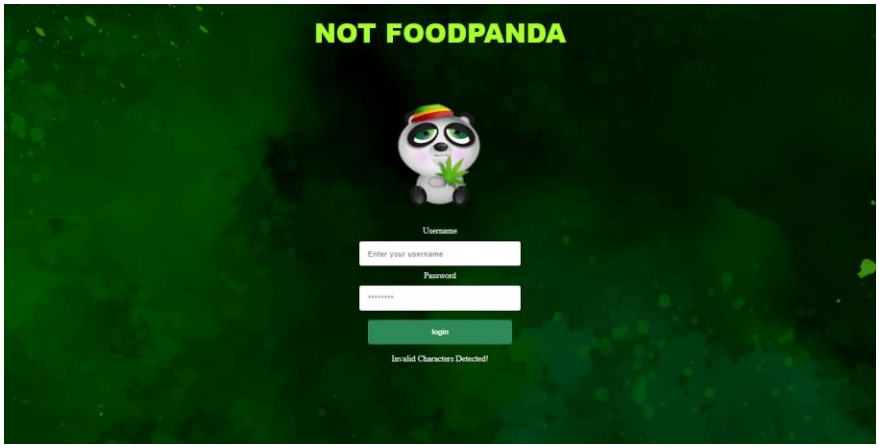


SQL Injection Prevention:

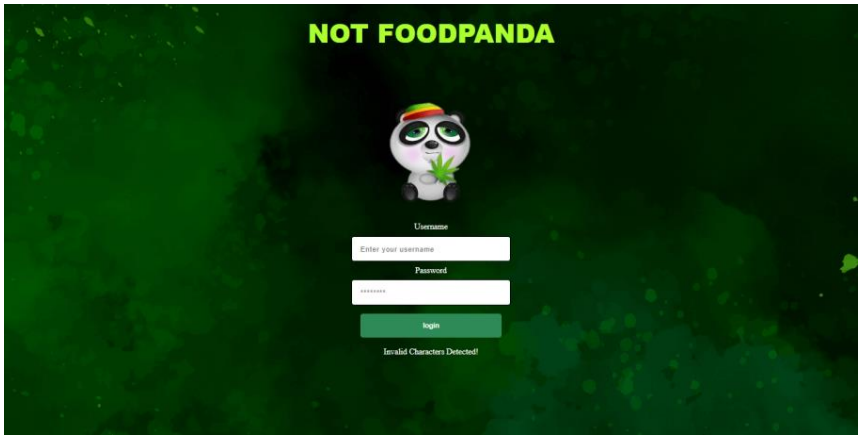
Payload: ' OR '1'='1



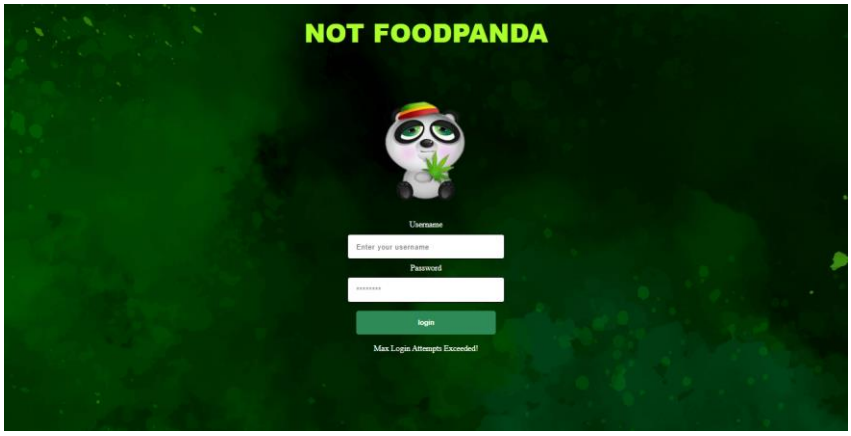
Payload: %27%20OR%20%271%27%3D%271 (URL Encoded)



Payload:'OR'1'='1 **(HTML Entity Encoding)**



Time Based SQL Injection Prevention:



Results and Evaluation: The implemented security measures effectively mitigate SQL injection vulnerabilities in the login portal. Through rigorous input validation, parameterized queries, and query time limits, the project successfully prevents various types of SQL injection attacks, including classic, blind, time-based.

Conclusion: The project demonstrates the importance of implementing comprehensive security measures to mitigate SQL injection vulnerabilities in web applications effectively. By combining input

validation, parameterized queries, query time limits, and continuous monitoring, the project provides robust protection against various types of SQL injection attacks. Future enhancements may include additional security features such as web application firewalls and intrusion detection systems to further fortify the application's defences against evolving cyber threats.