Time-Based SQL Injection in was found in “mcgs/admin/view-enquiry.php?viewid=160" in PHPGurukul Medical Card Generation System v1.0 allows remote attackers to execute arbitrary code via "viewid" GET request parameter.

Vendor Information:

Vendor Name: PHPGurukul

Affected Product Name: Medical Card Generation System

Product URL: https://phpgurukul.com/medical-card-generation-system-using-php-and-mysql/

Version: V 1.0

Affected URL: http://localhost/mcgs/admin/view-enquiry.php?viewid=160

Vulnerable Parameter: viewid

Vulnerability Type:

Category: Time-Based SQL Injection (SQLI)

Severity: High

Impact: Data leakage, database disruption, potential privilege escalation, denial of service (DOS)

Summary:

A Time-Based SQL Injection vulnerability was identified in the Medical Card Generation System by PHPGurukul. This vulnerability allows an attacker to interfere with the queries that the web application makes to its database. By manipulating the input through specially crafted SQL queries, an attacker can cause delays in the database response time, confirming the existence of the vulnerability. In this case, the vulnerability is found in the viewid parameter of the view-enquiry.php page.

Steps to Reproduce:

1. Navigate to Vulnerable URL:

- **The vulnerable URL is**: http://localhost/mcgs/admin/view-enquiry.php?viewid=160

1. 2. **Payload**: +and+(select+*+from+(select(sleep(10)))a)--

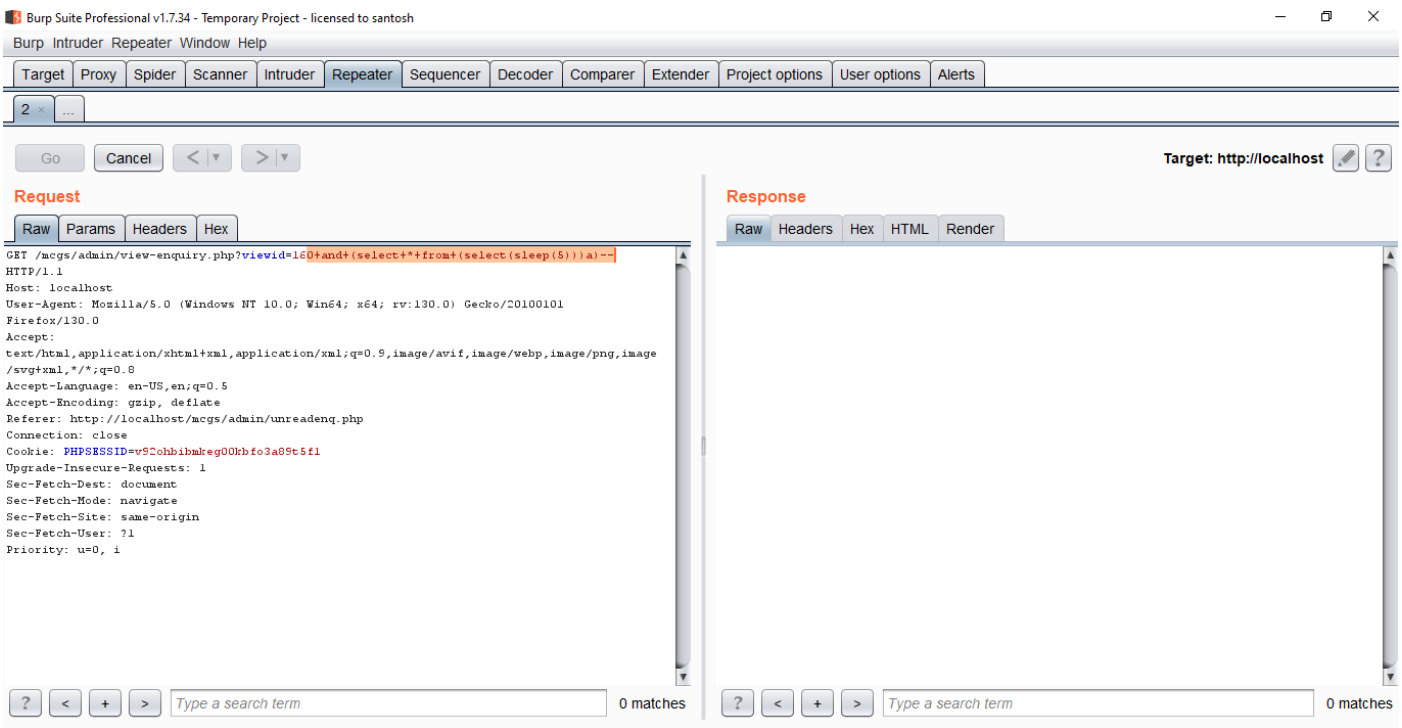
- **Inject the following SQL payload into the viewid parameter to test for a Time-Based SQL Injection:**
  http://localhost/mcgs/admin/view-enquiry.php?viewid=160' AND (SELECT * FROM (SELECT(SLEEP(5)))a)--

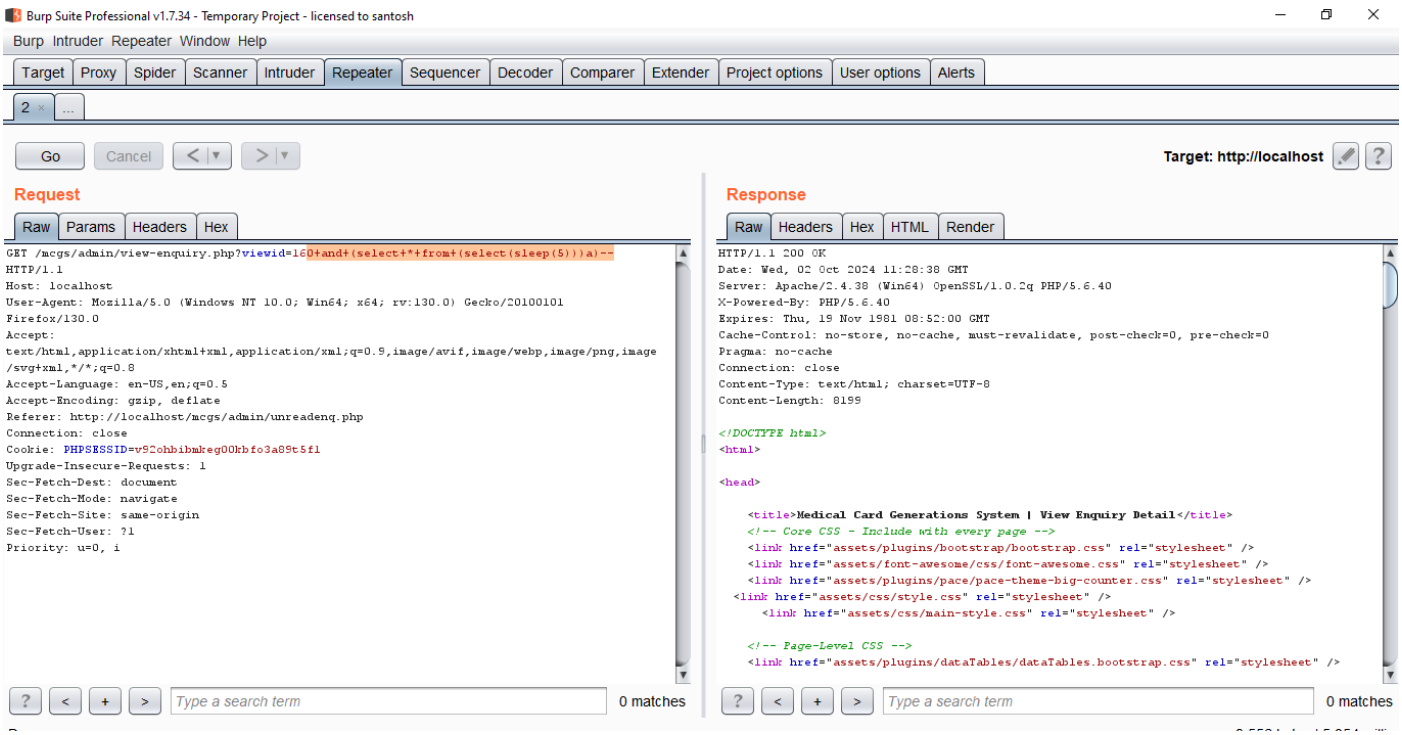
2. 3. **Expected Outcome:**

- If the vulnerability exists, the page will take 5 seconds longer than usual to respond. This indicates that the SQL query was successfully executed and delayed by the SLEEP(5) function, confirming the vulnerability.

Proof of Concept (PoC):

Watch the PoC video demonstration here: Click Here





Technical Details:

The SQL injection occurs due to improper validation or sanitization of user input in the viewid parameter. The injected query alters the normal execution flow of the database and uses the SLEEP function to delay the response:

/mcgs/admin/view-enquiry.php?viewid=160+and+(select+*+from+(select(sleep(10)))a)--

Here, SLEEP(5) forces the database to pause for 5 seconds, confirming that the query was executed successfully if the web application exhibits a delayed response.

Impact:

Data Exposure: Attackers can exploit this vulnerability to extract information from the database by using advanced SQL injection techniques.

Denial of Service (DoS): By repeatedly executing time-based payloads with increasing sleep durations, an attacker can degrade the application's performance or even cause it to become unresponsive.

Privilege Escalation: If the application uses shared database credentials across the platform, attackers might exploit this to gain higher privileges.

Mitigation Recommendations:

1**. Input Validation and Sanitization**: Ensure that all user input is properly validated and sanitized to prevent malicious SQL queries. Use parameterized queries or prepared statements to safeguard against SQL injection.

2. **Database Security**: Implement least privilege for database accounts. Ensure that the database user only has access to the necessary data and queries.

3. **Web Application Firewall (WAF):** Deploy a WAF to monitor and block suspicious SQL injection attempts.

4. Error Handling: Implement robust error handling to prevent error messages that could reveal the inner workings of the application or database to attackers.

5. **Regular Code Audits**: Perform regular security audits of the codebase to identify and patch vulnerabilities before they are exploited.

References:

OWASP SQL Injection: https://owasp.org/www-community/attacks/SQL_Injection

PHPGurukul Official Site: https://phpgurukul.com/

Conclusion:

The time-based SQL injection vulnerability in the PHPGurukul Medical Card Generation System is a critical issue that can allow attackers to disrupt the application's functionality or extract sensitive data from the database. It is recommended to address this vulnerability immediately by implementing proper input validation and using secure coding practices.