

PENETRATION TESTING REPORT

CSYE 6225 – SUMMER 2019

NAME	NUID	EMAIL ID
Kiran Kumar Kathiresan	001427809	kathiresan.k@husky.neu.edu
Nithin Muvva	001440858	muvva.n@husky.neu.edu
Akhil Prasad	001449445	prasad.ak@husky.neu.edu

1. SQL INJECTION

This kind of attack occurs when a malicious user tries to execute SQL queries to break into the web application. To prevent such attacks, the user input in the URL header must be sanitized.

Result: Used sqlmap to test the vulnerabilities with WAF the application throws HTTP status code-403 as sqlmap tries to hit many times the status code 403 is thrown 92 times. Whereas without WAF 403 error code is not thrown that means the injection was possible.

Without WAF:

```
root@kali:~# sqlmap -u 'https://nowaf.csye6225-sul9-muvvan.me/LMSApp-0.0.1-SNAPSHOT/user/register' --headers="Content-Type:application/json" --data="{\"username\" : \"muvva.n@husky.neu.edu\", \"password\" : \"radom123\"}"

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

[*] starting @ 17:01:03 /2019-08-10/

JSON data found in POST data. Do you want to process it? [Y/n/q] y
[17:01:06] [INFO] testing connection to the target URL
[17:01:07] [WARNING] the web server responded with an HTTP error code (400) which could interfere with the results of the tests
[17:01:07] [INFO] checking if the target is protected by some kind of WAF/IPS
[17:01:07] [INFO] testing if the target URL content is stable
[17:01:07] [INFO] target URL content is stable
[17:01:07] [INFO] testing if (custom) POST parameter 'JSON username' is dynamic
[17:01:08] [WARNING] (custom) POST parameter 'JSON username' does not appear to be dynamic
[17:01:08] [WARNING] heuristic (basic) test shows that (custom) POST parameter 'JSON username' might not be injectable
[17:01:08] [INFO] testing for SQL injection on (custom) POST parameter 'JSON username'
[17:01:08] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[17:01:09] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[17:01:09] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[17:01:10] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[17:01:10] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[17:01:11] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[17:01:11] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[17:01:11] [INFO] testing 'MySQL inline queries'
[17:01:11] [INFO] testing 'PostgreSQL inline queries'
[17:01:12] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[17:01:12] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
```

```

17:01:13 [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
17:01:14 [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
17:01:14 [INFO] testing 'Oracle AND time-based blind'
17:01:15 [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
17:01:21 [WARNING] (custom) POST parameter 'JSON username' does not seem to be injectable
17:01:21 [INFO] testing if (custom) POST parameter 'JSON password' is dynamic
17:01:21 [WARNING] (custom) POST parameter 'JSON password' does not appear to be dynamic
17:01:21 [WARNING] heuristic (basic) test shows that (custom) POST parameter 'JSON password' might not be injectable
17:01:21 [INFO] testing for SQL injection on (custom) POST parameter 'JSON password'
17:01:22 [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
17:01:23 [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
17:01:23 [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
17:01:23 [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
17:01:24 [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
17:01:24 [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
17:01:25 [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
17:01:25 [INFO] testing 'MySQL inline queries'
17:01:25 [INFO] testing 'PostgreSQL inline queries'
17:01:25 [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
17:01:25 [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
17:01:26 [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
17:01:26 [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
17:01:27 [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
17:01:27 [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
17:01:28 [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
17:01:28 [INFO] testing 'Oracle AND time-based blind'
17:01:29 [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
17:01:35 [WARNING] (custom) POST parameter 'JSON password' does not seem to be injectable
17:01:35 [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'
17:01:35 [WARNING] HTTP error codes detected during run:
400 (Bad Request) - 265 times


```

With WAF:

```

root@kali:~# sqlmap -u 'https://csye6225-sul9-muvvan.me/LMSApp-0.0.1-SNAPSHOT/user/register' --headers="Content-Type:application/json" --data="{\"username\" : \"muvva.n@husky.neu.edu\", \"password\" : \"123456789\"}"

```



```

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

[*] starting @ 16:25:46 /2019-08-10/

JSON data found in POST data. Do you want to process it? [Y/n/q] y
[16:26:16] [INFO] testing connection to the target URL
[16:26:18] [WARNING] the web server responded with an HTTP error code (400) which could interfere with the results of the tests
[16:26:18] [INFO] checking if the target is protected by some kind of WAF/IPS
[16:26:18] [INFO] heuristics detected web page charset 'ascii'
[16:26:18] [CRITICAL] heuristics detected that the target is protected by some kind of WAF/IPS
do you want sqlmap to try to detect backend WAF/IPS? [y/N] y
[16:26:35] [WARNING] dropping timeout to 10 seconds (i.e. '--timeout=10')
[16:26:35] [INFO] using WAF scripts to detect backend WAF/IPS protection
[16:26:36] [CRITICAL] WAF/IPS identified as 'Amazon Web Services Web Application Firewall (Amazon)'
are you sure that you want to continue with further target testing? [y/N] y
[16:26:40] [WARNING] please consider usage of tamper scripts (option '--tamper')
[16:26:40] [INFO] testing if the target URL content is stable
[16:26:41] [INFO] target URL content is stable
[16:26:41] [INFO] testing if (custom) POST parameter 'JSON username' is dynamic
[16:26:41] [WARNING] (custom) POST parameter 'JSON username' does not appear to be dynamic
[16:26:41] [WARNING] heuristic (basic) test shows that (custom) POST parameter 'JSON username' might not be injectable
[16:26:41] [INFO] testing for SQL injection on (custom) POST parameter 'JSON username'
[16:26:41] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'

```

```

[16:26:46] [INFO] testing 'Oracle stacked queries (DBMS PIPE.RECEIVE_MESSAGE - comment)'
[16:26:47] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[16:26:47] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[16:26:48] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[16:26:48] [INFO] testing 'Oracle AND time-based blind'
[16:26:49] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[16:26:55] [WARNING] (custom) POST parameter 'JSON username' does not seem to be injectable
[16:26:55] [INFO] testing if (custom) POST parameter 'JSON password' is dynamic
[16:26:55] [WARNING] (custom) POST parameter 'JSON password' does not appear to be dynamic
[16:26:55] [WARNING] heuristic (basic) test shows that (custom) POST parameter 'JSON password' might not be injectable
[16:26:55] [INFO] testing for SQL injection on (custom) POST parameter 'JSON password'
[16:26:55] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[16:26:56] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[16:26:56] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[16:26:57] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[16:26:58] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[16:26:58] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[16:26:59] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[16:26:59] [INFO] testing 'MySQL inline queries'
[16:26:59] [INFO] testing 'PostgreSQL inline queries'
[16:26:59] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[16:26:59] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[16:26:59] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[16:27:00] [INFO] testing 'Oracle stacked queries (DBMS PIPE.RECEIVE_MESSAGE - comment)'
[16:27:00] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[16:27:01] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[16:27:01] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[16:27:02] [INFO] testing 'Oracle AND time-based blind'
[16:27:02] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[16:27:09] [WARNING] (custom) POST parameter 'JSON password' does not seem to be injectable
[16:27:09] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'
[16:27:09] [WARNING] HTTP error codes detected during run:
400 (Bad Request) - 168 times, 401 (Unauthorized) - 3 times, 403 (Forbidden) - 96 times, 404 (Not Found) - 1 times

```

Blocked Requests:

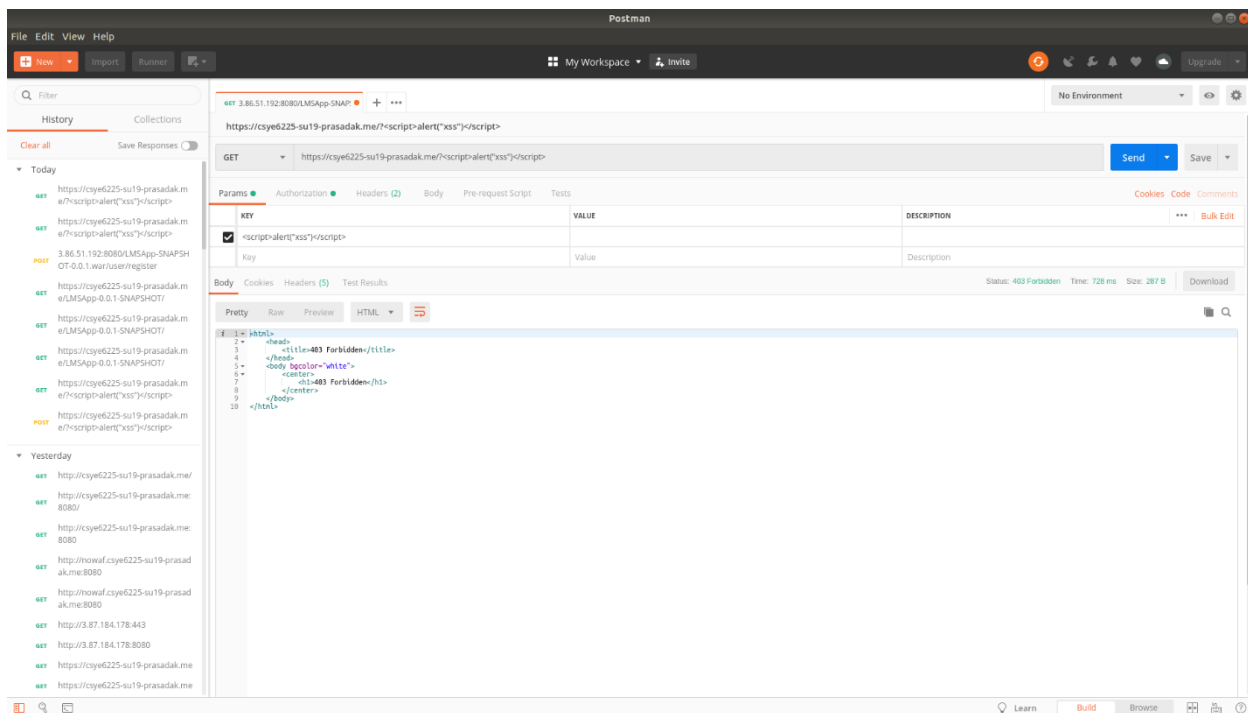
Source IP	URI	Matches rule	Action	Time (UTC)
▶ 174.63.108.71	/user/register? TBNy=5348%20AND%201%3D1%20UNION%20ALL%20SELECT%201%2CNULL%2C%27%3Cscript%3Ealert%28%22XSS%22%29%3C%2Fscript%3E%27%2Ctable_name%20FROM%20information_schema.tables%20WHERE%202%3E1--%2F%2A%2A%2F%3B%20EXEC%20xp_cmdshell%28%27cat%20..%2F..%2F%2F%2F%2F%2F%2F%2F%2F%29%23	generic-mitigate-sqli	Block	20:14:42
▶ 174.63.108.71	/user/register? id=1%20AND%201=1%20UNION%20ALL%20SELECT%201%2CNULL%2C%27%3Cscript%3Ealert%28%22XSS%22%29%3C%2Fscript%3E%27%2Ctable_name%20FROM%20information_schema.tables%20WHERE%202%3E1--%2F%2A%2A%2F%3B%20EXEC%20xp_cmdshell%28%27cat%20..%2F..%2F..	generic-mitigate-sqli	Block	20:14:52

2. CROSS SITE REQUEST FORGERY

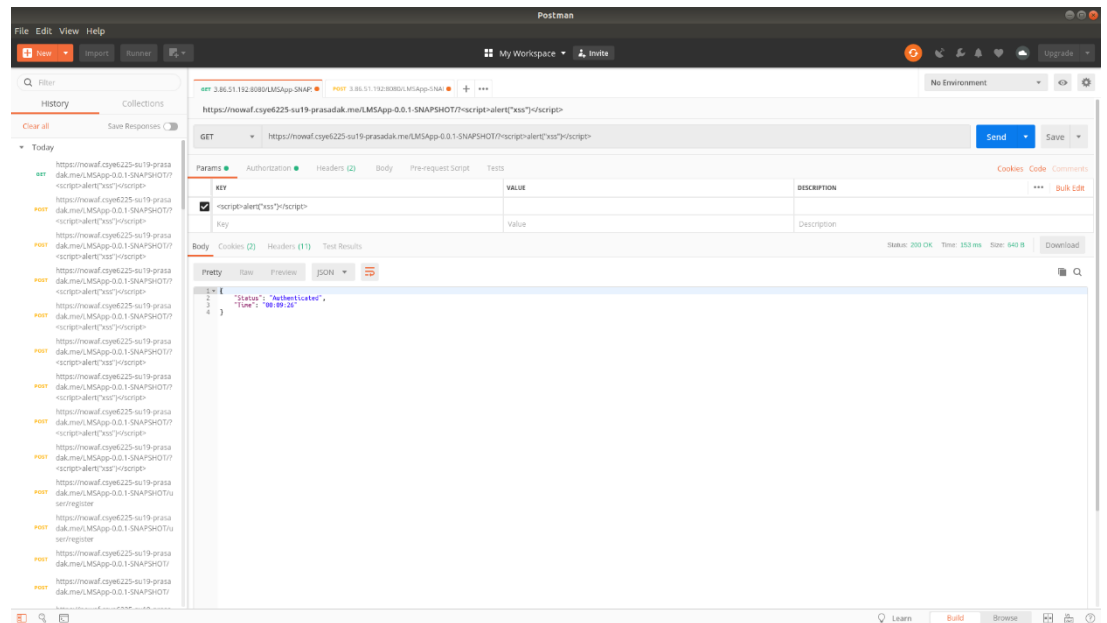
Illegitimate exposure to username and password might cause malicious people to access the database. Bcrypt eliminates this by encrypting the password to a sequence of strings with which you cannot directly access the database even if you could see the database table. Still using tokens there is a chance someone could acquire the token and do a Man in the middle attack. To prevent this we have used `csrf.disable()` in spring security and used Stateless for session creation.

Result:

With WAF:



Without
WAF:



Blocked Requests:

Sampled requests

To view new samples, choose **Get new samples**.

generic-mitigate-xss

Get new samples

Sample data from 2019-08-09 22:29:04 to 22:44:04

Source IP	URI	Matches rule	Action	Time (UTC)
▶ 174.63.108.71	/? %3Cscript%3Ealert%28%2 2xss%22%29%3C/script% 3E	generic-mitigate-xss	Block	22:38:32

3. *Insufficient Attack Protection*

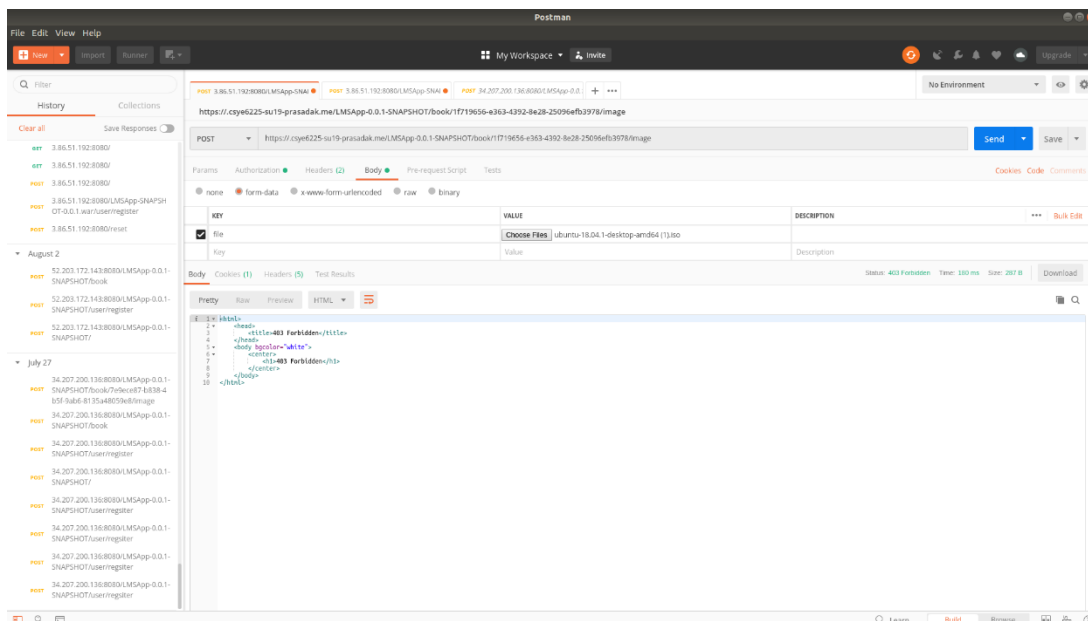
This attack involves unusual request patterns and unusual pattern in body of the request. This attack could help malicious actors to know the working flaws in the application. To prevent that we had chosen to eliminate this by using firewall.

CONDITION : Body Size

Result: When an attachment larger than the size specified is sent to the server, we get a 403 Forbidden Status Code.

With WAF:

When size of content in body is increased, gives 403 error



Without WAF:

