Penetration Testing

As the note taking web application does not have a UI, the number of possible vulnerabilities are reduced significantly. Some of these are exploited using tools in Kali Linux and other tools.

Web Application Vulnerability Mitigation In April 2017, OWASP released the new iteration of the Top 10 for public comment. The categories listed in the new proposed Top 10 are many of the same application flaw categories from the 2013 Top 10 and past versions:

A1	Injection
A2	Broken Authentication and Session Management
A3	Cross-Site Scripting (XSS)
A4	Broken Access Control (NEW)
A5	Security Misconfiguration
A6	Sensitive Data Exposure
A7	Insufficient Attack Protection (NEW)
A8	Cross-Site Request Forgery (CSRF)
A9	Using Components with Known Vulnerabilities
A10	Underprotected APIs (NEW)

We will focus on the following 3 Attack Vectors for our analysis:

1. File Size Constraint
2. SQL Injection
3. Cross Site Scripting(XSS)

Attack Vectors

1. **File Size Constraint:**

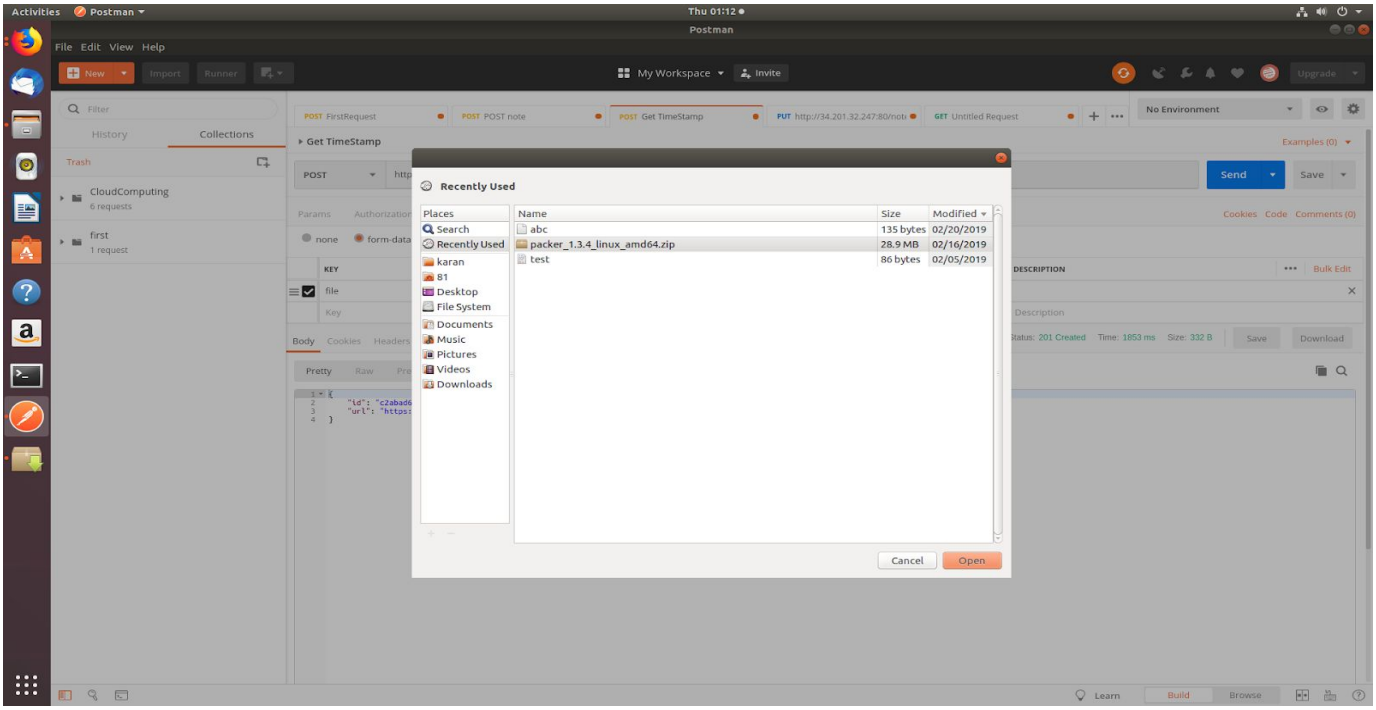
   Attack Vector: Unauthorized File Upload to S3

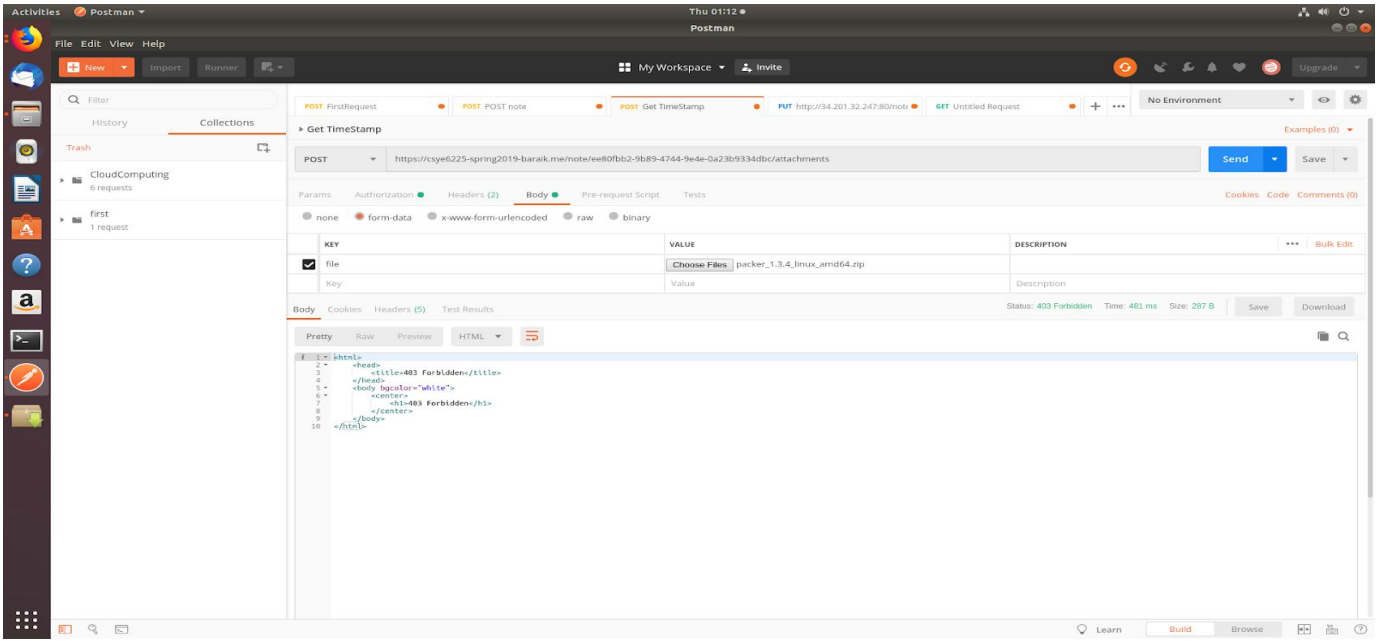
   Selecting a large file (28 MB) as an attachment to a note. This request gets blocked by WAF as it exceeds the maximum file size constraint as a result user gets 403 forbidden error.

   Why This Vector?

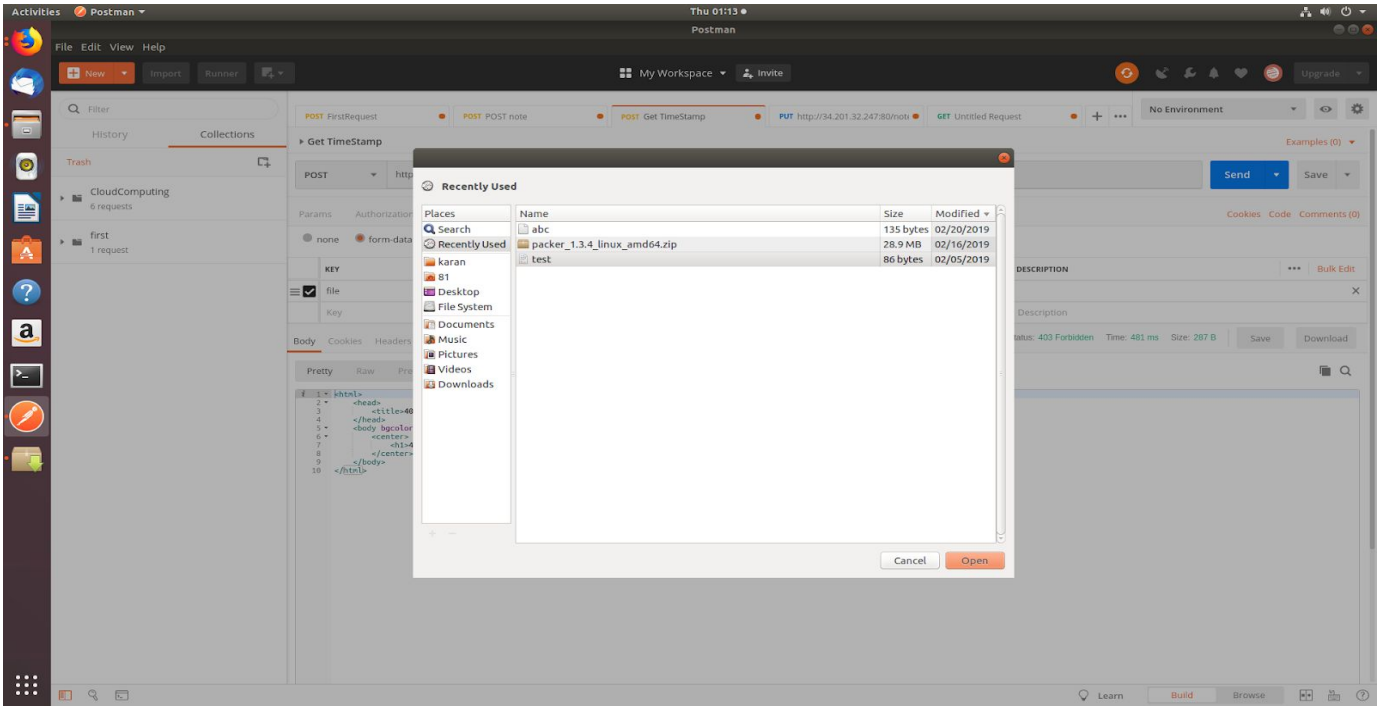
   Large file uploads can cause an delay or an outage in the network hence a check must be established in order to prevent this situation.

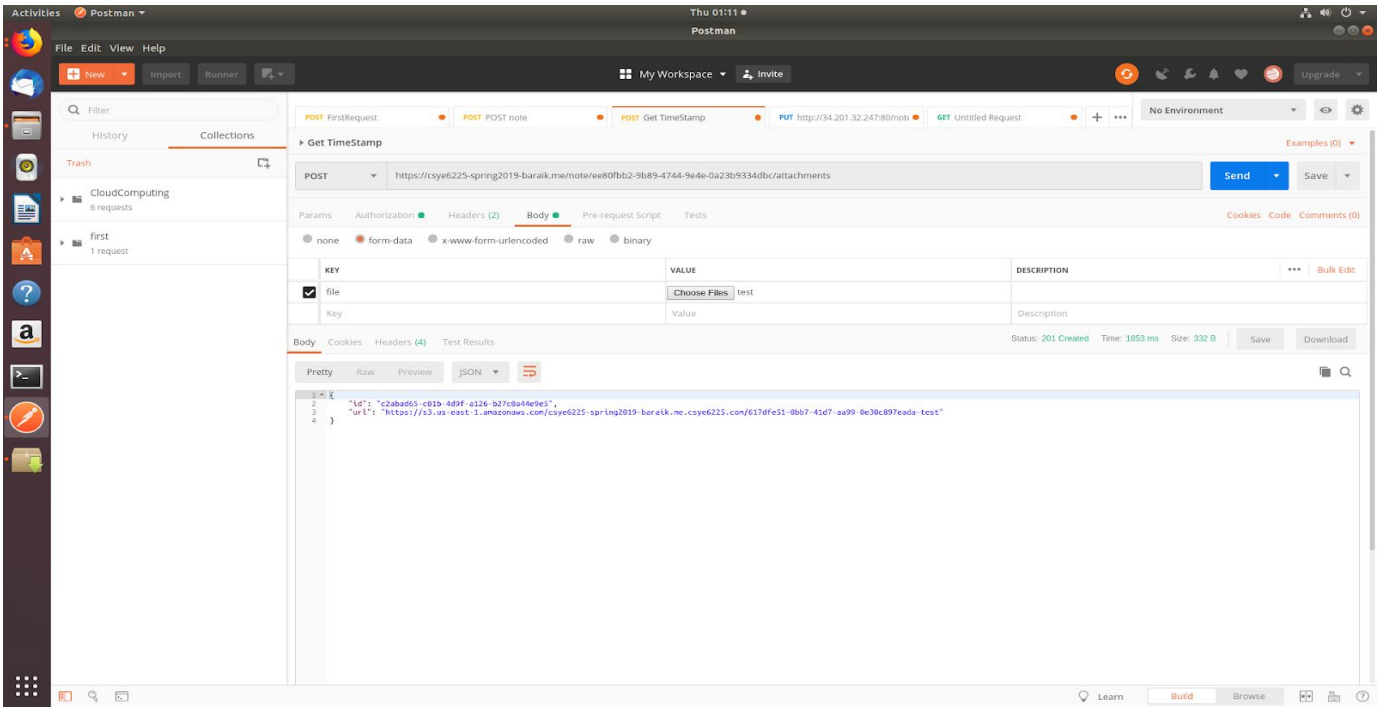
   Result:





Selecting a file of 86 bytes as an attachment to notes, WAF allows this request as file size is within the maximum file size constraint. Hence, user receives link of s3 bucket where attachment is stored and ID of the attachment.




Activities
Postman
Thu 01:11
Postman
File Edit View Help
New
Import
Runner
My Workspace
Invite
Upgrade
Filter
History
Collections
Trash
CloudComputing
6 requests
first
1 request
POST FirstRequest
POST POST note
POST Get TimeStamp
PUT http://34.201.32.247:80/not
GET Untitled Request
No Environment
Get TimeStamp
Examples (0)
POST
https://csye6225-spring2019-baraik.me/note/ee80fbb2-9b89-4744-9e4e-0a23b9334dbc/attachments
Send
Save
Params
Authorization
Headers (2)
Body
Pre-request Script
Tests
Cookies
Code
Comments (0)
none
form-data
x-www-form-urlencoded
raw
binary
KEY
VALUE
DESCRIPTION
Bulk Edit
file
Choose Files
test
Key
Value
Description
Body
Cookies
Headers (4)
Test Results
Status: 201 Created
Time: 1853 ms
Size: 332 B
Save
Download
Pretty
Raw
Preview
JSON
{
"id": "c2abad65-c01b-4d9f-a126-b27c0a44e9e5",
"url": "https://s3.us-east-1.amazonaws.com/csye6225-spring2019-baraik.me.csye6225.com/617dfe51-0bb7-41d7-aa99-0e30c897eada-test"
}
Learn
Build
Browse

2. **SQL Injection:**

Attack Vector: Injection

It is a code injection technique for data driven applications in which we test the username and password authentication while in place of username or password, when 1=1(always true) is given, the code, which is vulnerable, exposes the result even without being authenticated and authorized.

Why This Vector?

Attackers can access the database even without knowing the actual passwords and other parameters, keeping us unknown to this fact. Since the webapp has excessive usage of SQL involving user, notes and attachment tables, it is logical and useful to test the code for SQL Injection.

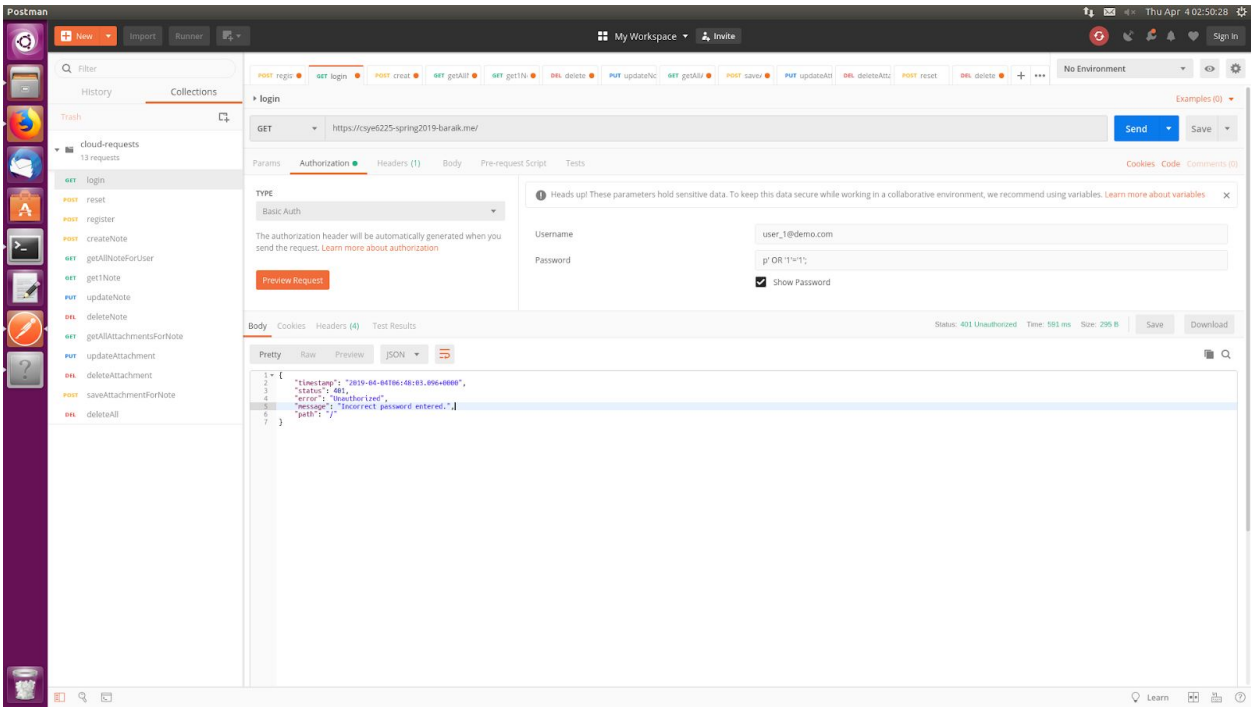
Tested on Server with WAF:

```
root@kali:~# sqlmap -u https://csye6225-spring2019-dalalvi.me/user/register --data "
{\r\n\t\"emailId\": \"dalalvivek007@gmail.com\",\r\n\t\"password\": \"Password123@\"\r\n}"

        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.3#stable}
|_ -| . ["]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's resp

[*] starting @ 04:04:38 /2019-04-04/

JSON data found in POST data. Do you want to process it? [Y/n/q] Y
[04:04:41] [INFO] testing connection to the target URL
[04:04:41] [INFO] heuristics detected web page charset 'ascii'
[04:04:41] [WARNING] the web server responded with an HTTP error code (403) which could interfere with the results of the test
[04:04:41] [CRITICAL] previous heuristics detected that the target is protected by some kind of WAF/IPS
[04:04:41] [INFO] testing if the target URL content is stable
[04:04:42] [INFO] target URL content is stable
[04:04:42] [INFO] testing if (custom) POST parameter 'JSON emailId' is dynamic
[04:04:42] [WARNING] (custom) POST parameter 'JSON emailId' does not appear to be dynamic
[04:04:42] [WARNING] heuristic (basic) test shows that (custom) POST parameter 'JSON emailId' might not be injectable
[04:04:42] [INFO] testing for SQL injection on (custom) POST parameter 'JSON emailId'
[04:04:42] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[04:04:44] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[04:04:44] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[04:04:44] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[04:04:45] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[04:04:45] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[04:04:46] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[04:04:46] [INFO] testing 'MySQL inline queries'
[04:05:01] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[04:05:02] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[04:05:02] [INFO] testing 'MySQL inline queries'
[04:05:02] [INFO] testing 'PostgreSQL inline queries'
[04:05:02] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[04:05:02] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[04:05:03] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[04:05:03] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[04:05:04] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[04:05:04] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[04:05:05] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[04:05:05] [INFO] testing 'Oracle AND time-based blind'
[04:05:06] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[04:05:13] [WARNING] (custom) POST parameter 'JSON password' does not seem to be injectable
[04:05:13] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--
[04:05:13] [WARNING] HTTP error codes detected during run:
403 (Forbidden) - 264 times

[*] ending @ 04:05:13 /2019-04-04/
```

Tested on Server without WAF:

```
root@kali:~# sqlmap -u https://csye6225-spring2019-davdag.me/user/register --data "
{\r\n\t\"emailId\": \"dalalvivek007@gmail.com\",\r\n\t\"password\": \"Password123@\"\r\n}"

        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.3#stable}
|_ -| . ['] |     | .'| . |
|___|_  ['] |_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end

[*] starting @ 03:59:13 /2019-04-04/

JSON data found in POST data. Do you want to process it? [Y/n/q] Y
[03:59:18] [INFO] testing connection to the target URL
[03:59:19] [WARNING] the web server responded with an HTTP error code (400) which could interfere with the results
[03:59:19] [INFO] testing if the target URL content is stable
[03:59:19] [WARNING] target URL content is not stable (i.e. content differs). sqlmap will base the page comparison
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] C
[03:59:23] [INFO] testing if (custom) POST parameter 'JSON emailId' is dynamic
[03:59:23] [WARNING] (custom) POST parameter 'JSON emailId' does not appear to be dynamic
[03:59:23] [WARNING] heuristic (basic) test shows that (custom) POST parameter 'JSON emailId' might not be injectal
[03:59:24] [INFO] testing for SQL injection on (custom) POST parameter 'JSON emailId'
[03:59:24] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[03:59:26] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[03:59:26] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[03:59:27] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[03:59:47] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'
[03:59:47] [INFO] testing 'MySQL inline queries'
[03:59:47] [INFO] testing 'PostgreSQL inline queries'
[03:59:47] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[03:59:47] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[03:59:48] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[03:59:48] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[03:59:49] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[03:59:49] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[03:59:50] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[03:59:51] [INFO] testing 'Oracle AND time-based blind'
[03:59:52] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[03:59:59] [WARNING] (custom) POST parameter 'JSON password' does not seem to be injectable
[03:59:59] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values f
[03:59:59] [WARNING] HTTP error codes detected during run:
400 (Bad Request) - 276 times

[*] ending @ 03:59:59 /2019-04-04/
```

Testing with 1=1:



Result:

SQLInjection was tried, both, manually and using SQLmap tool. For the server with WAF, it gave 403 Forbidden always. For the server without WAF, it reached the webapp but still gave 400 Bad Request because the password stored in database is not in plain text but BCrypted using SALT. This is also verified by performing SQLInjection manually using postman where password had 1=1 which BCrypts this password and raises error of incorrect password.

3. **Cross Site Scripting (XSS):**

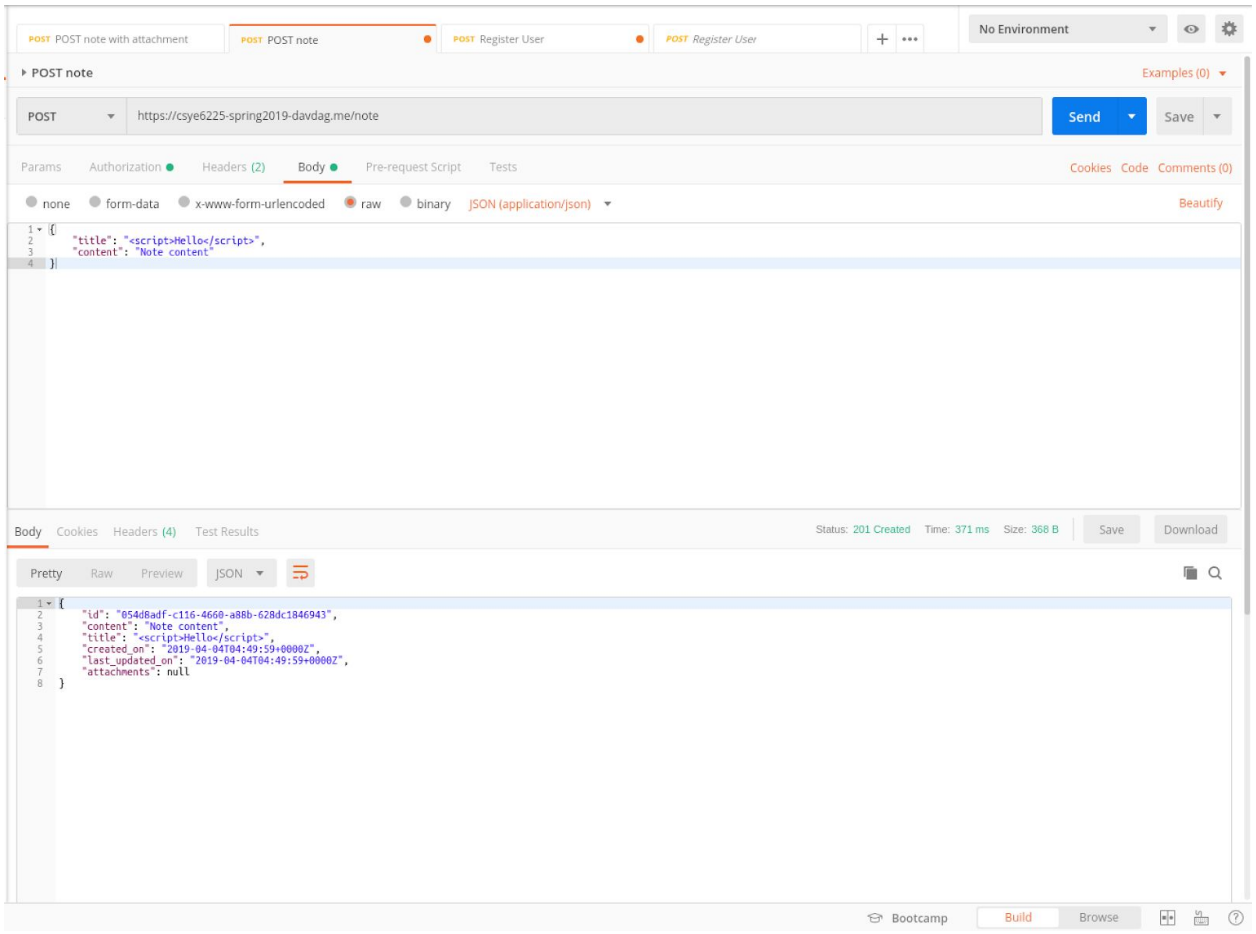
Attack Vector: Cross Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. In case of REST APIs, it can be any form of script which can be run on the server side.

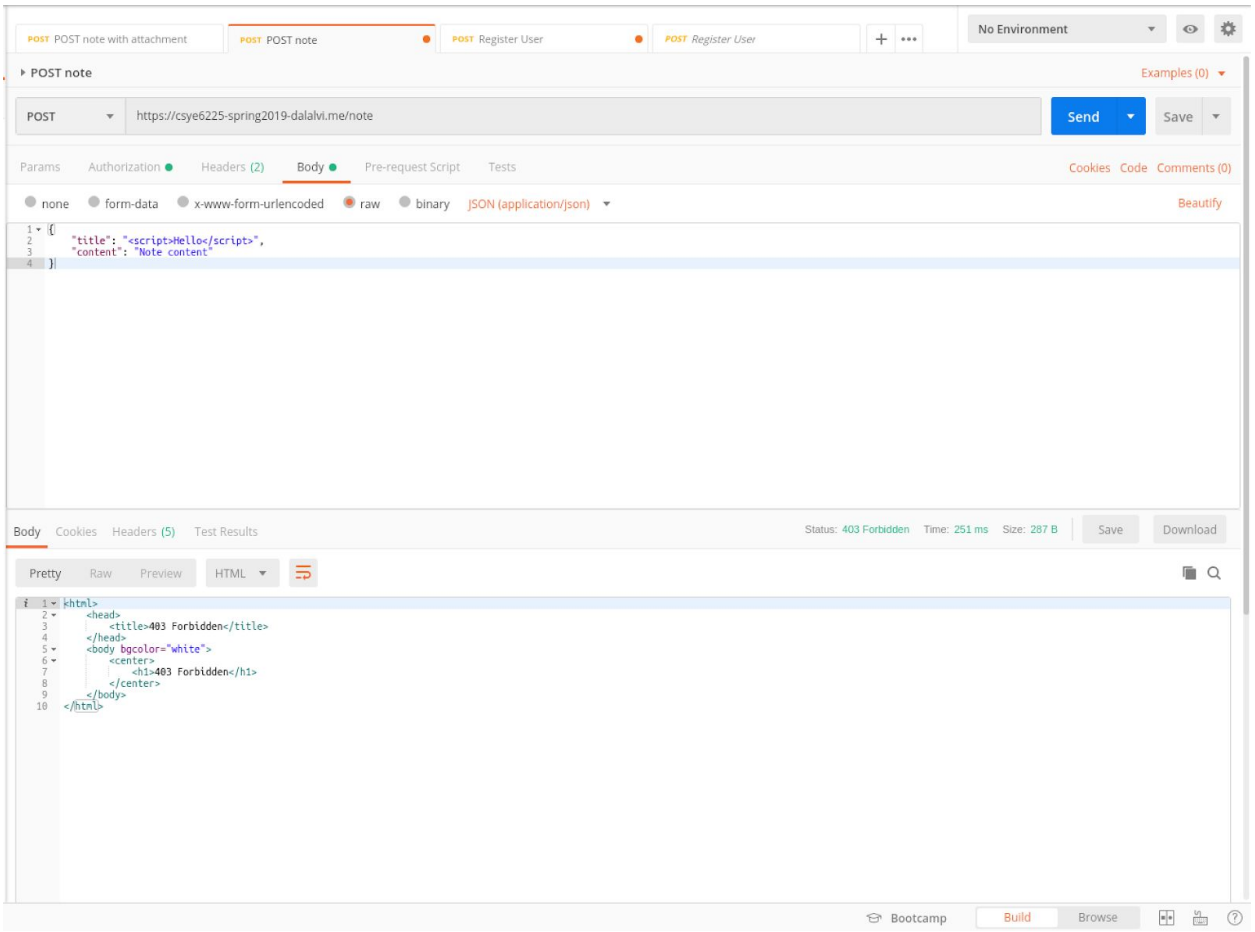
Why This Vector?

Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it. With REST APIs, this becomes risky when the attacker is successful in sending code snippets which run on the server side without the knowledge of the site owner. Such issues can be very dangerous and a WAF like the one provided by AWS, can mitigate these issues by restricting requests with payload containing any kind of scripts in it.

Without WAF results:



With WAF results:



Result:

Cross Site Scripting was tried manually using Postman. For the server with WAF, it gave 403 Forbidden always. For the server without WAF, the request was processed successfully. Hence, we successfully mitigated the XSS attack vector using WAF.

Note on CSRF:

Since we have REST endpoints which are stateless and there is no cookie information which the application accesses or requires, we have allowed CSRF rules to be bypassed. In order to handle the CSRF rules, we will have to explicitly configure that in the application by using Spring security feature.