CSYE 6225
Spring 2019
Penetration Testing

Submitted by:

Nishita Sikka 001816375
Sunil Yadav 001492711
Murtaza Haji 001830090
Saurabh Kulkarni 001858169

Assignment Guidelines for Penetration Testing:

Penetration Testing

- Identify and test your application against at least 3 attack vectors that do not exploit UI vulnerabilities. - You will document your findings in a PDF (Google doc exported as PDF) and commit it to your GitHub repository. - Your report should be as detailed as possible.
- **You will document attacks on your own web application with and without the AWS WAF in place.**

Your report should provide details on following along with screenshots:

1. Attack Vector
2. Result
3. Why did you choose this specific attack vector?

What is Penetration Testing?

Also known as pen testing or maybe ethical hacking, penetration testing of a web application is mostly a practice to find security vulnerabilities that could be exploited by an attacker. The pen test could be an automated testing procedure for having more information regarding the weak spots in an application. So, it is actually a white hat attack because the good guys are trying to break in to find the weak spots in the applications further avoiding future cybersecurity hacks.

The framework we are going to use is Kali Linux.

What is an Attack Vector?

Attack vectors basically are the enablers for hackers to be able to exploit vulnerabilities in a system which is way to get access to a web application or any network to attempt malicious activities.

1. **A1: Injection** – (Using Kali Linux/WAF)

Attack vector:

The most known injection flaws are the SQL injection flaws which occurs due to un-sanitized data value requests to the interpreter. Often. An injection flaw occurs when any unknown host application sends untrusted data to an interpreter by the use of which an attacker can easily alter the contents of the requests/triggers and cause dangerous consequences.

Our web application is supported by Postgres database.

Logging into our database named **"public"**

public=# select * from users;

```
public=# select * from users;
 id |           email            |                           password
----+----------------------------+--------------------------------------------------------------
  1 | nishita@gmail.com          | $2b$10$zfxJGyyC6yQGYIDmzBCdwesbzK9er65OHDhZeqMeON324NEQGJBxC
  2 | nia@gmail.com              | $2b$10$a8VHe0qSmtE4gRWYulUgZ.wqnGxNsfEy0eFgDV0AJgpfDLKzNE9bG
  3 | nishitasikka12@gmail.com   | $2b$10$0JY4Jksp4cmndtIK/5MqteJQxNI8LMqmknIL/p613KZJn.ZhWb3Ku
  4 | aaaaikka12@gmail.com       | $2b$10$NzrutFRxk5zQ1aw58VmH4.aBejG2vQBC76gPlyZHK2v6o5VKSljsu
  5 | aaikka12@gmail.com         | $2b$10$31S7BJ75oAOxlFCk2eHAEOEmm7XRLvLPImq0braTaRlKF/ohrPvKa
  6 |                            | $2b$10$zjUq3ZI9NsMUCFwi3RhitetsO8DuWZUqffoQCI8M0u9BLJvTKTgnm
  7 | dnn@gmail.com              | $2b$10$7LCHm837Q964ITW.bghnAe2GhHs3cKOOIbtBWvDzssQ7TE4LwNPtm
  8 | nishitasikka2695@gmail.com | $2b$10$.PoafGawtJ3YuOdhhigV8.mp6K/XLBcfYVBctWWeYrXUtOwAWob6u
  9 | nishitasikka26@mail.com    | $2b$10$uL.i9C0yccdOCa/gTtC4cuGMm2BmjJDlL8YTZO9RgYUsE/TY5Vz5C
 10 | nishi@mail.com             | $2b$10$pWJZ.VwlyMhKaB7IaPURgeQqUwAaLCkuj34wde8kqEwMEYXvG0Yf.
 11 | code@gmail.com             | $2b$10$IfOPCjeyp8N2KyqK/5nmeO/.fZxlbAOX5FjzF8SEUsp69hS9hu0RK
 12 | review@gmail.com           | $2b$10$ipUg3RbFSTY/MDk8STS3BOIFvB3TLQpPJ/gQ8/3XObFPggEW7NsFe
 13 | ashish@code.com            | $2b$10$Ox0IJ2zSzhIPnsdC/72EXulvsRxVszC3p68/hRWZ3ZmKk.CU0VK3e
 14 | nishita@code.com           | $2b$10$eQ/.DVWAfat5aF9mwaubguWrTdUIX4r0KfXdHVXd/IXboGKGXsrIG
 15 | test@code.com              | $2b$10$Qe0N.S0eU9Sgap/l..7bseWzKM6rWlTMiGI5QEyszoYe4D/JriJB6
 16 | f@f.com                    | $2b$10$OBeclxfBTvusS4cw.JJx0.BIoym1zIxuN6ApJPgGBiAiWhG.gq3ua
 17 | hey@hey.com                | $2b$10$xS9KKqfS2kI7kXzd0N0fNuKhGGQeUEPa09dmcvkBw8iDZjKCA7EP6
 18 | n@g.com                    | $2b$10$EyDg7Tg58gdF64Qag36zIeTgKFKjImgn8u4NGZ0TIJCIgMIiT/stK
 19 | sikka.n@husky.neu.edu      | $2b$10$wZF1PBdP8lSd7AU1TvOwkekiAtqwXPtye0IvypCvH.Rygq1wlp6YO
 20 | yadavsunil251993@gmail.com | $2b$10$4LDvOItBs2F/hc1wcoVHhePVdaS2BVCUhLu0cVi0Eo2YTfL6uCm3S
 21 | nishita.sikka26@gmail.com  | $2b$10$MAy6AJ3Nexb.cghAOPUNbeJ38SsZf8EDPMQDEOkuup70Cz1YsNQRK
(21 rows)

public=#
```

public=# select * from Users where id=50;

```
public=# ^C
public=# select * from Users where id=50;
 id | email | password
----+-------+----------
(0 rows)
```

There is no record with id = 50

public=# select * from Users where id=50 or 1=1;

```
public=# select * from Users where id=50 or 1=1;
 id |            email            |                                                      password
----+-----------------------------+------------------------------------------------------------------
  1 | nishita@gmail.com           | $2b$10$zfxJGyyC6yQGYIDmzBCdwesbzK9er65OHDhZeqMeON324NEQGJBxC
  2 | nia@gmail.com               | $2b$10$a8VHe0qSmtE4gRWYulUgZ.wqnGxNsfEy0eFgDV0AJgpfDLKzNE9bG
  3 | nishitasikka12@gmail.com    | $2b$10$0JY4Jksp4cmndtIK/5MqteJQxNI8LMqmknIL/p613KZJn.ZhWb3Ku
  4 | aaaaikka12@gmail.com        | $2b$10$NzrutFRxk5zQ1aw58VmH4.aBejG2vQBC76gPlyZHK2v6o5VKSljsu
  5 | aaikka12@gmail.com          | $2b$10$31S7BJ75oAOxlFCk2eHAEOEmm7XRLvLPImq0braTaRlKF/ohrPvKa
  6 |                             | $2b$10$zjUq3ZI9NsMUCFwi3RhitetsO8DuWZUqffoQCI8M0u9BLJvTKTgnm
  7 | dnn@gmail.com               | $2b$10$7LCHm837Q964ITW.bghnAe2GhHs3cKOOIbtBWvDzssQ7TE4LwNPtm
  8 | nishitasikka2695@gmail.com  | $2b$10$.PoafGawtJ3YuOdhhigV8.mp6K/XLBcfYVBctWWeYrXUtOwAWob6u
  9 | nishitasikka26@mail.com     | $2b$10$uL.i9C0yccdOCa/gTtC4cuGMm2BmjJDlL8YTZO9RgYUsE/TY5Vz5C
 10 | nishi@mail.com              | $2b$10$pWJZ.VwlyMhKaB7IaPURgeQqUwAaLCkuj34wde8kqEwMEYXvG0Yf.
 11 | code@gmail.com              | $2b$10$IfOPCjeyp8N2KyqK/5nmeO/.fZxlbAOX5FjzF8SEUsp69hS9hu0RK
 12 | review@gmail.com            | $2b$10$ipUg3RbFSTY/MDk8STS3BOIFvB3TLQpPJ/gQ8/3XObFPggEW7NsFe
 13 | ashish@code.com             | $2b$10$Ox0IJ2zSzhIPnsdC/72EXulvsRxVszC3p68/hRWZ3ZmKk.CU0VK3e
 14 | nishita@code.com            | $2b$10$eQ/.DVWAfat5aF9mwaubguWrTdUIX4r0KfXdHVXd/IXboGKGXsrIG
 15 | test@code.com               | $2b$10$Qe0N.S0eU9Sgap/l..7bseWzKM6rWlTMiGI5QEyszoYe4D/JriJB6
 16 | f@f.com                     | $2b$10$OBeclxfBTvusS4cw.JJx0.BIoym1zIxuN6ApJPgGBiAiWhG.gq3ua
 17 | hey@hey.com                 | $2b$10$xS9KKqfS2kI7kXzd0N0fNuKhGGQeUEPa09dmcvkBw8iDZjKCA7EP6
 18 | n@g.com                     | $2b$10$EyDg7Tg58gdF64Qag36zIeTgKFKjImgn8u4NGZ0TIJCIgMIiT/stK
 19 | sikka.n@husky.neu.edu       | $2b$10$wZF1PBdP8lSd7AU1TvOwkekiAtqwXPtye0IvypCvH.Rygq1wlp6YO
 20 | yadavsunil251993@gmail.com  | $2b$10$4LDvOItBs2F/hc1wcoVHhePVdaS2BVCUhLu0cVi0Eo2YTfL6uCm3S
 21 | nishita.sikka26@gmail.com   | $2b$10$MAy6AJ3Nexb.cghAOPUNbeJ38SsZf8EDPMQDEOkuup70Cz1YsNQRK
(21 rows)
```

Displaying on the 21 values in the whole table since OR 1=1 is provided trying to make the query succeed.

public=# SELECT * from Users where email = '1' or '1' = '1' and password = '1' or '1' = '1';

```
ublic=# SELECT * from Users where email = '1' or '1' = '1' and password = '1' or '1' = '1';
id |            email            |                                                      password
---+-----------------------------+------------------------------------------------------------------
 1 | nishita@gmail.com           | $2b$10$zfxJGyyC6yQGYIDmzBCdwesbzK9er65OHDhZeqMeON324NEQGJBxC
 2 | nia@gmail.com               | $2b$10$a8VHe0qSmtE4gRWYulUgZ.wqnGxNsfEy0eFgDV0AJgpfDLKzNE9bG
 3 | nishitasikka12@gmail.com    | $2b$10$0JY4Jksp4cmndtIK/5MqteJQxNI8LMqmknIL/p613KZJn.ZhWb3Ku
 4 | aaaaikka12@gmail.com        | $2b$10$NzrutFRxk5zQ1aw58VmH4.aBejG2vQBC76gPlyZHK2v6o5VKSljsu
 5 | aaikka12@gmail.com          | $2b$10$31S7BJ75oAOxlFCk2eHAEOEmm7XRLvLPImq0braTaRlKF/ohrPvKa
 6 |                             | $2b$10$zjUq3ZI9NsMUCFwi3RhitetsO8DuWZUqffoQCI8M0u9BLJvTKTgnm
 7 | dnn@gmail.com               | $2b$10$7LCHm837Q964ITW.bghnAe2GhHs3cKOOIbtBWvDzssQ7TE4LwNPtm
 8 | nishitasikka2695@gmail.com  | $2b$10$.PoafGawtJ3YuOdhhigV8.mp6K/XLBcfYVBctWWeYrXUtOwAWob6u
 9 | nishitasikka26@mail.com     | $2b$10$uL.i9C0yccdOCa/gTtC4cuGMm2BmjJDlL8YTZO9RgYUsE/TY5Vz5C
10 | nishi@mail.com              | $2b$10$pWJZ.VwlyMhKaB7IaPURgeQqUwAaLCkuj34wde8kqEwMEYXvG0Yf.
11 | code@gmail.com              | $2b$10$IfOPCjeyp8N2KyqK/5nmeO/.fZxlbAOX5FjzF8SEUsp69hS9hu0RK
12 | review@gmail.com            | $2b$10$ipUg3RbFSTY/MDk8STS3BOIFvB3TLQpPJ/gQ8/3XObFPggEW7NsFe
13 | ashish@code.com             | $2b$10$Ox0IJ2zSzhIPnsdC/72EXulvsRxVszC3p68/hRWZ3ZmKk.CU0VK3e
14 | nishita@code.com            | $2b$10$eQ/.DVWAfat5aF9mwaubguWrTdUIX4r0KfXdHVXd/IXboGKGXsrIG
15 | test@code.com               | $2b$10$Qe0N.S0eU9Sgap/l..7bseWzKM6rWlTMiGI5QEyszoYe4D/JriJB6
16 | f@f.com                     | $2b$10$OBeclxfBTvusS4cw.JJx0.BIoym1zIxuN6ApJPgGBiAiWhG.gq3ua
17 | hey@hey.com                 | $2b$10$xS9KKqfS2kI7kXzd0N0fNuKhGGQeUEPa09dmcvkBw8iDZjKCA7EP6
18 | n@g.com                     | $2b$10$EyDg7Tg58gdF64Qag36zIeTgKFKjImgn8u4NGZ0TIJCIgMIiT/stK
19 | sikka.n@husky.neu.edu       | $2b$10$wZF1PBdP8lSd7AU1TvOwkekiAtqwXPtye0IvypCvH.Rygq1wlp6YO
20 | yadavsunil251993@gmail.com  | $2b$10$4LDvOItBs2F/hc1wcoVHhePVdaS2BVCUhLu0cVi0Eo2YTfL6uCm3S
21 | nishita.sikka26@gmail.com   | $2b$10$MAy6AJ3Nexb.cghAOPUNbeJ38SsZf8EDPMQDEOkuup70Cz1YsNQRK
21 rows)
```

Now in order to bypass the security of the database the SQL code needs to be injected on the valid input fields in a way that the statement should always be valid on execution. Similarly, in the above query, a hacker can see the email and the password or any other information if they use ‘1’=’1’ and can even actually DROP this entire table using the same query as follows.

public=# select * from Users where id=50 or 1=1; DROP TABLE Users;

Now the hacker has actually dropped the entire table in the database, resulting toward loss of important user information. Hence, providing that the table was deleted even with an incorrect query.

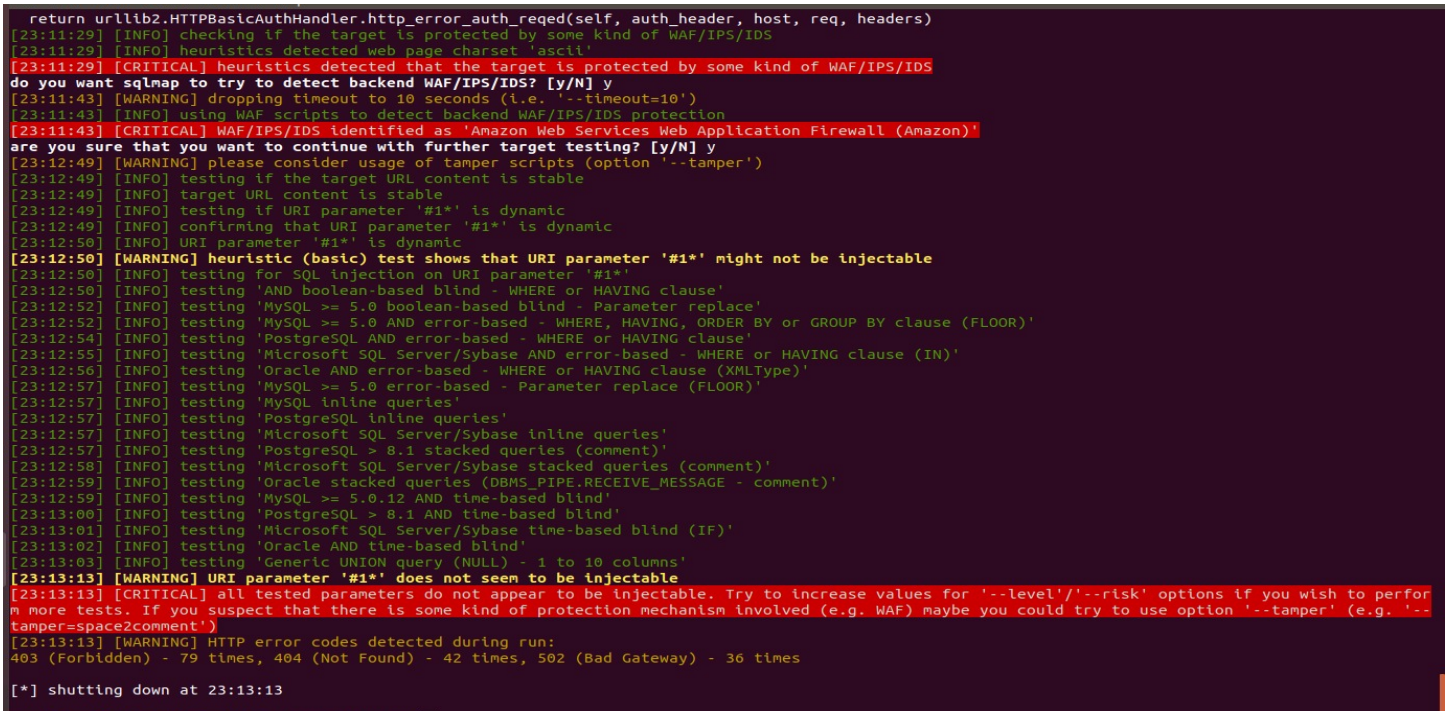
SQL Injection: SQL map to the domain csye6225-spring2019-sikkan.me. It can be seen that the SQL map starts, and the target URL is stable and SQL map is unable to fetch any data and gets a 403 status.

```
nishitasikka@ubuntu:~$ sqlmap -u https://csye6225-spring2019-sikkan.me
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.3.2#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 ap
plicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 22:53:48 /2019-04-01/

[22:53:48] [INFO] testing connection to the target URL
[22:53:48] [INFO] heuristics detected web page charset 'ascii'
[22:53:48] [WARNING] the web server responded with an HTTP error code (403) which could interfere with the results of the tests
[22:53:48] [INFO] testing if the target URL content is stable
[22:53:49] [INFO] target URL content is stable
[22:53:49] [CRITICAL] no parameter(s) found for testing in the provided data (e.g. GET parameter 'id' in 'www.site.com/index.php?id=1')
[22:53:49] [WARNING] HTTP error codes detected during run:
403 (Forbidden) - 2 times

[*] ending @ 22:53:49 /2019-04-01/
```

Now, in the screenshot below the we are trying to implement SQL injection while having the WAF active on our load balancer which results that it is not injectable.

sqlmap -u "https://csye6225-spring2019-sikkan.me/note/1e6dc90d-4ceb-45cc-a04e-6cdccad3363d" --auth-type=BASIC –auth cred="sikka.n@husky.neu.edu:Pass@1234" --dbms "PostgreSQL" --level 5 --risk 3

```
[23:34:15] [INFO] checking if the injection point on Referer parameter 'Referer' is a false positive
[23:34:16] [WARNING] false positive or unexploitable injection point detected
[23:34:16] [WARNING] Referer parameter 'Referer' does not seem to be injectable
[23:34:16] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[23:34:16] [WARNING] User-Agent parameter 'User-Agent' does not appear to be dynamic
[23:34:16] [WARNING] heuristic (basic) test shows that User-Agent parameter 'User-Agent' might not be injectable
[23:34:17] [INFO] testing for SQL injection on User-Agent parameter 'User-Agent'
[23:34:17] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[23:34:17] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[23:34:17] [INFO] testing 'PostgreSQL OR error-based - WHERE or HAVING clause'
[23:34:18] [INFO] testing 'PostgreSQL error-based - Parameter replace'
[23:34:18] [INFO] testing 'PostgreSQL error-based - Parameter replace (GENERATE_SERIES)'
[23:34:18] [INFO] testing 'PostgreSQL inline queries'
[23:34:18] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[23:34:18] [INFO] testing 'PostgreSQL > 8.1 stacked queries'
[23:34:18] [INFO] testing 'PostgreSQL stacked queries (heavy query - comment)'
[23:34:19] [INFO] testing 'PostgreSQL stacked queries (heavy query)'
[23:34:19] [INFO] testing 'PostgreSQL < 8.2 stacked queries (Glibc - comment)'
[23:34:19] [INFO] testing 'PostgreSQL < 8.2 stacked queries (Glibc)'
[23:34:19] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[23:34:19] [INFO] testing 'PostgreSQL > 8.1 OR time-based blind'
[23:34:19] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind (comment)'
[23:34:20] [INFO] testing 'PostgreSQL > 8.1 OR time-based blind (comment)'
[23:34:20] [INFO] testing 'PostgreSQL AND time-based blind (heavy query)'
[23:34:20] [INFO] testing 'PostgreSQL OR time-based blind (heavy query)'
[23:34:20] [INFO] testing 'PostgreSQL AND time-based blind (heavy query - comment)'
[23:34:20] [INFO] testing 'PostgreSQL OR time-based blind (heavy query - comment)'
[23:34:21] [INFO] testing 'PostgreSQL > 8.1 time-based blind - Parameter replace'
[23:34:21] [INFO] testing 'PostgreSQL time-based blind - Parameter replace (heavy query)'
[23:34:21] [INFO] testing 'Generic UNION query (37) - 1 to 20 columns'
[23:34:43] [INFO] testing 'Generic UNION query (37) - 21 to 40 columns'
[23:34:47] [INFO] testing 'Generic UNION query (37) - 41 to 60 columns'
[23:34:50] [INFO] testing 'Generic UNION query (37) - 61 to 80 columns'
[23:34:55] [INFO] testing 'Generic UNION query (37) - 81 to 100 columns'
[23:34:58] [INFO] checking if the injection point on User-Agent parameter 'User-Agent' is a false positive
[23:34:58] [WARNING] false positive or unexploitable injection point detected
[23:34:58] [WARNING] User-Agent parameter 'User-Agent' does not seem to be injectable
[23:34:58] [CRITICAL] all tested parameters do not appear to be injectable. 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')
[23:34:58] [WARNING] HTTP error codes detected during run:
400 (Bad Request) - 3 times, 404 (Not Found) - 11 times, 502 (Bad Gateway) - 517 times

[*] shutting down at 23:34:58
```



Reasons for choosing this Attack Vector –

Injections result into loss of user data/other important content and result towards denial of access. Also, any source of injection vector maybe used by the attacker for proposing vulnerabilities in SQL. Also, if the values contain the usual SQL syntax statements the database tends to execute the statements as such and causes triggers which have really bad/dangerous consequences.

2. **A2: Broken Authentication, A7: Insufficient Attack Protection** – (Using AWS WAF)

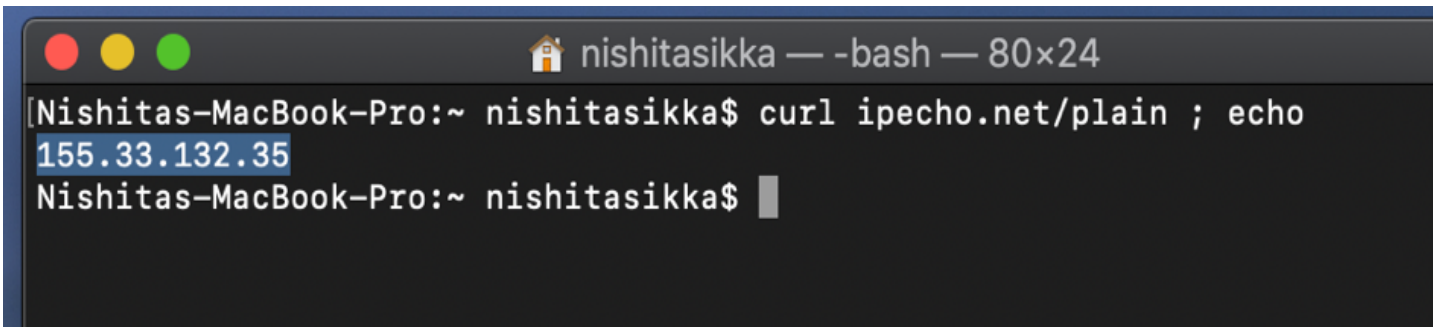
Attack vector:

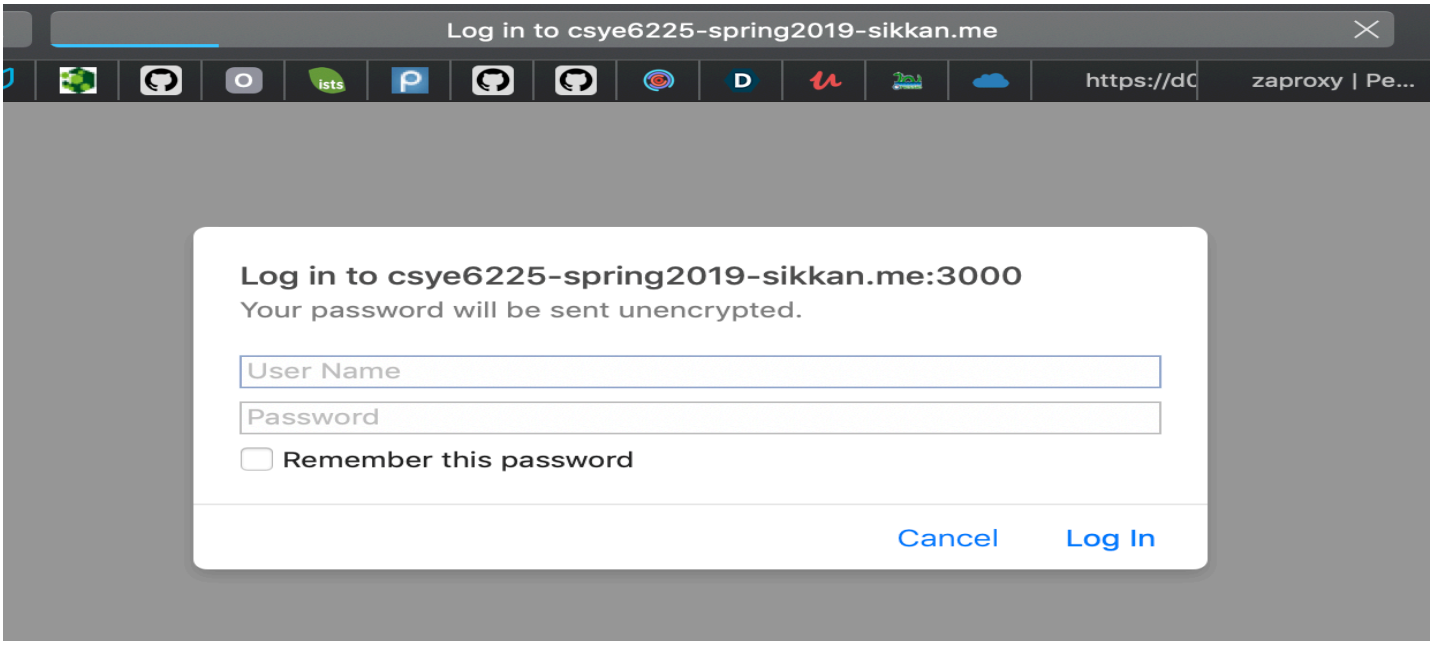
The most common attack vector is the broken authentication when the IP address or the password of the user is actually accessible. This way the hackers can actually hide behind the genuine user ID and gain all the access to the web application/network and the programs.

Strong Authentication is highly required to ensure that the unique identity of the user since for the web applications it can expose unwanted data of the user such as credentials or even sessions.

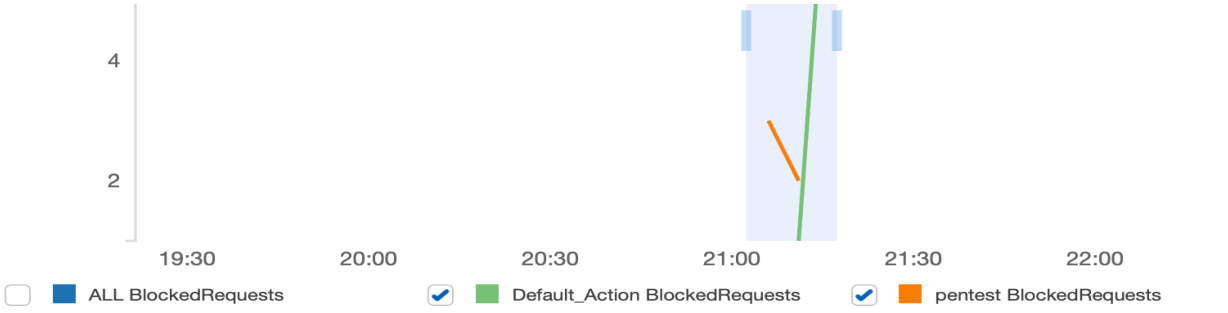
IP Backlisting: An access list basically for not allowing the IP address to access control. Hence, blacklisting could be one way to protect your user identity.

Public IP address of computer – 155.33.132.35





Using WAF and blacklisting the public IP address the following are the logs from **AWS WAF & Shield**



Sampled requests

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

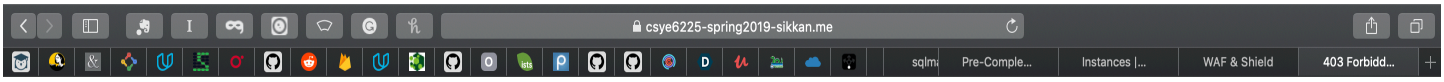
pen_test

Get new samples

Sample data from 2019-03-31 01:02:18 to 01:17:18

Source IP	URI	Matches rule	Action	Time (UTC)
155.33.132.35	/	pen_test	Block	01:09:59
155.33.132.35	/favicon.ico	pen_test	Block	01:10:00
155.33.132.35	/	pen_test	Block	01:10:15
155.33.132.35	/	pen_test	Block	01:11:01
155.33.132.35	/	pen_test	Block	01:11:03

Since the IP address is blocked the user receives 403 Forbidden Blacklisted IP Address



403 Forbidden

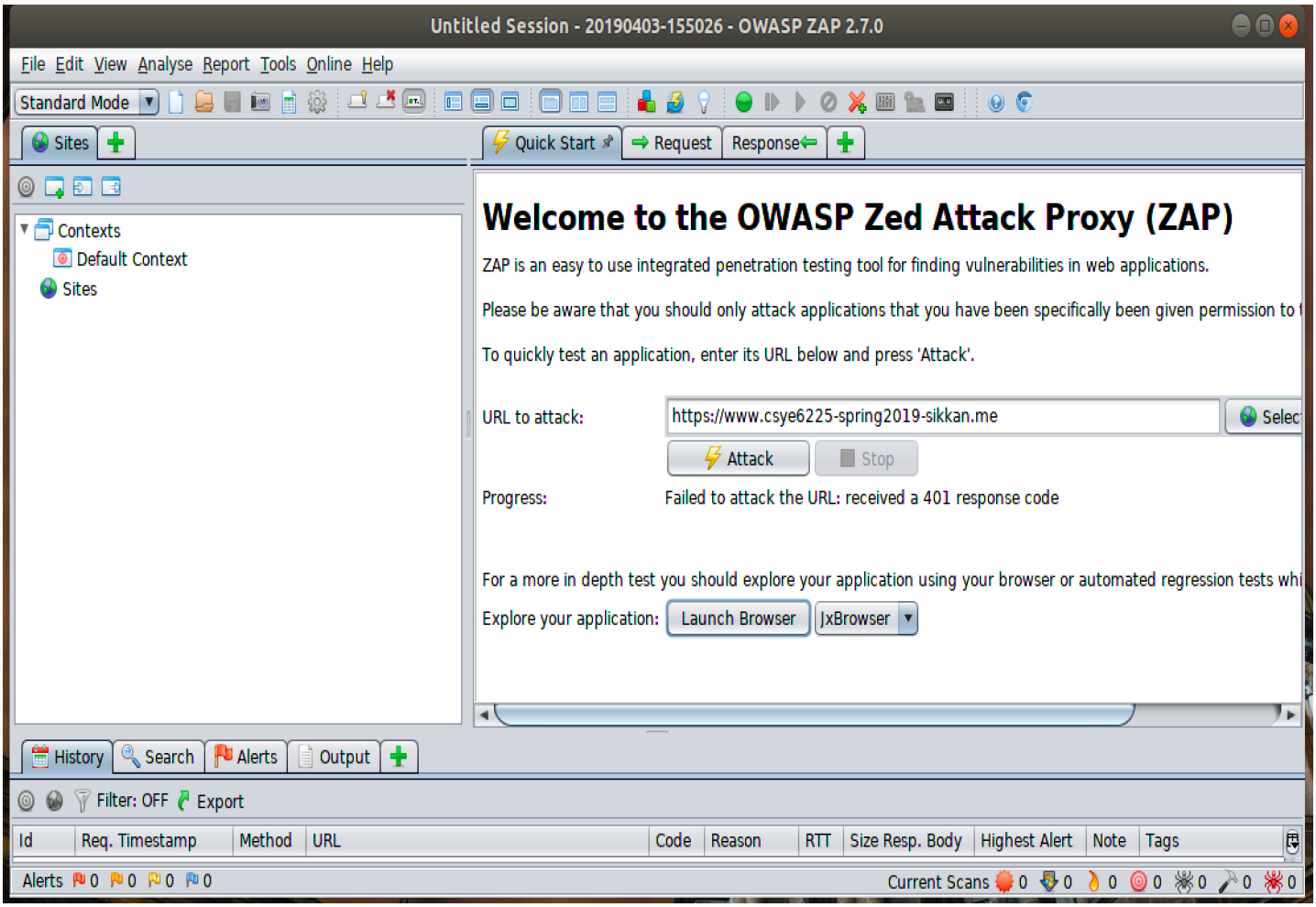
Reasons for choosing this Attack Vector –

Usually Broken Authentication is one of the main reasons why hackers are easily able access millions of user credentials/sessions and can easily hide behind genuine user ID. Credentials can be guessed easily overwritten through weak account functions and weak sessions. Also, users can be impersonated using the tokens while launching HTTP requests until the token expires.

3. **A5 – Security Misconfiguration, Cookie Without Secure Flags/ Incomplete or No Cache-Control –** (Using Kali Linux)

Attack vector:

If a cookie has been accessed without a secure flag the cookie can be accessed via any means of an unencrypted connection. Also, when it comes to cache-control and pragma HTTP header not being set properly may allow the browser/proxies to cache content.



In our case we have a secured connection and hence the URL can't be attacked which shows complete protection for the API. The great solution to this is that whenever a cookie contains a sensitive information or is a session token, then it should always be passed using an encrypted channel ensuring that the secure flag or the cookie set containing sensitive information. Lastly, ensure that the cache-control HTTP header us set.

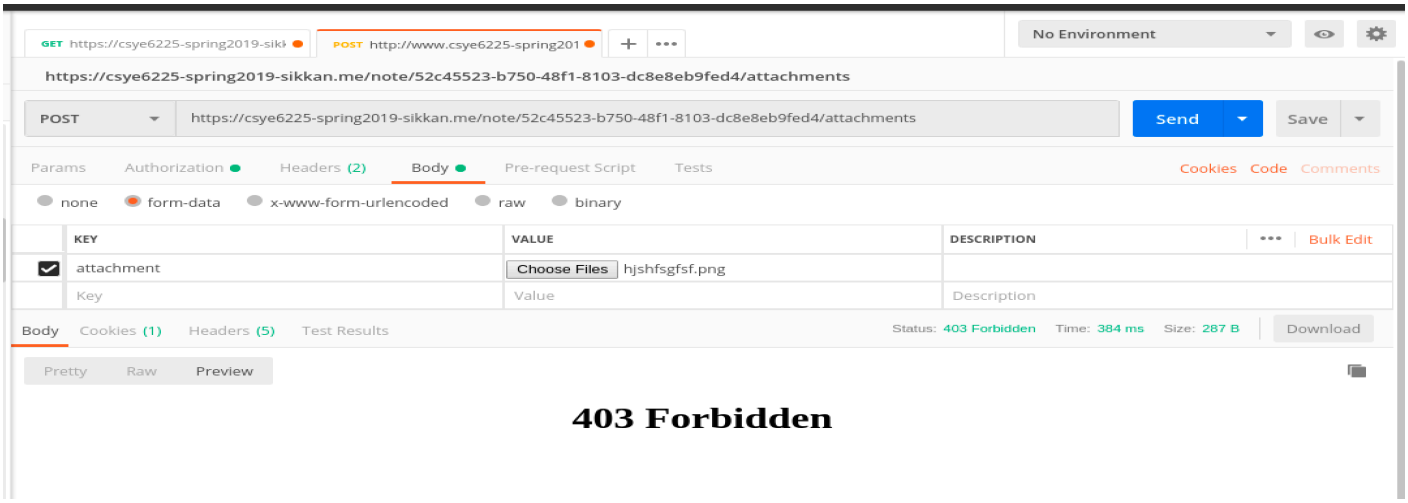
Reasons for choosing this Attack Vector –

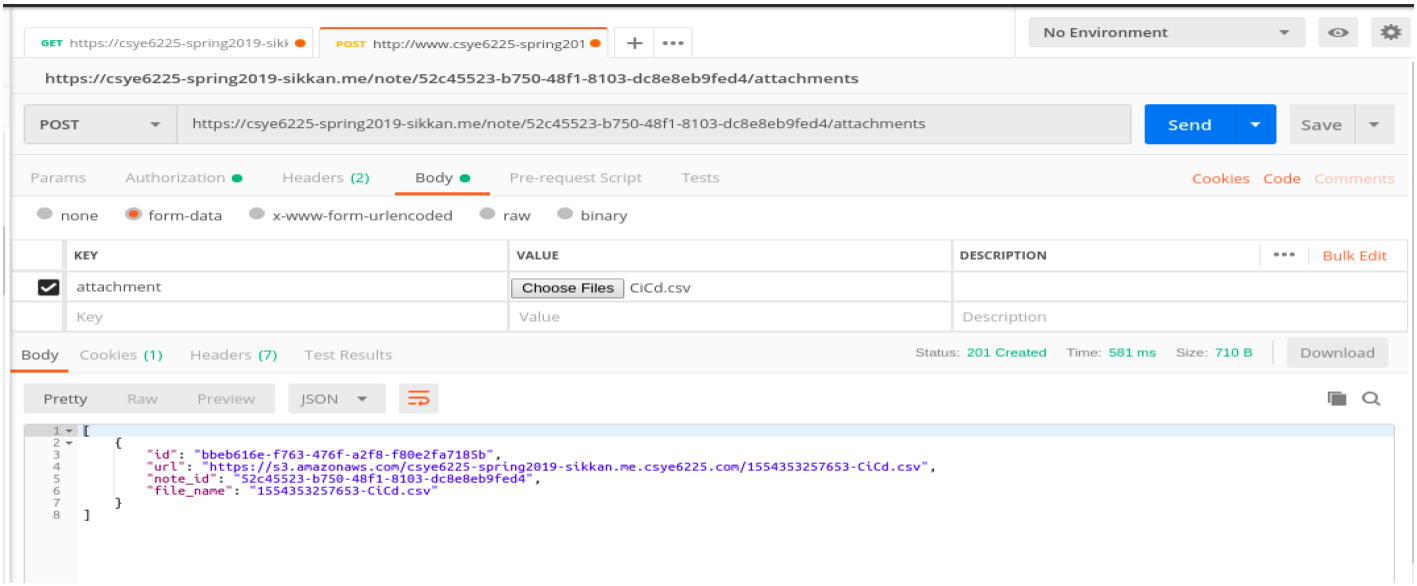
Misconfigurations especially that have a security impact can attack your application anytime applicable to the operating system, middleware, platform, application code etc. For instance, if the Server Tokens are full for a web server in a production system, the versions for the web server may be exposed and enable information leaks in the software. Hence AWS WAF may be used to get rid of such a misconfiguration. Also, there might be a case where the stack traces are returned, again this can be dangerous for the application.

4. A7: Insufficient Attack Protection: Size Constraint conditions:
Reasons for choosing this Attack Vector –

For this category the hackers are able to adapt the exact toolset to be able to exploit the vulnerabilities while launching automated attacks to detect vulnerable systems which can cause abnormal request patterns or flaws in an application.
Hence one way is having the size constraint conditions which helps to ensure if the HTTP request falls within a range and avoids processing abnormal requests.

In our case for example when it comes to adding attachments to the notes, the file with the bigger size gives a forbidden error whereas the file with a smaller size is easily attached.





References:

- https://d0.awsstatic.com/whitepapers/Security/aws-waf-owasp.pdf
- https://www.synopsys.com/blogs/software-security/owasp-top-10-application-security-risks/
- https://www.ostechnix.com/install-kali-linux-tools-using-katoolin-linux/
- https://www.kali.org