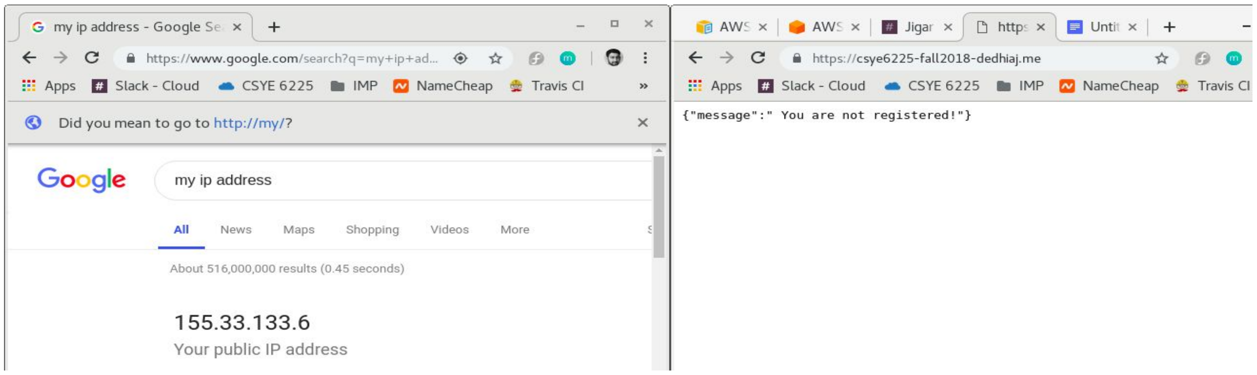
Report : Attacks on web application with and without the AWS WAF

Attack Vector 1: Blacklisting unauthorized IP addresses

Reason for choosing this specific attack vector:
There are always a few blacklisted IP-addresses which are blocked for spamming resources. We need to prevent this IP addresses from accessing our resources and avoiding misuse. Before installing Web Application Firewall, all the IP addresses were able to access our web application and utilize all our resources.



```
{"message":" You are not registered!"}
```

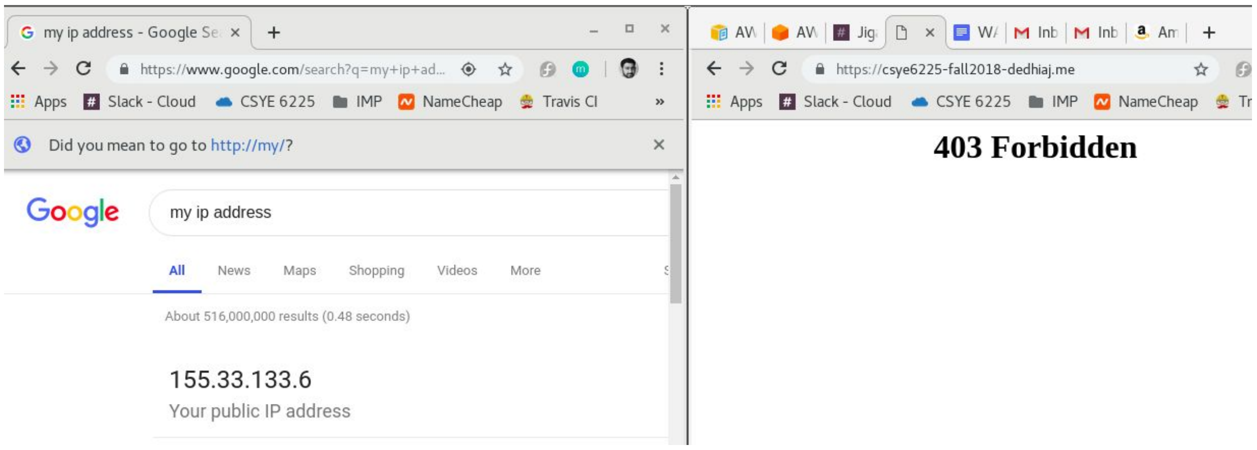
About 516,000,000 results (0.45 seconds)

155.33.133.6
Your public IP address

my ip address - Google Se
https://www.google.com/search?q=my+ip+ad...
Apps
Slack - Cloud
CSYE 6225
IMP
NameCheap
Travis CI
Did you mean to go to http://my/?
Google
my ip address
All
News
Maps
Shopping
Videos
More
https://csye6225-fall2018-dedhiaj.me

Testing:
Try to hit the application from your IP addresses via the domain name of the application.
For eg: https://csye6225-fall2018-dedhiaj.me/
If the application returns a message in the form of json, your IP address is not in the blocked IP address list. But, if the application returns a "403 Forbidden" message, your IP has been blocked by the server.

Result:
Setting up Web Application Firewall, helped us to avoid all the unauthorized IP addresses from access our application and exploiting our resources.



403 Forbidden

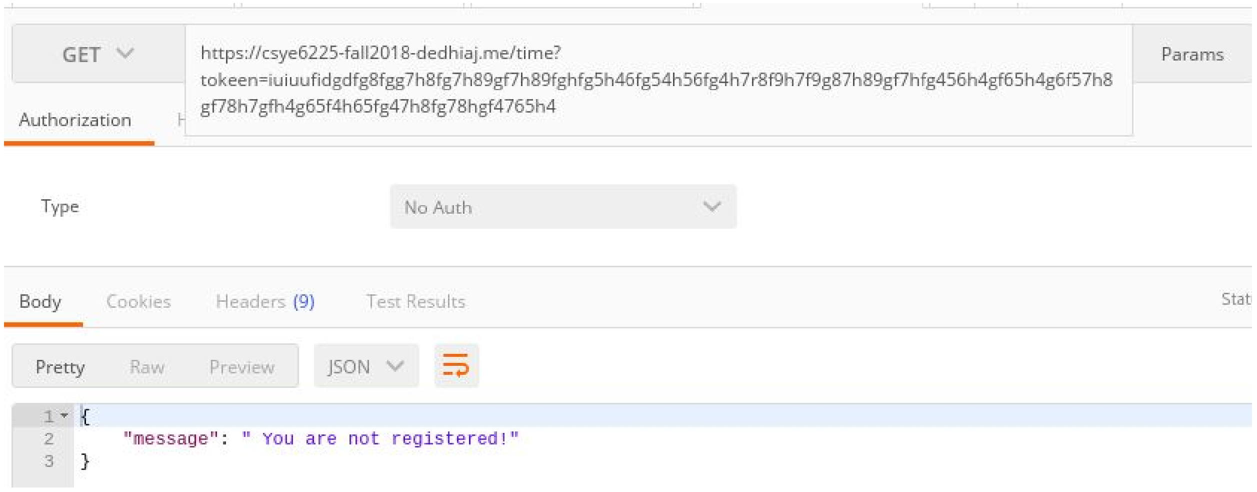
About 516,000,000 results (0.48 seconds)

155.33.133.6
Your public IP address

my ip address - Google Se
https://www.google.com/search?q=my+ip+ad...
Apps
Slack - Cloud
CSYE 6225
IMP
NameCheap
Travis CI
Did you mean to go to http://my/?
Google
my ip address
All
News
Maps
Shopping
Videos
More
https://csye6225-fall2018-dedhiaj.me

<u>Attack Vector 2</u>: Size constraints on Requested URL

<u>Reason for choosing this specific attack vector:</u>
There might be cases where some hackers might try to break your application by hitting/sending a long URL request to your application in order to get the insights of your application structure. We need to protect our web application from such access requests and hide our server directory structure. For instance, in the below request, the token parameter has been passed which is very big and not an accepted url request parameter by our application.

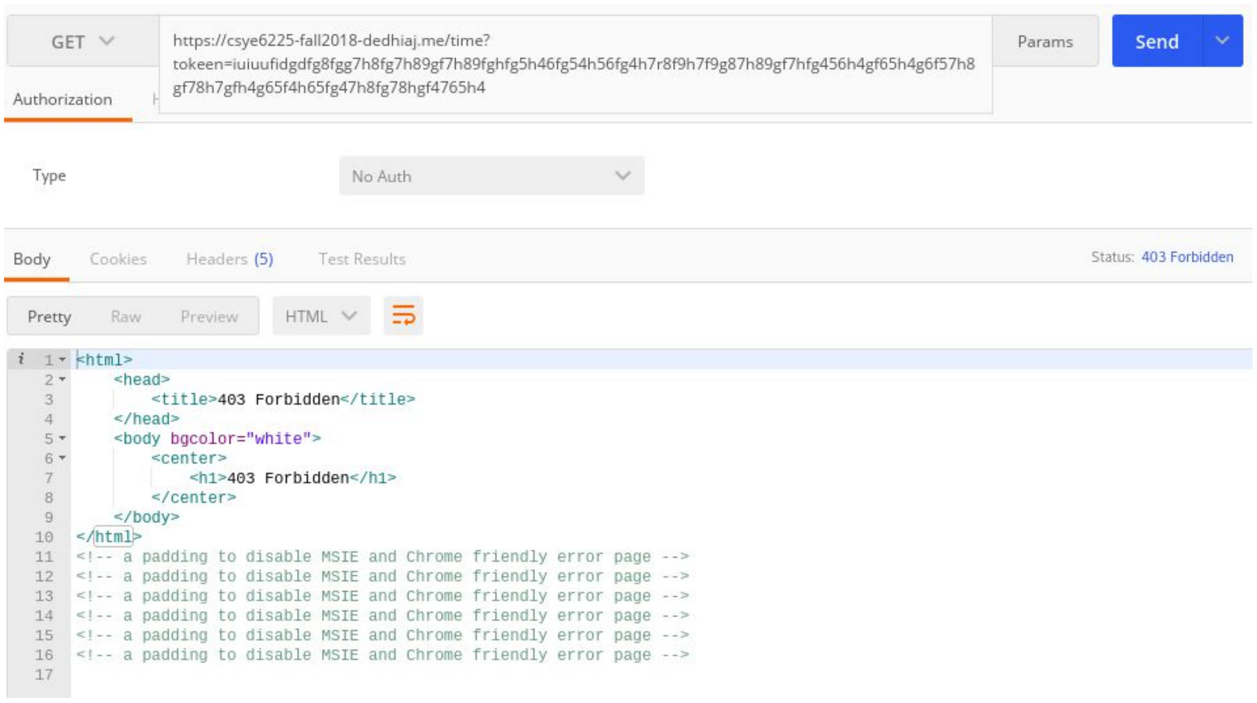


GET https://csye6225-fall2018-dedhiaj.me/time?tokeen=iuiuufidgdfg8fgg7h8fg7h89gf7h89fghfg5h46fg54h56fg4h7r8f9h7f9g87h89gf7hfg456h4gf65h4g6f57h8gf78h7gfh4g65f4h65fg47h8fg78hgf4765h4

Type: No Auth

```
{
    "message": " You are not registered!"
}
```

<u>Result:</u> After setting up Web Application Firewall, we do not allow the request of this sort to reach the web application and return the request from the firewall itself. This helps us make sure that our server folder structure is intact and not accessed by anyone.



GET https://csye6225-fall2018-dedhiaj.me/time?tokeen=iuiuufidgdfg8fgg7h8fg7h89gf7h89fghfg5h46fg54h56fg4h7r8f9h7f9g87h89gf7hfg456h4gf65h4g6f57h8gf78h7gfh4g65f4h65fg47h8fg78hgf4765h4

Type: No Auth

Status: 403 Forbidden

```
<html>
    <head>
        <title>403 Forbidden</title>
    </head>
    <body bgcolor="white">
        <center>
            <h1>403 Forbidden</h1>
        </center>
    </body>
</html>
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
```

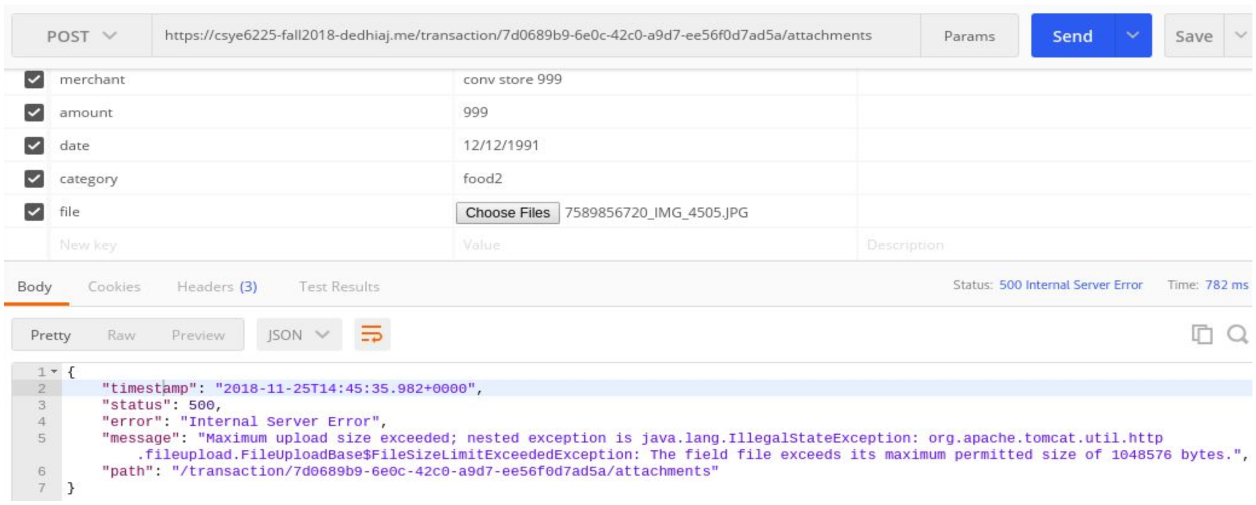
Attack Vector 3: Size constraints on uploading large file as attachments

Reason for choosing this specific attack vector:

Sometimes users are unaware of the file size limit while uploading a particular attachment. Also, sometimes they might attach a different format of document and that might affect the performance of our application. This might result in lack of performance and impact the credibility of the application. Thus we need to stop user from allowing to upload the file from firewall level and not reach the application to degrade it's performance. We have handled this by considering the request body size restrictions.

Testing:

We are trying to upload a file of 6MB for our attachment upload. It gives us the file-size error after reaching the application.



https://csye6225-fall2018-dedhiaj.me/transaction/7d0689b9-6e0c-42c0-a9d7-ee56f0d7ad5a/attachments

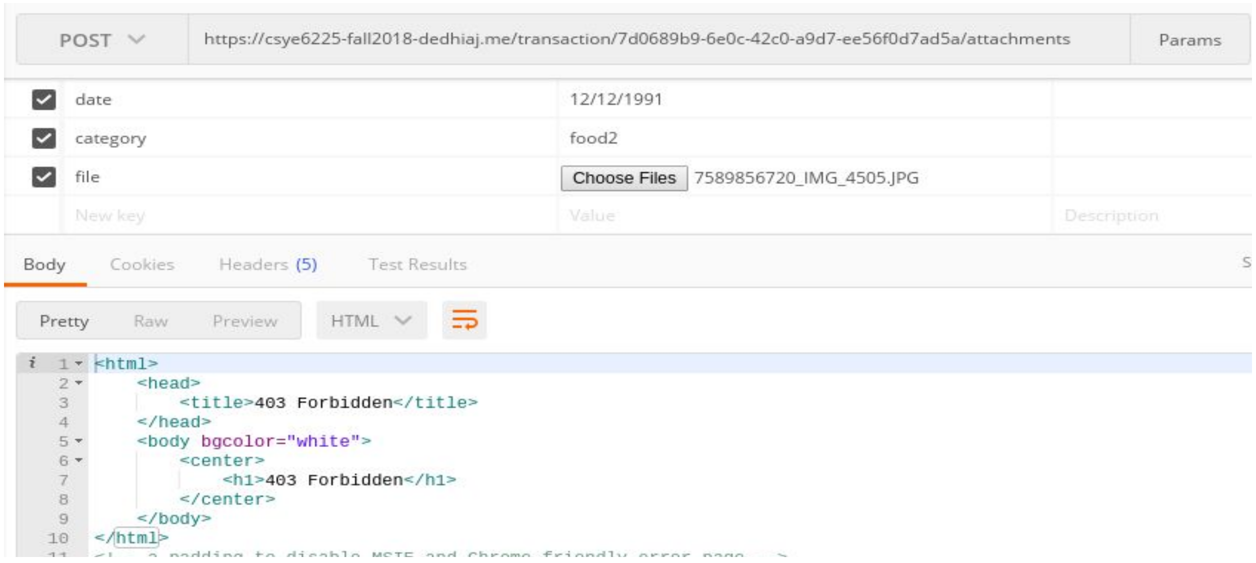
✓	merchant	conv store 999
✓	amount	999
✓	date	12/12/1991
✓	category	food2
✓	file	Choose Files 7589856720_IMG_4505.JPG
	New key	Value

Status: 500 Internal Server Error Time: 782 ms

```
{
    "timestamp": "2018-11-25T14:45:35.982+0000",
    "status": 500,
    "error": "Internal Server Error",
    "message": "Maximum upload size exceeded; nested exception is java.lang.IllegalStateException: org.apache.tomcat.util.http
       .fileupload.FileUploadBase$FileSizeLimitExceededException: The field file exceeds its maximum permitted size of 1048576 bytes.",
    "path": "/transaction/7d0689b9-6e0c-42c0-a9d7-ee56f0d7ad5a/attachments"
}
```

Result:

After setting up the Web Application Firewall, the firewall checks on the request body size and restricts user from uploading big files before letting the request reach the application.



https://csye6225-fall2018-dedhiaj.me/transaction/7d0689b9-6e0c-42c0-a9d7-ee56f0d7ad5a/attachments

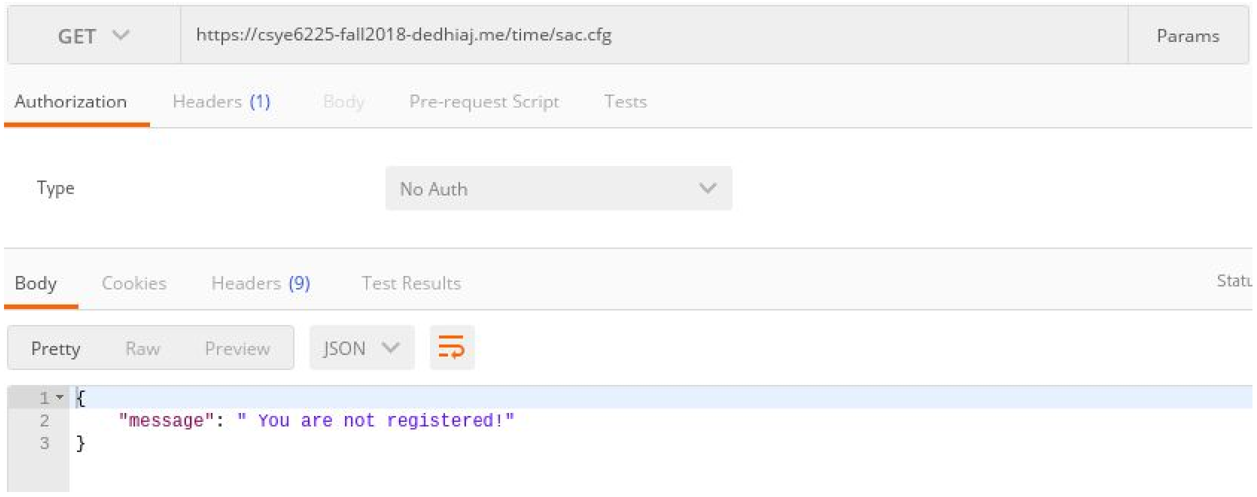
✓	date	12/12/1991
✓	category	food2
✓	file	Choose Files 7589856720_IMG_4505.JPG
	New key	Value

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

Attack Vector 4: Constraints on Server Side File Access

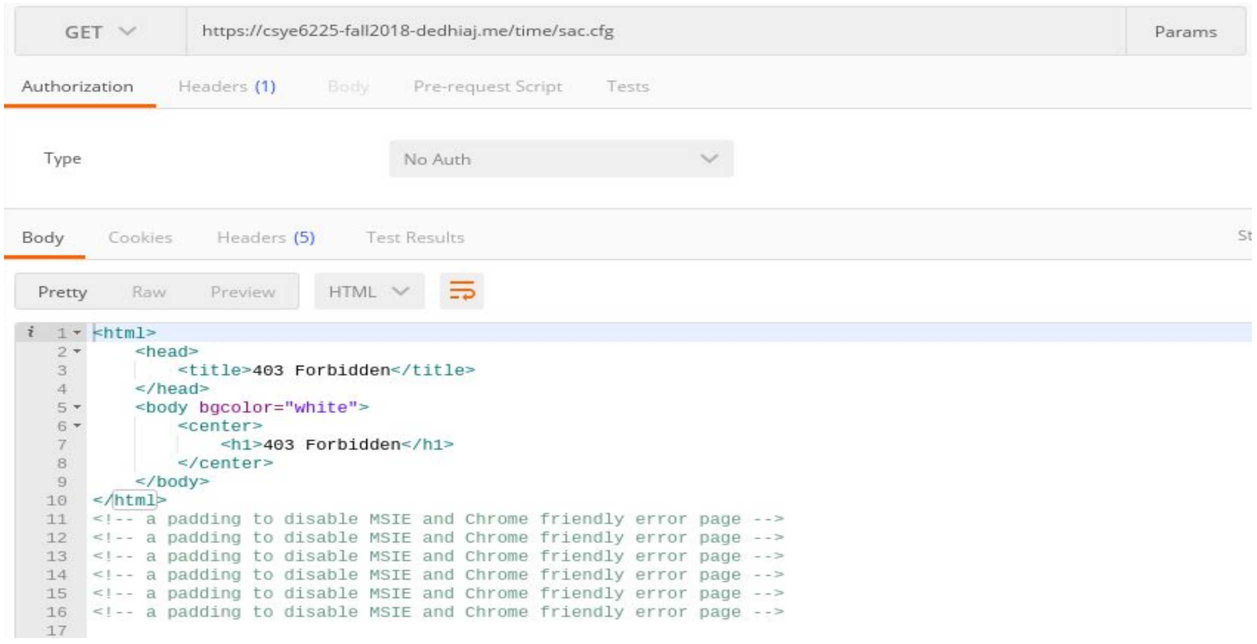
Reason for choosing this specific attack vector:

Sometimes hacker come to know the path of folder structure on server side and try to access the files and directories available directly. We might want to control this access as these file may consist of configuration data, setup data, secure data or logs which we might be want to expose to end user. Thus we need to stop user from allowing access to server side files from firewall level. We have handled this by considering the request body size restrictions.



```
{
    "message": " You are not registered!"
}
```

Result:



```
<html>
    <head>
        <title>403 Forbidden</title>
    </head>
    <body bgcolor="white">
        <center>
            <h1>403 Forbidden</h1>
        </center>
    </body>
</html>
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
<!-- a padding to disable MSIE and Chrome friendly error page -->
```

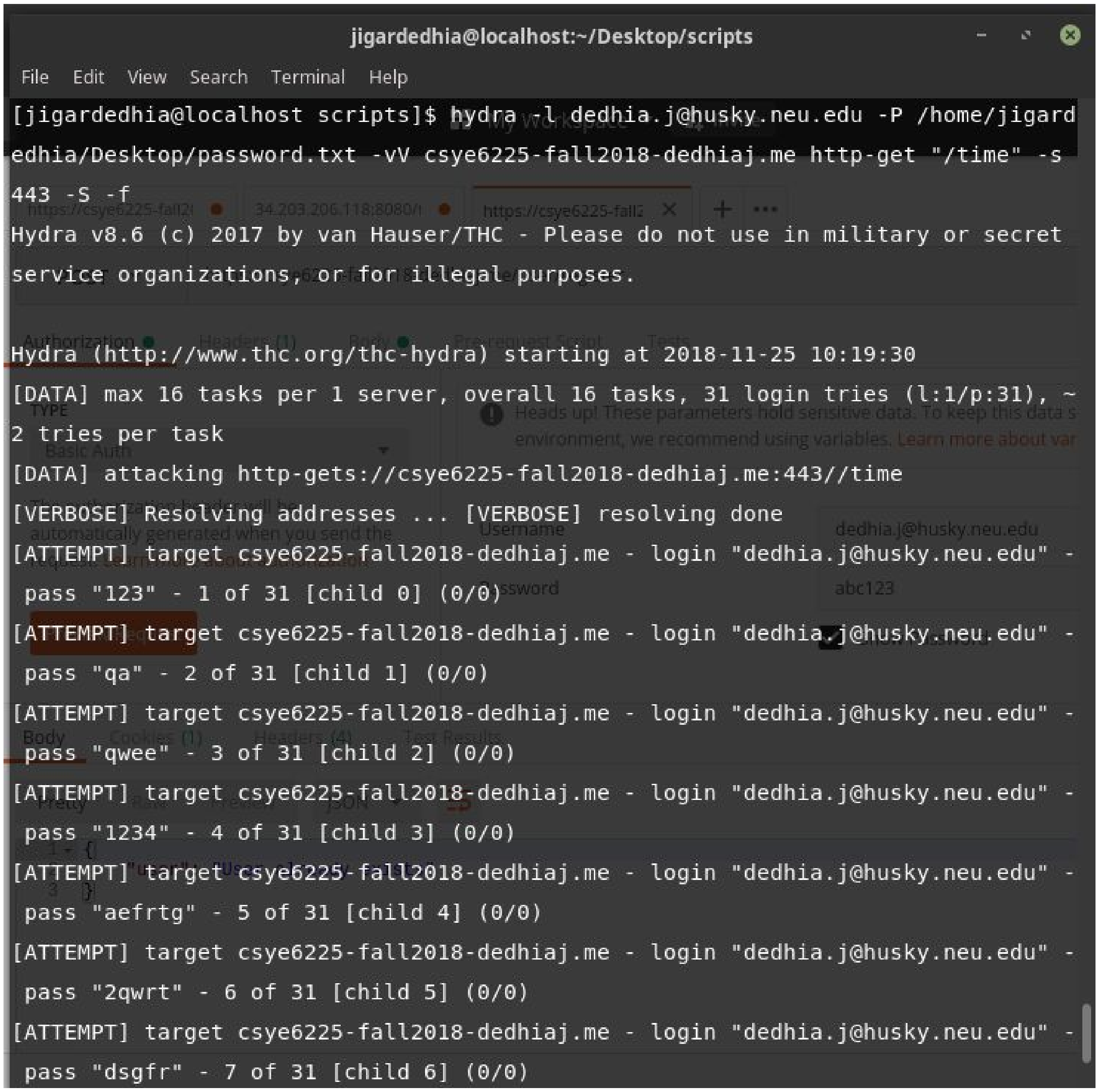
Attack Vector 5: Constraints on huge number of requests (HTTP flood)

Reason for choosing this specific attack vector:
Hacker might want to identify sensitive user informations for instance password, credit card details etc. or try to slow down the server by sending multiple brute force requests. We would like to avoid this at directly to protect sensitive data and prevent performance impact on our application.

Testing:
We create a hydra request/attack at command line level to find out the password for a particular user-id using bruteforce method. We have a list of passwords stored and would try to test those passwords to identify a particular user's password.
The request format is as follows:

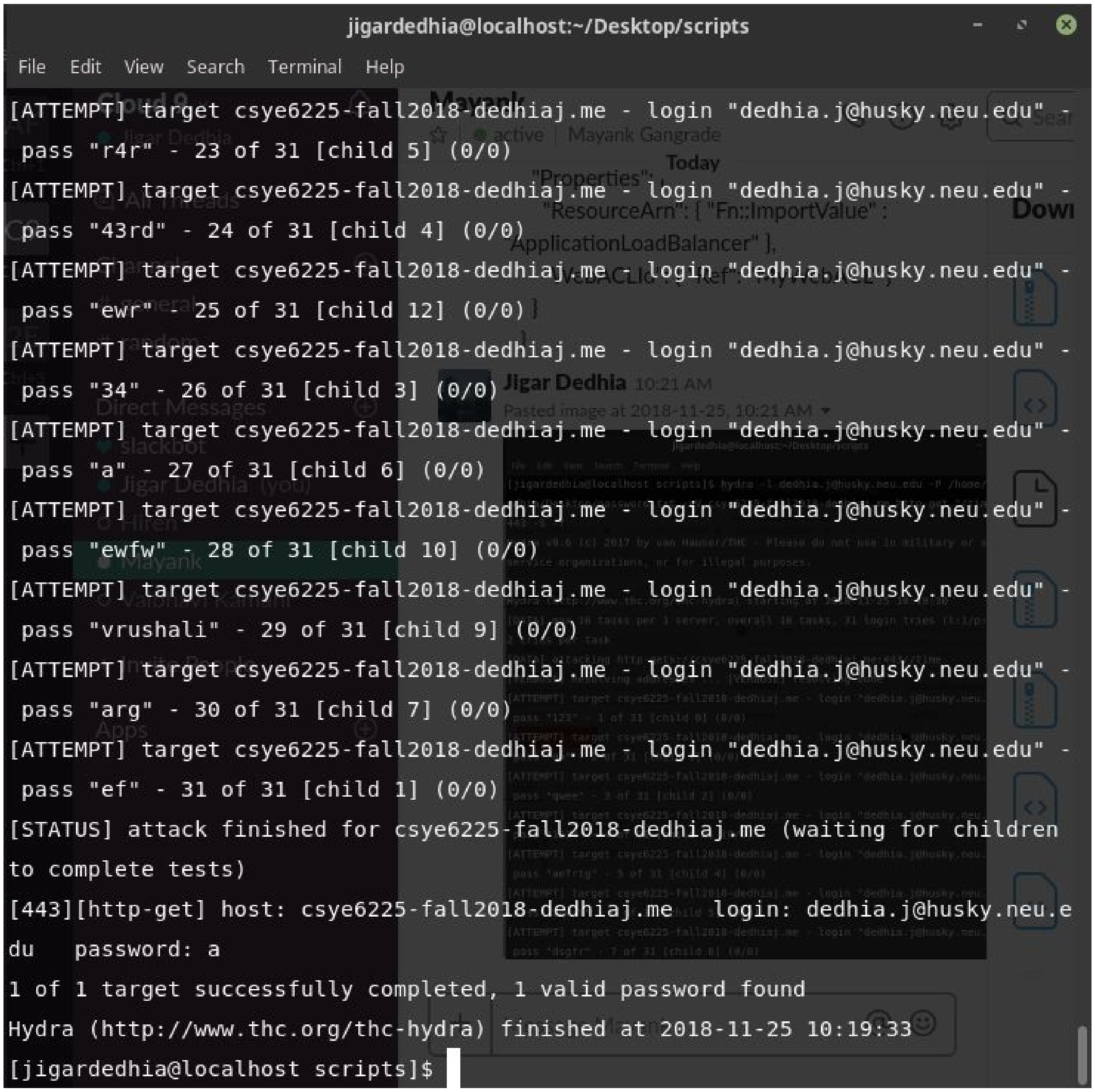
```
hydra -l abcdef@gmail.com -P password.txt -vV <domain name of the application> http-get "<api url>" -s 443 -S -f
```

-f → tells hydra to stop on first valid password it finds
-P → tells us that the file provided in the next part is the list of passwords to be tested
-s → asks you for the port number you want to run a request on
-S → tells you that the request is a secured request
-l → tells you the username you want to run the bruteforce for
http-get →shows the method you would access the url with
-v → verbose


jigardedhia@localhost:~/Desktop/scripts
File Edit View Search Terminal Help

```
[jigardedhia@localhost scripts]$ hydra -l dedhia.j@husky.neu.edu -P /home/jigard
edhia/Desktop/password.txt -vV csye6225-fall2018-dedhiaj.me http-get "/time" -s
443 -S -f
Hydra v8.6 (c) 2017 by van Hauser/THC - Please do not use in military or secret
service organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2018-11-25 10:19:30
[DATA] max 16 tasks per 1 server, overall 16 tasks, 31 login tries (l:1/p:31), ~
2 tries per task
[DATA] attacking http-gets://csye6225-fall2018-dedhiaj.me:443//time
[VERBOSE] Resolving addresses ... [VERBOSE] resolving done
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "123" - 1 of 31 [child 0] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "qa" - 2 of 31 [child 1] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "qwee" - 3 of 31 [child 2] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "1234" - 4 of 31 [child 3] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "aefrtg" - 5 of 31 [child 4] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "2qwrt" - 6 of 31 [child 5] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "dsgfr" - 7 of 31 [child 6] (0/0)
```


jigardedhia@localhost:~/Desktop/scripts
File Edit View Search Terminal Help

```
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "r4r" - 23 of 31 [child 5] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "43rd" - 24 of 31 [child 4] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "ewr" - 25 of 31 [child 12] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "34" - 26 of 31 [child 3] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "a" - 27 of 31 [child 6] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "ewfw" - 28 of 31 [child 10] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "vrushali" - 29 of 31 [child 9] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "arg" - 30 of 31 [child 7] (0/0)
[ATTEMPT] target csye6225-fall2018-dedhiaj.me - login "dedhia.j@husky.neu.edu" -
 pass "ef" - 31 of 31 [child 1] (0/0)
[STATUS] attack finished for csye6225-fall2018-dedhiaj.me (waiting for children
to complete tests)
[443][http-get] host: csye6225-fall2018-dedhiaj.me   login: dedhia.j@husky.neu.e
du   password: a
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2018-11-25 10:19:33
[jigardedhia@localhost scripts]$
```

Result:

We can avoid this detection of passwords or even the bruteforce attack, by setting up the Web Application Firewall to block the IP address if it tries to hit the server multiple times within a certain period of time. We can do this by adding a few threshold parameters and creating a lambda function to manage the IP address access and the threshold values. If the threshold values are crossed, lambda function adds the IP address to the blacklisted IP address list.