## **[Online Class and Exam Scheduling System](https://code-projects.org/online-class-and-exam-scheduling-system-in-php-with-source-code/)**

## --designation.php

### NAME OF AFFECTED PRODUCT(S)

[Online Class and Exam Scheduling System In PHP With Source Code](https://code-projects.org/online-class-and-exam-scheduling-system-in-php-with-source-code/)

### Vendor Homepage

https://code-projects.org/online-class-and-exam-scheduling-system-in-php-with-source-code/

### Vulnerable File

designation.php

### Software Link

https://download.code-projects.org/details/93487762-3e23-48ab-a56f-af5e61441ee1

### Vulnerability Type

/Scheduling PHP/scheduling/pages/designation.php

### Root Cause

A SQL injection vulnerability was found in the 'designation.php' file of the 'pages' project. This issue occurs because an attacker injects malicious code from the parameter "designation” and uses it directly in SQL queries without proper scrubbing or validation. This allows an attacker to forge input values to manipulate SQL queries and perform unauthorized actions.

### Impact

Attackers can exploit this SQL injection vulnerability to achieve unauthorized database access, sensitive data leakage, data tampering, comprehensive system control, and even service interruption, posing a serious threat to system security and business continuity.

### source

<div class="row">

<div class="col-md-12">

<form method="post" action="designation\_update.php">

<div class="form-group">

<label **for**="date">Update Designation</label><br>

<input type="hidden" class="form-control" id="id" name="id" **value**="<?php echo $\_REQUEST['id'];?>" readonly>

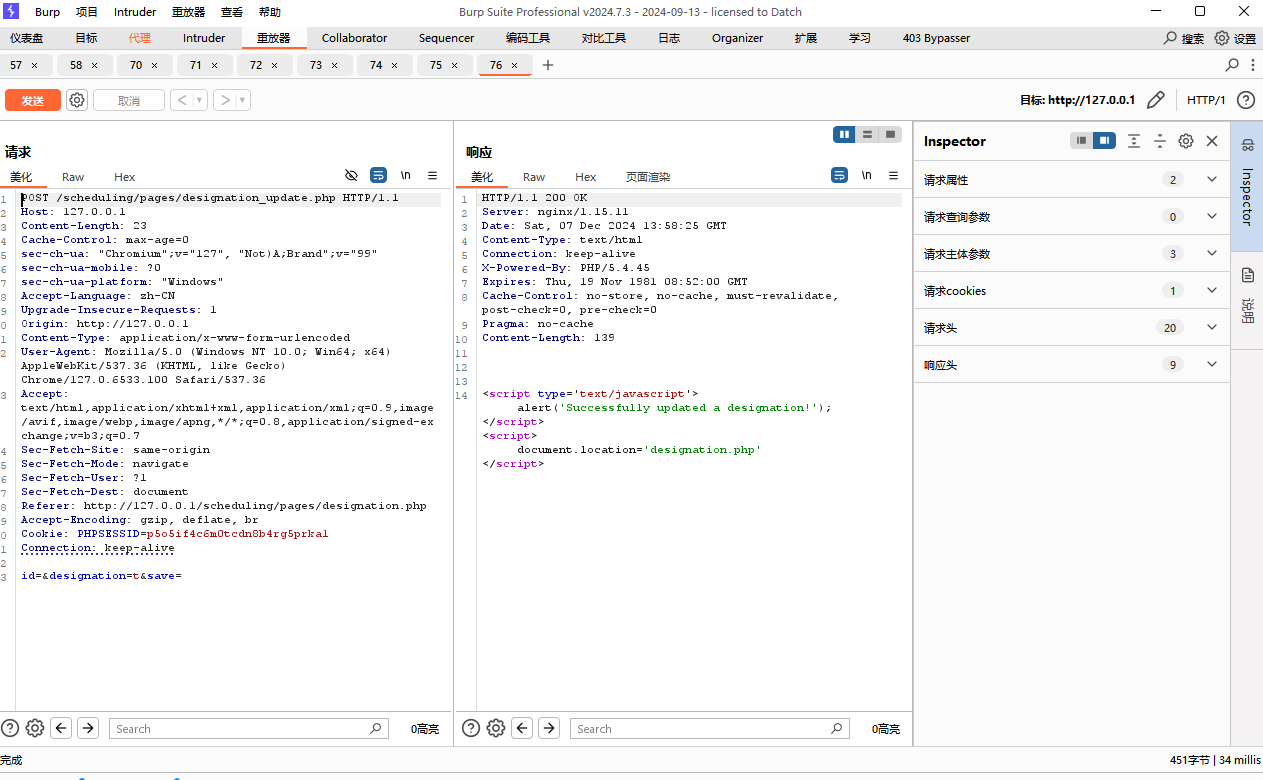
<input type="text" class="form-control" id="class" name="designation" **value**="<?php echo $\_REQUEST['designation'];?>" placeholder="Designation" required>

</div><!-- /.form group -->

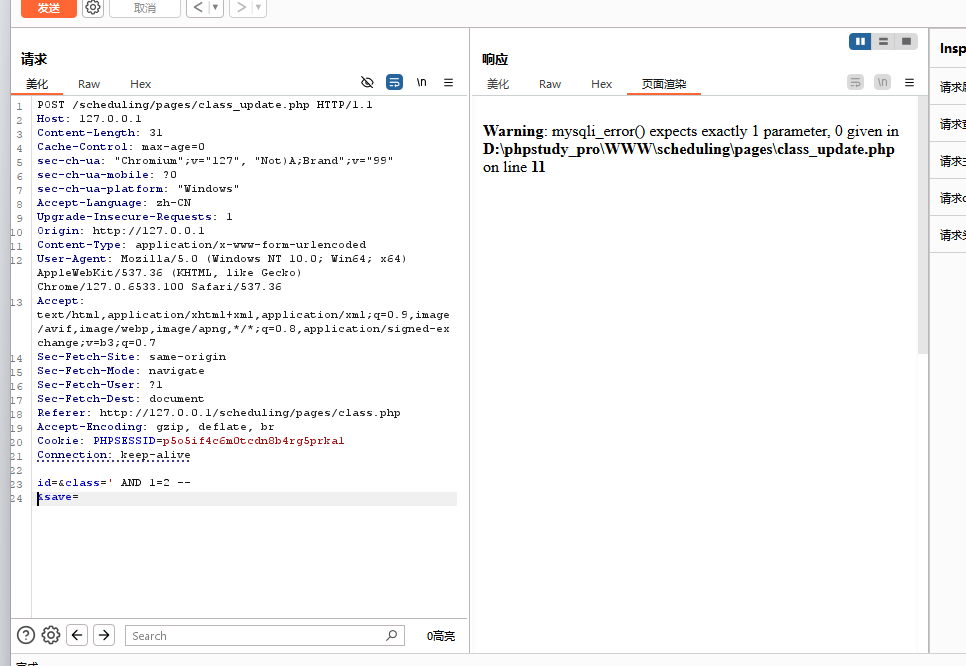
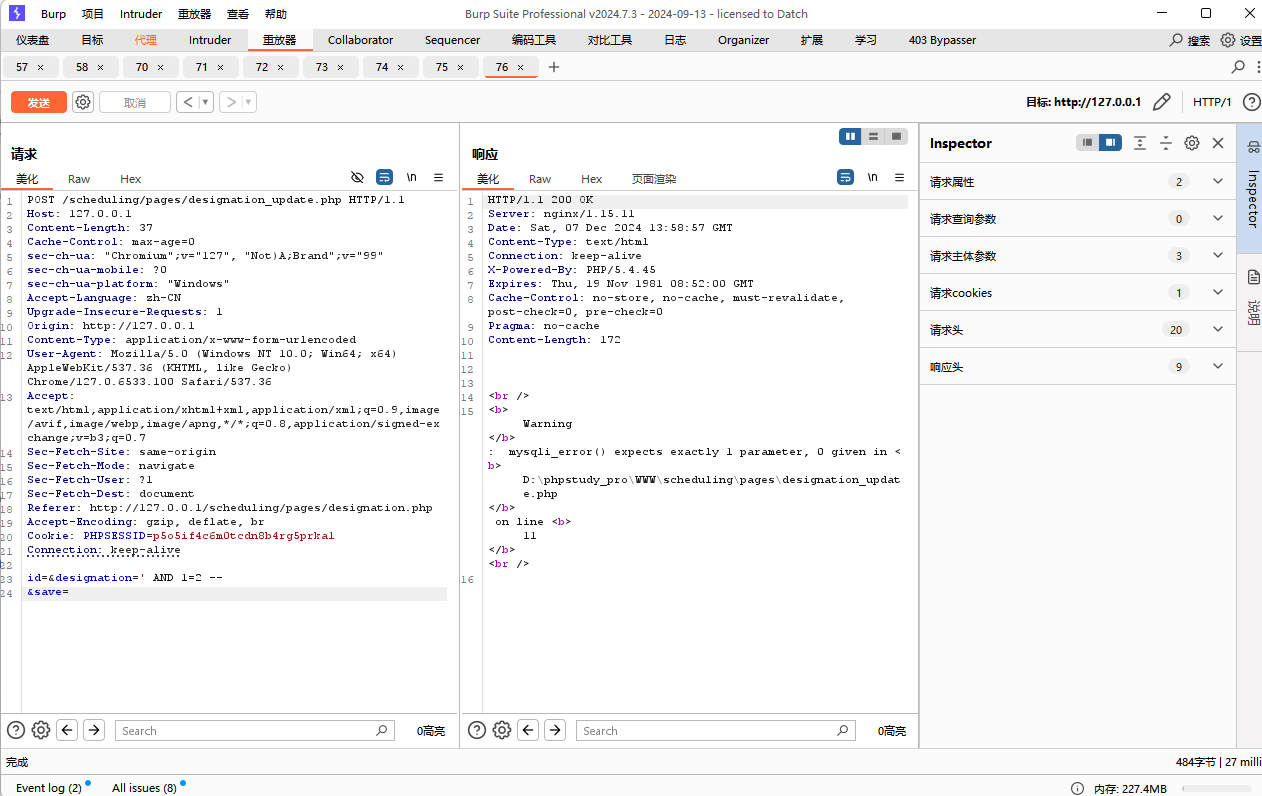
</div>

### </div>

### attack



If you enter ' AND 1=2 --, an error message is displayed



### Suggested repair

Use prepared statements and parameter binding:  
Preparing statements can prevent SQL injection as they separate SQL code from user input data. When using prepare statements, the value entered by the user is treated as pure data and will not be interpreted as SQL code.

Input validation and filtering:  
Strictly validate and filter user input data to ensure it conforms to the expected format.

Minimize database user permissions:  
Ensure that the account used to connect to the database has the minimum necessary permissions. Avoid using accounts with advanced permissions (such as' root 'or' admin ') for daily operations.

Regular security audits:  
Regularly conduct code and system security audits to promptly identify and fix potential security vulnerabilities.