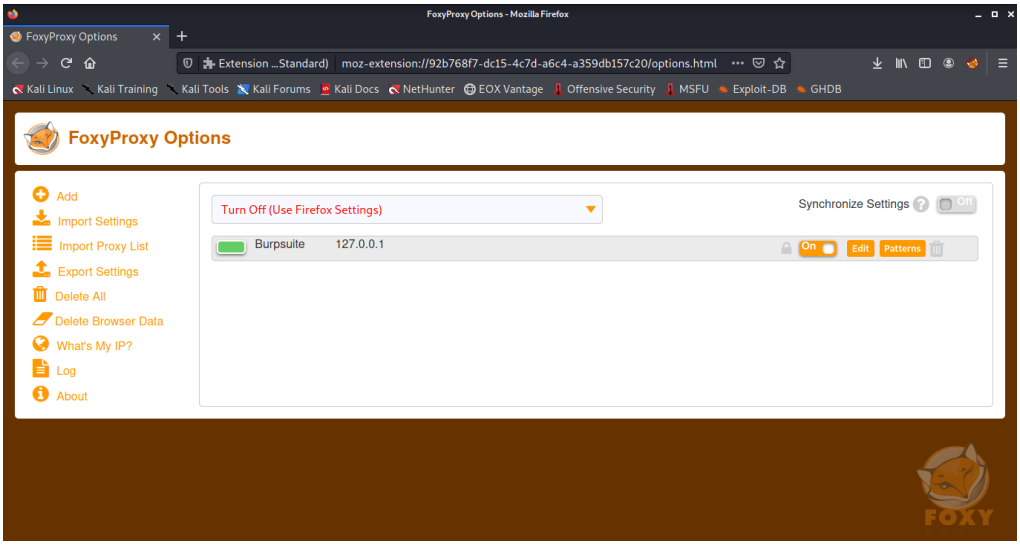
B.TECH	COMPUTER SCIENCE ENGINEERING	PESU
	INFORMATION SECURITY	

BUG BOUNTY ASSIGNMENT

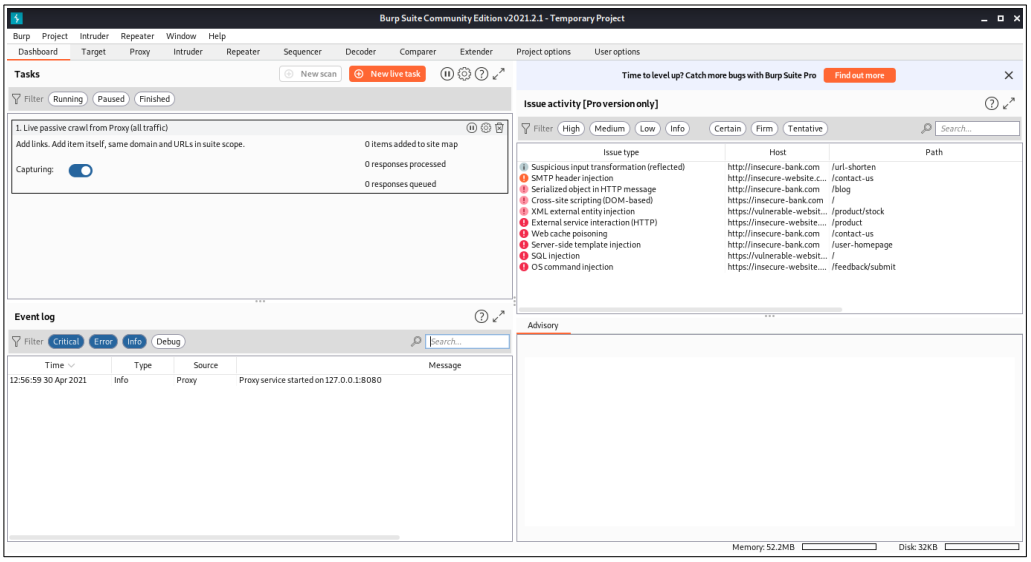
Important Details regarding this assignment: -

1. **Operating System**: - Kali Linux
2. **Software(s) Used**: - Burp Suite
3. **Web Browser**: - Mozilla Firefox
4. **Add-ons**: - Foxy Proxy Standard

Screenshot of the **Foxy Proxy** options for **burpsuite**,



burpsuite Community Edition UI Screenshot,



Website Details: -

The website I will be using for this Bug Bounty Assignment is,

Damn Vulnerable Web Application (DVWA)

DVWA is a PHP/MySQL web application, whose main goal is to be an aid for security professionals to test their skills and tools in a legal environment.

More on **DVWA** – https://dvwa.co.uk/

For accessing this web application, we can simply use the docker version of this web application which will be accessible locally (***http://localhost:90***) and we won't be needing any other explicit permissions to perform Bug Bounty activities as this web application is hosted on own system (***locally – http://localhost:90***)as stated before.

To download the docker image and successfully run the web application to perform the Bug Bounty activities,

1. Download docker,

   sudo apt install docker.io

2. Download the docker image from the website,

   https://hub.docker.com/r/vulnerables/web-dvwa

3. Finally run the command to run the docker image,
   First run the below command,

   sudo usermod -aG docker $USER

   Next, make sure apache2 is running,

   service apache2 start

   service apache2 status (*to check if it is running or not*)

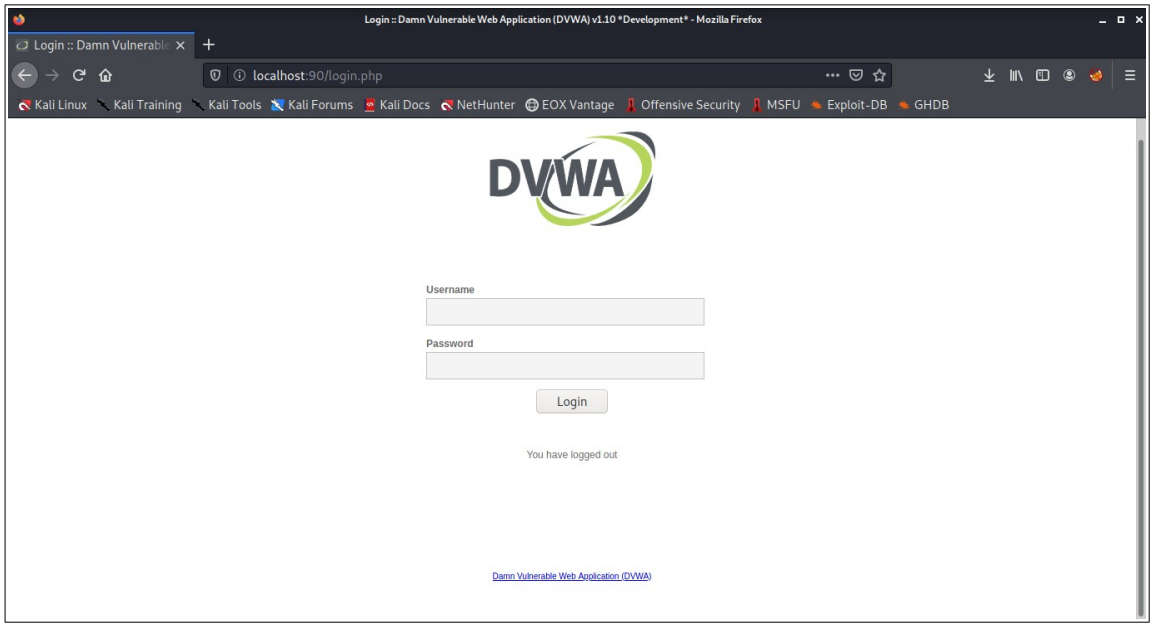
   Finally run the below command to run the docker images for **DVWA**,

   sudo docker run --rm -it -p 80:80 vulnerables/web-dvwa

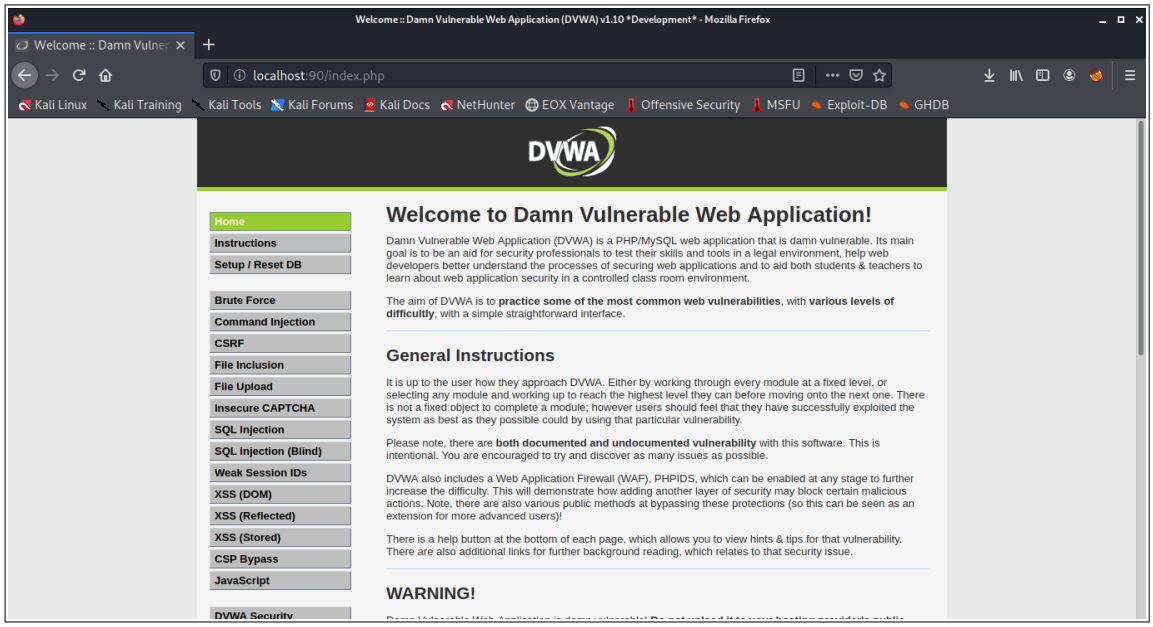
All the required docker images will be downloaded in some time and the website will be available locally at **localhost:80** and by default the credentials are **admin** and **password**.

In this case port **90** is being used for the **localhost** as port 80 is already being used. After logging in as **admin** and **password** we need to **Reset and Create** the Database for the website to start functioning and we will be redirected to the login page and again log in as **admin** and **password** we will see the website which looks like this,

Login page,

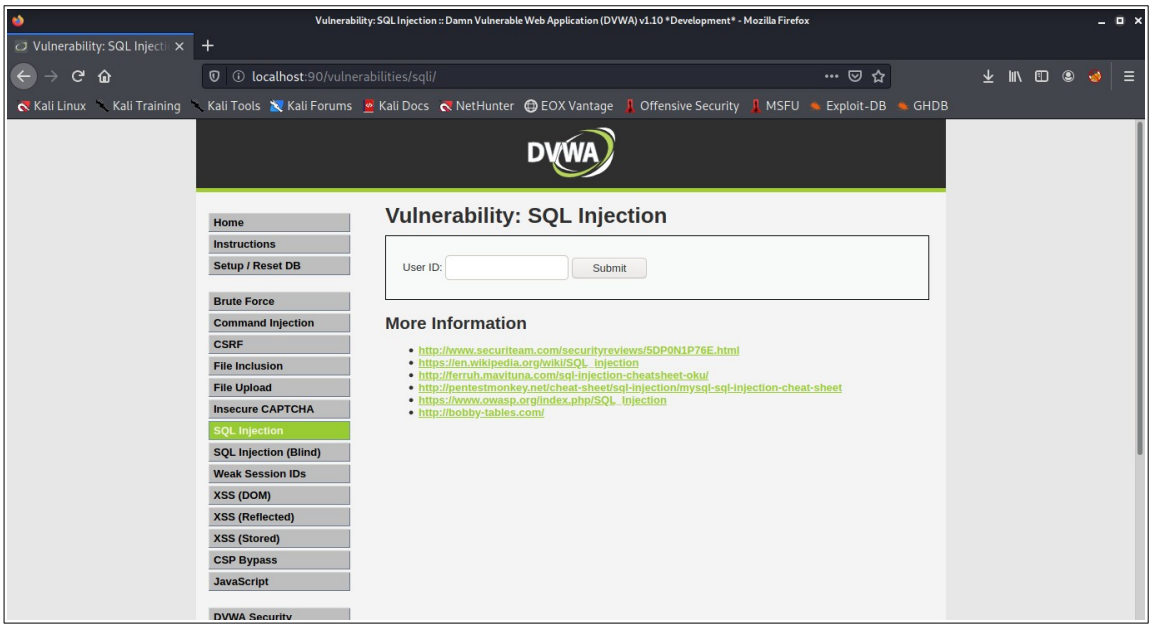


Main Website after logging in,

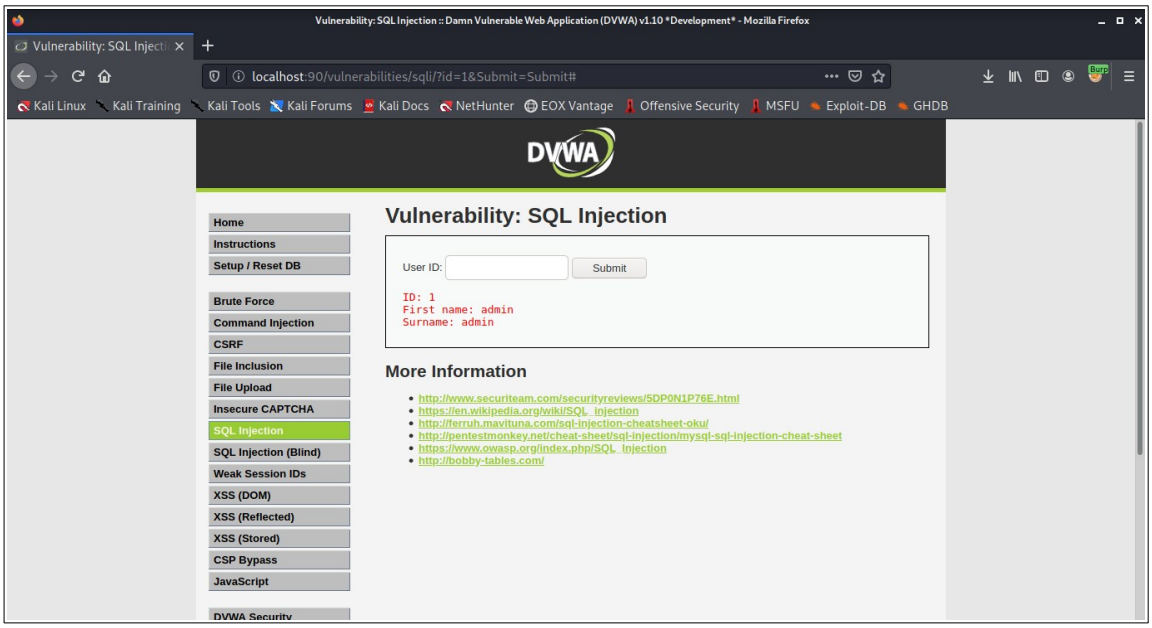


Vulnerability: SQL Injection

SQL Injection Webpage,

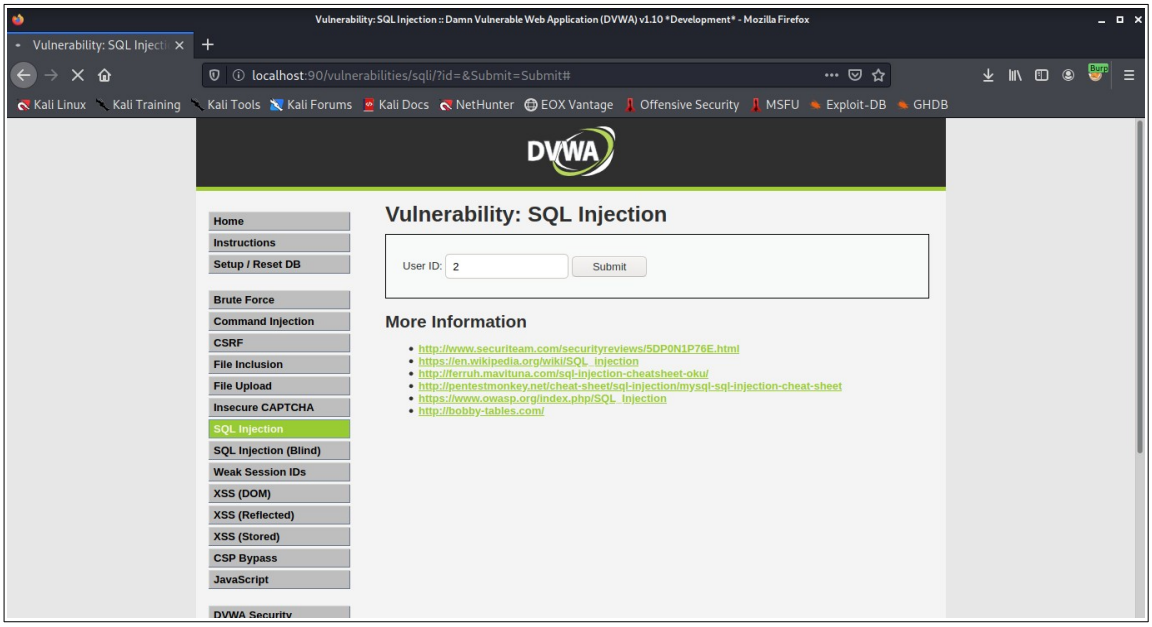


First lets see the output for User ID 1,

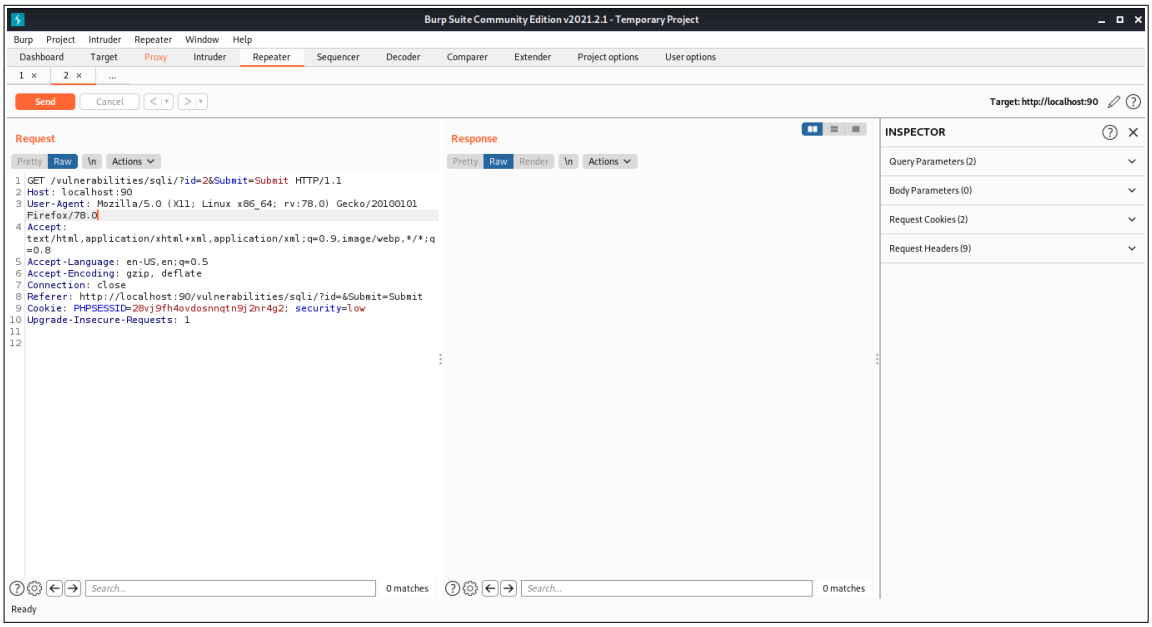


This was just to see what the output will be like.

Now for the attack we can try for User ID 2,

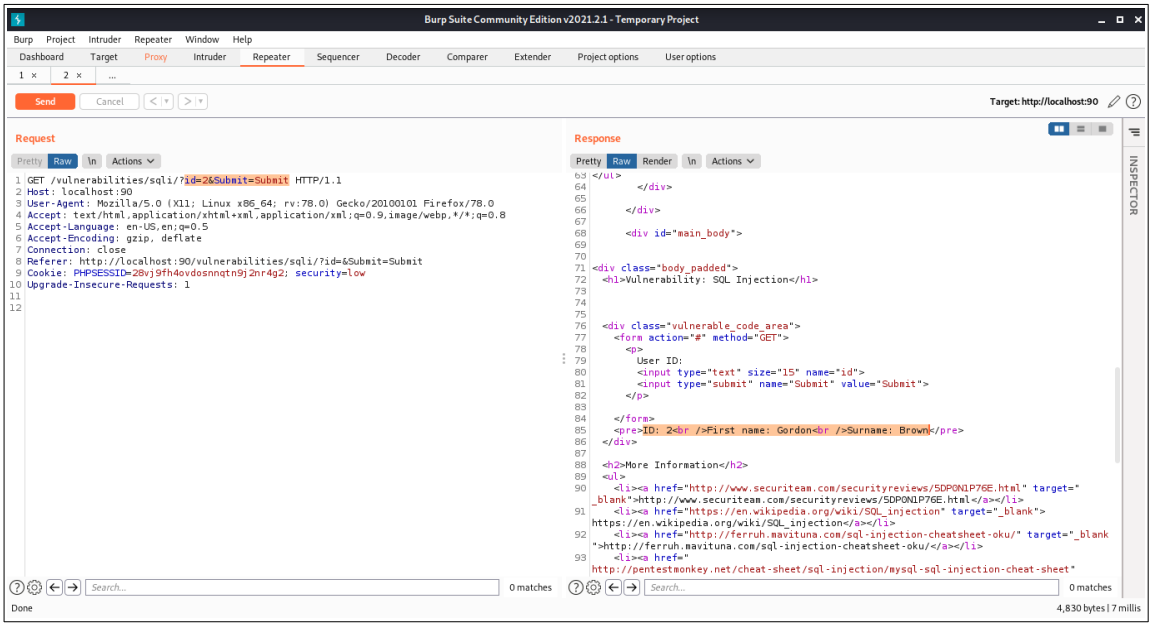


In burpsuite, we can observe various tweaks using the **Repeater** tab without having to reload the main website multiple times,

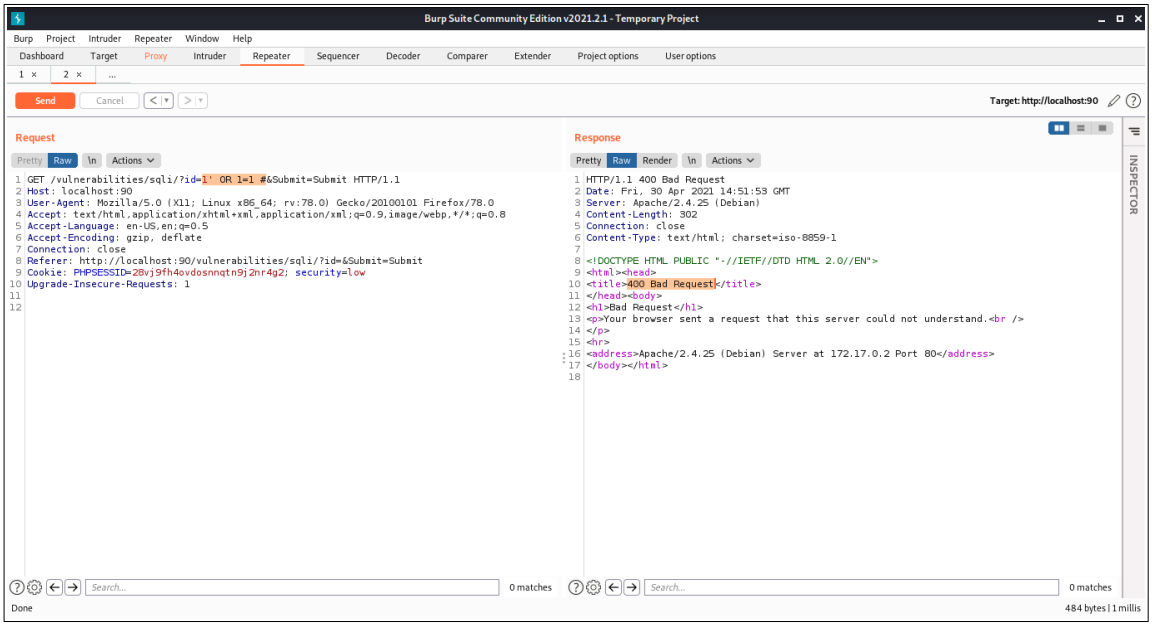


Here we can see the Response part as well for any modified request made.

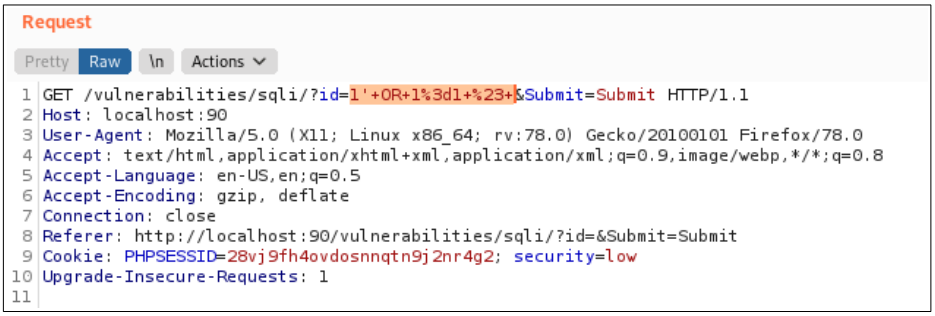
After clicking the send button we can see the details for user ID 2 without the website showing the respective details,



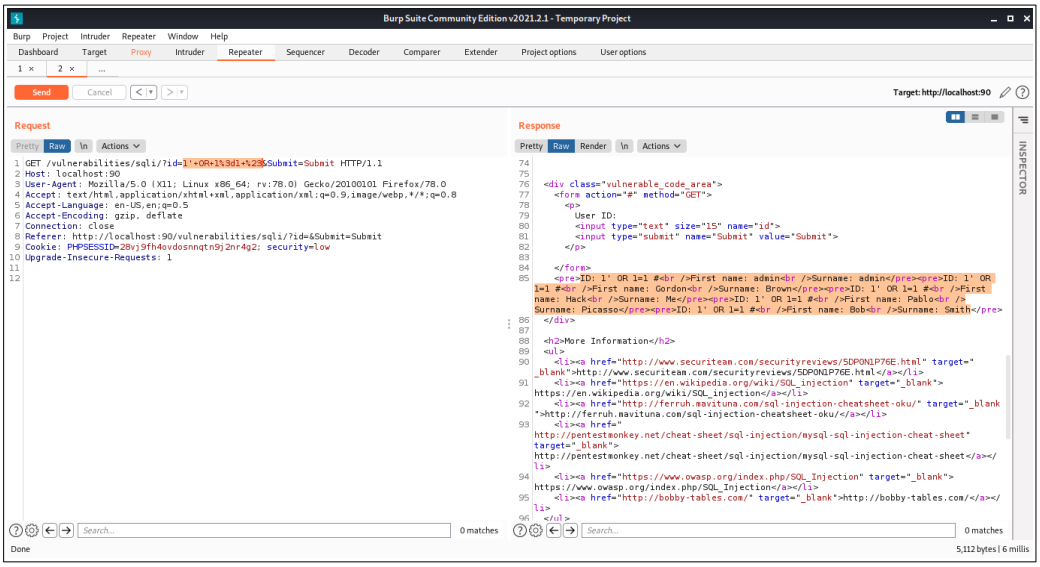
Now performing the SQL Injection attack,



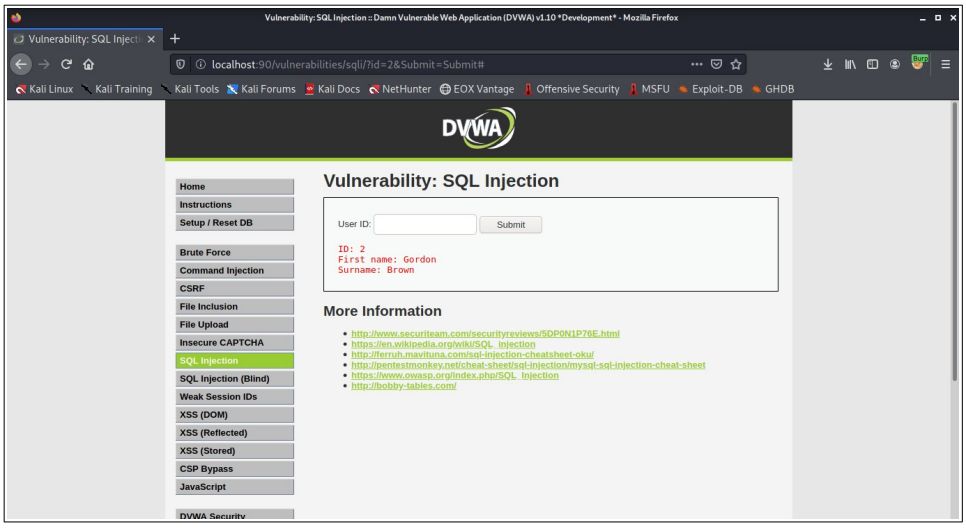
The highlighted texts show that we get a **400 Bad request**. This maybe due to some encode-decode complication as we are using **RAW HTTP** so we need to do is **URL encode**.



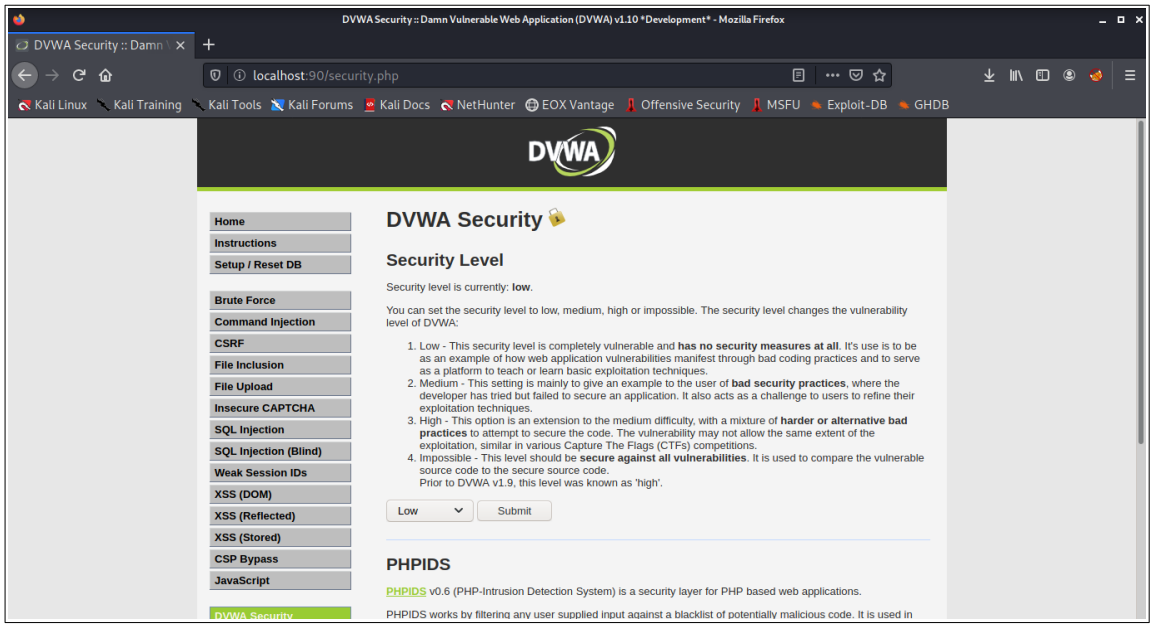
So here the highlighted text shows the URL encoded version of "**1' OR 1=1 #**".
Now we can send the request again,



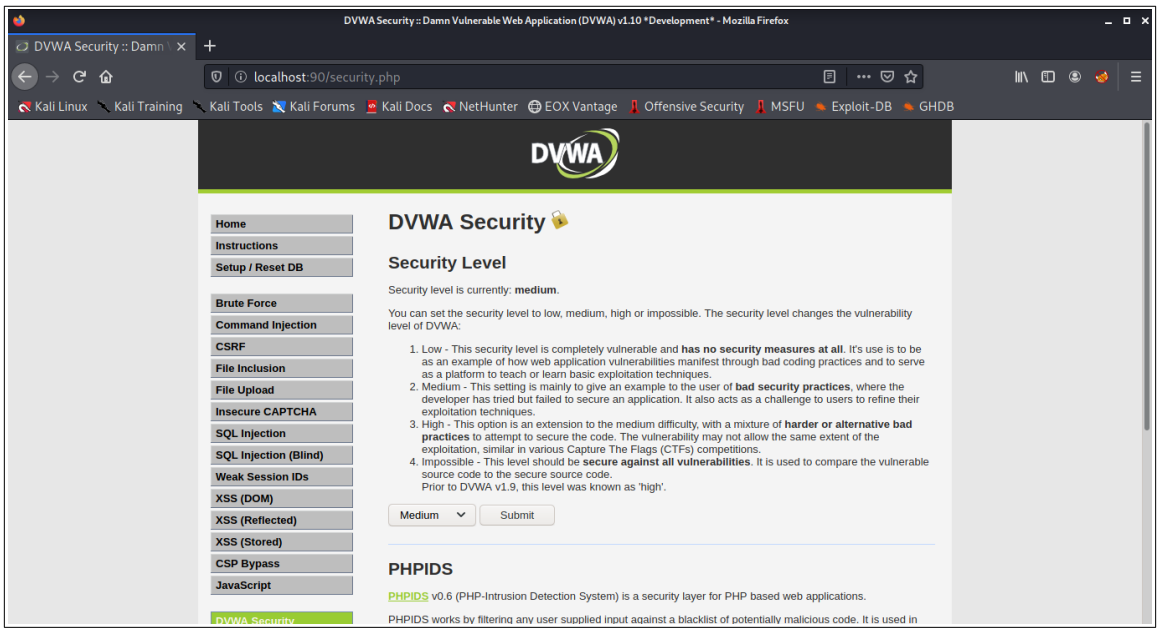
We can see now that some SQL Injection leaking all of the database out.
Once the request for the Submit action for User ID 2 is forwarding in the Proxy-> Intercept as seen earlier we can see the result in the webpage,



***Note**: - It can be found that the **DVWA** has an option of increasing the security level of the website,*

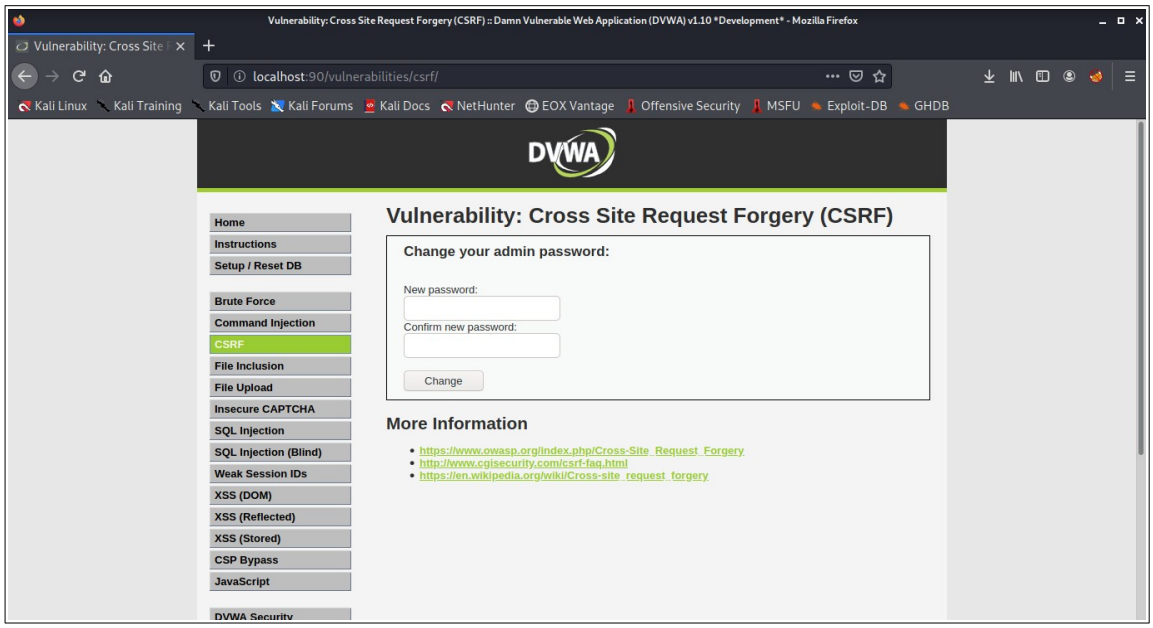


*So the Security level can be increased to **Medium**.*

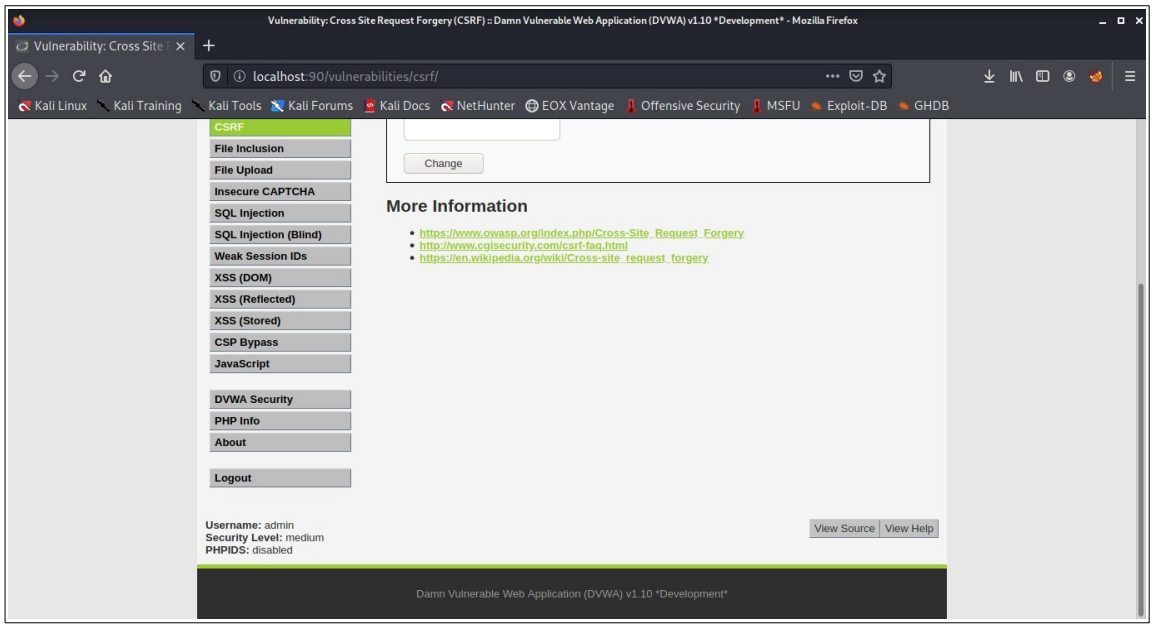


Vulnerability: Cross Site Request Forgery (CSRF)

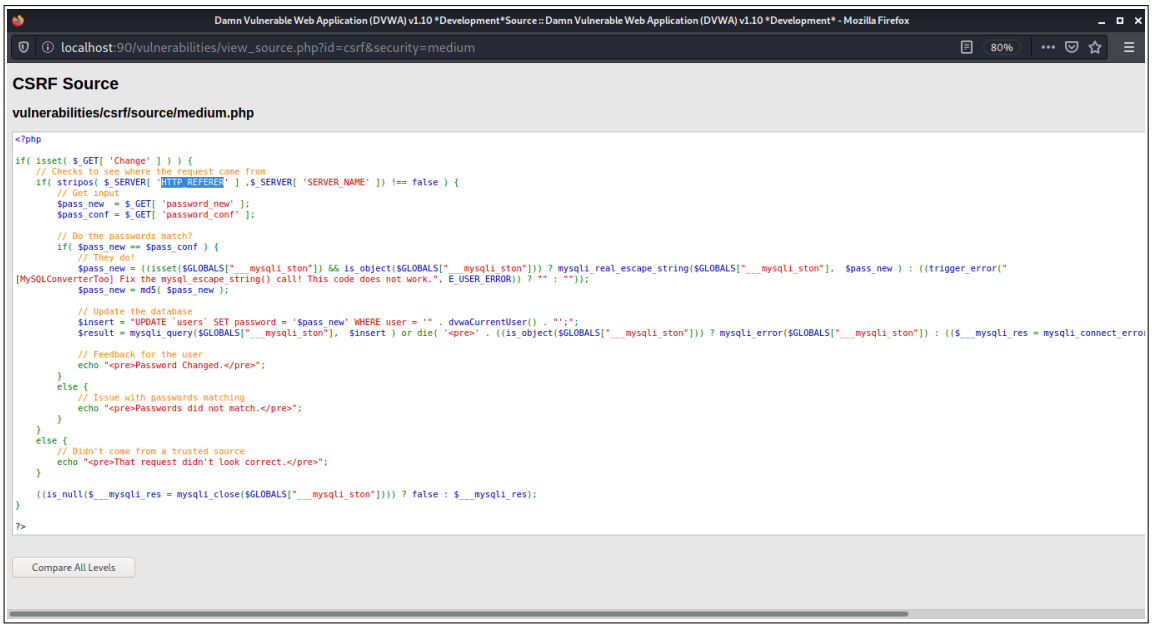
A quick look at the CSRF Vulnerability webpage,



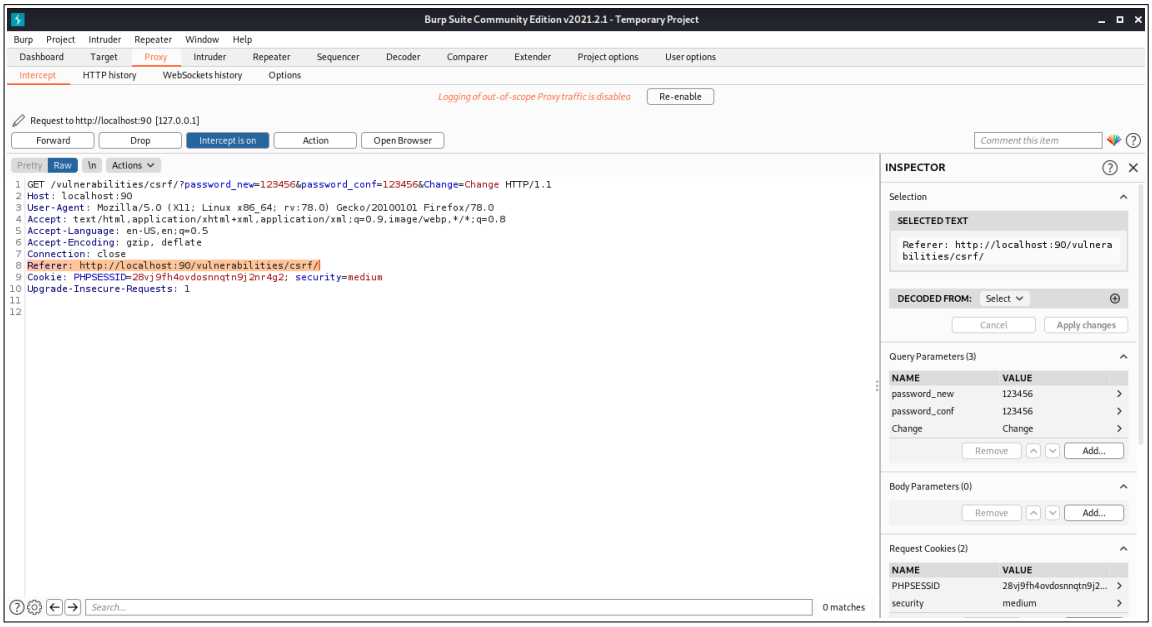
There is also a **View Source** button at the end,



The source code shows us the medium security settings(*the highlighted text*) to the webpage which is nothing but the use of HTTP referrer,



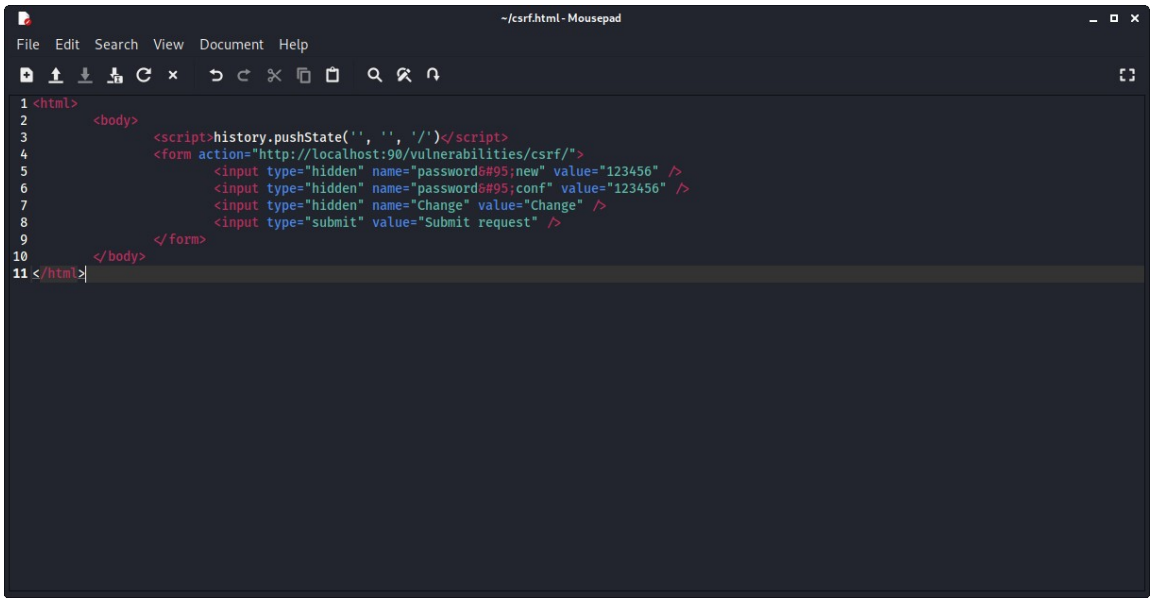
The **HTTP_REFERRER** is like a parameter when we try to change the password it tells the **SERVER** where the original request came from, the source, like where the link is generated from and if the source is not what it has to be from, then this **HTTP_REFERRER** gets removed. Trying this with changing the password to **123456**,



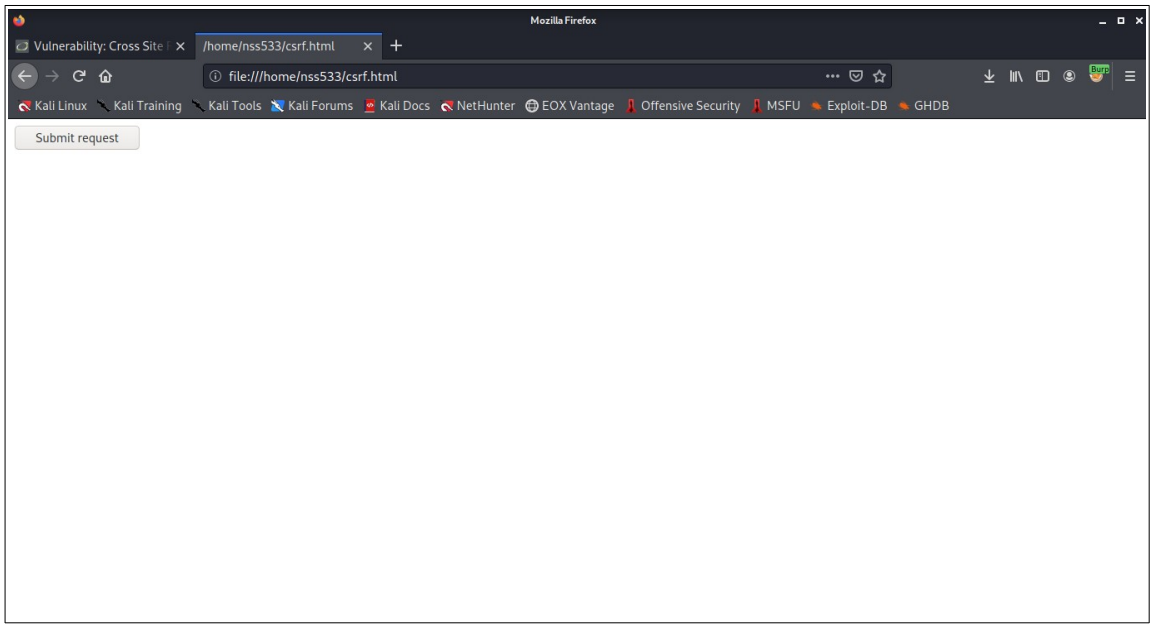
Here we can see the referrer parameter which is shown above(*highlighted text*)

In this attack with medium security while perform the CSRF attack, when we try to run this from another link the referrer parameter would be removed so it is better to keep a copy of this parameter – **Referer: http://localhost:90/vulnerabilities/csrf/**

Now we will create a CSRF html for the above request in a separate *html* file(*csrf.html*),

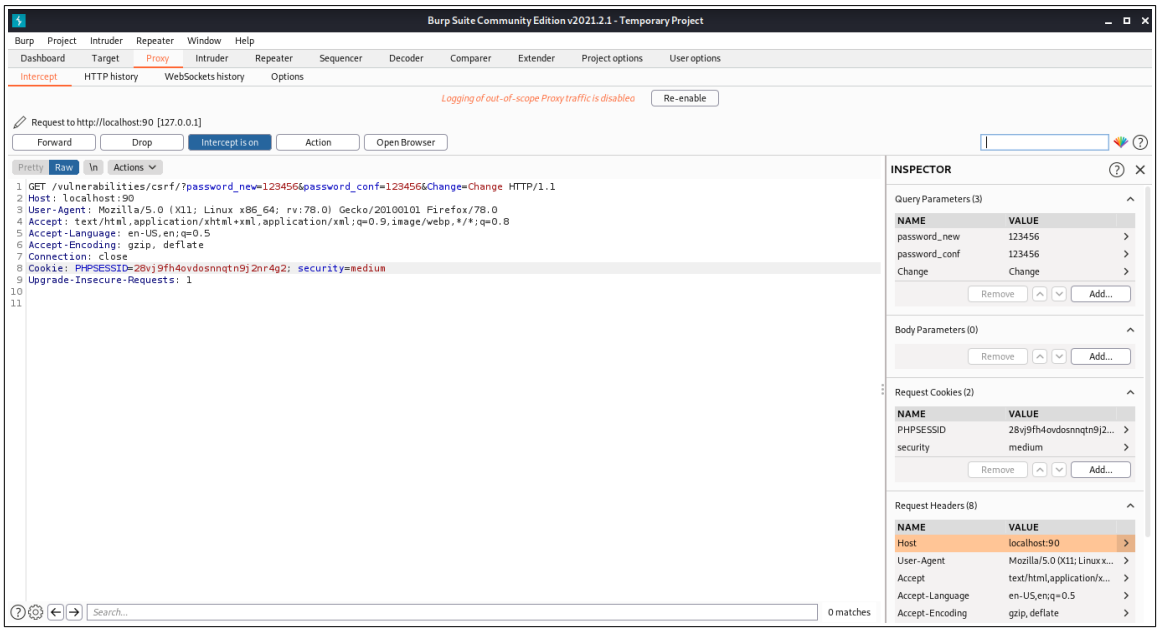


Now this file can be opened in Firefox,

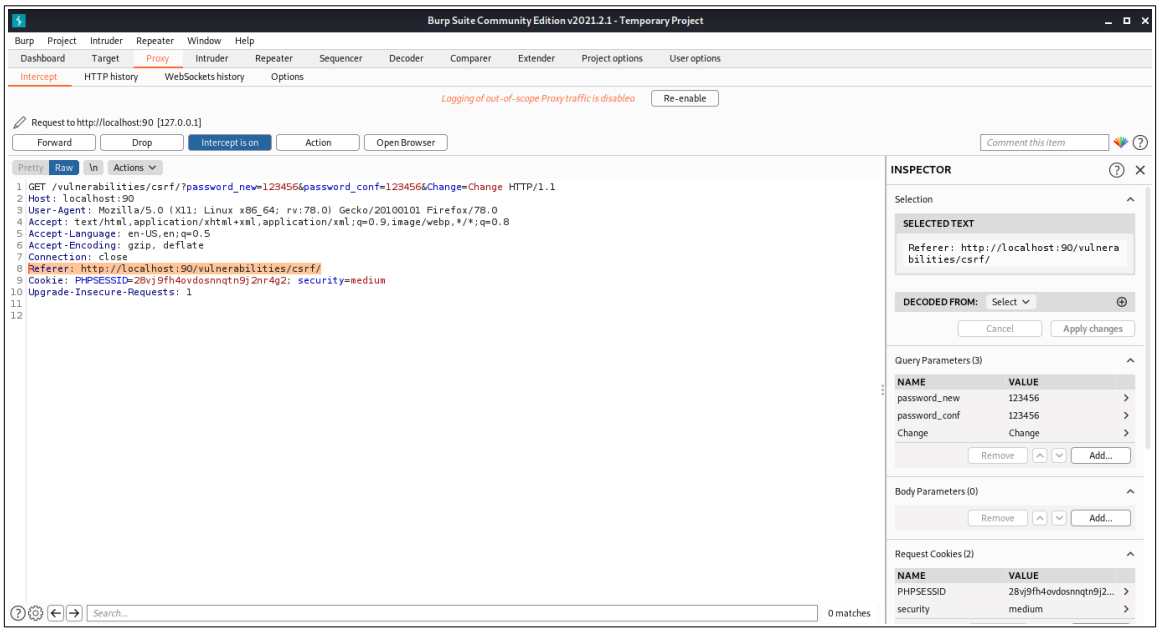


The **submit request** button is visible because we programmed the html file only for this purpose.

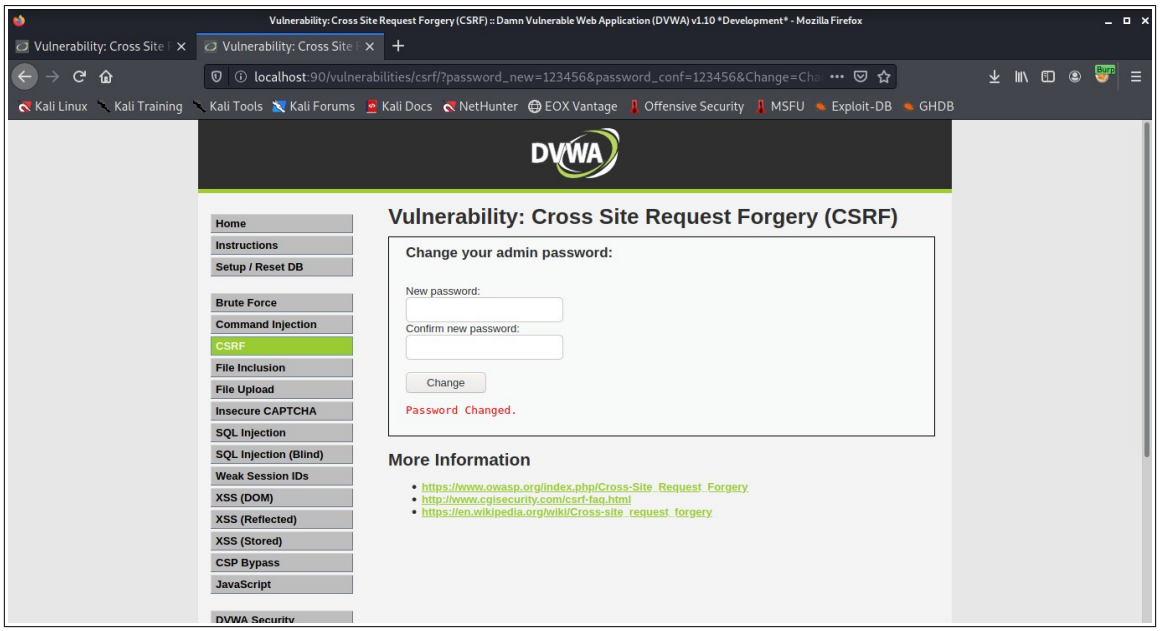
After clicking the submit button,



So here we can clearly observe that the **Referer** parameter is not there. The server gets to know that the source is not original so it won't allow to authenticate. To bypass this security level we can now refer back to the **Referer** parameter which was copied earlier and using burpsuite it can be copied to this intercept and forwarded,



The highlighted text shows the addition of the **Referer** parameter in this intercept launched separately from the system's home directory.



And now the password has been changed.

The CSRF attack was successfully executed and this hence proved the power of using **burpsuite**, in launching an attack from a different source.

But a point to note is that if this was an actual victim's browser, how can we intercept as the we did this attack on attacker's browser not the victim's one.

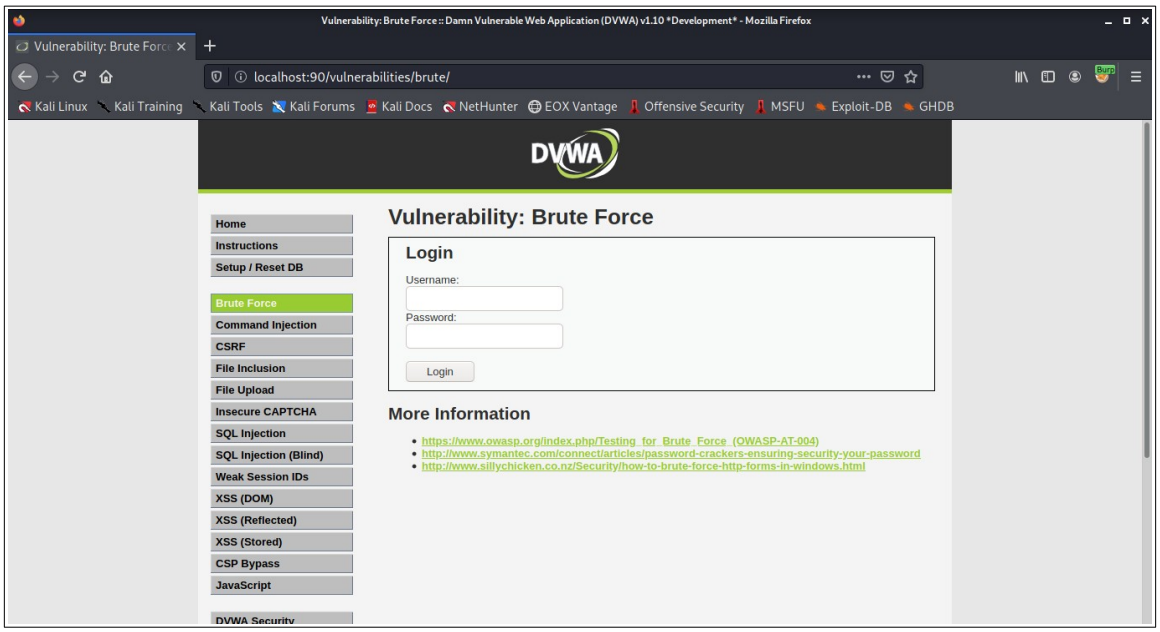
We cannot intercept using burpsuite on a victim's browser when the server is requested any action such as changing password as seen here.

Thus in this HTTP Referer won't work in the case explained point above and can be used to bypass in this case. There is a solution for this issue as well.

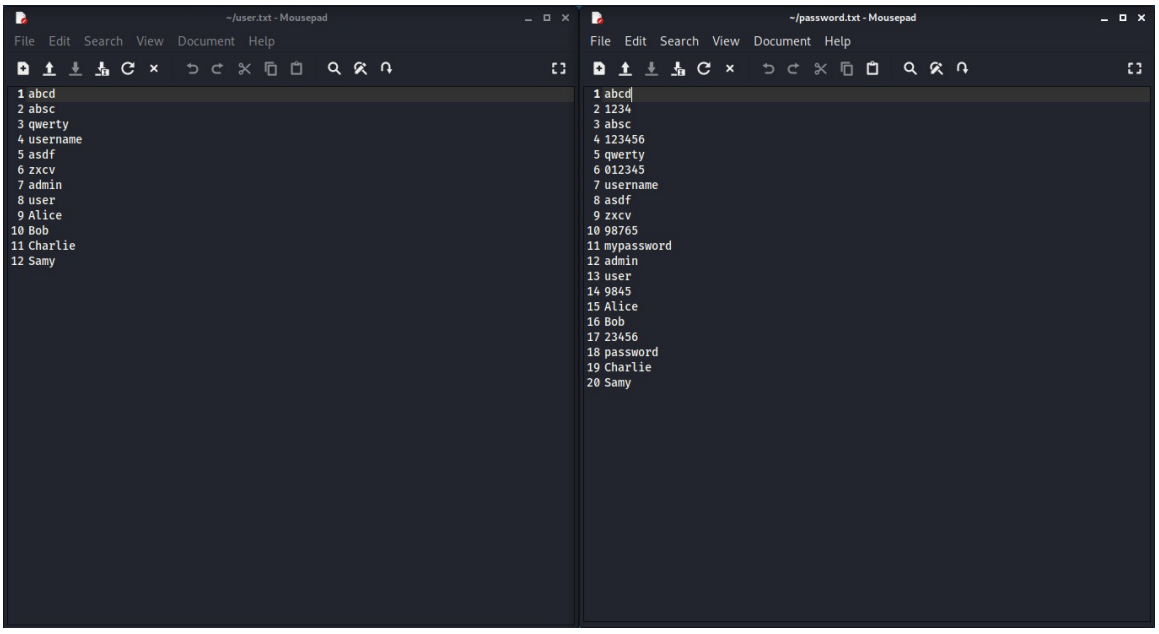
So to successfully execute this attack on a victim we need to perform session hijacking or we can merge with Cross-site Scripting attack (XSS) and more similar attacks.

Vulnerability: Brute Force

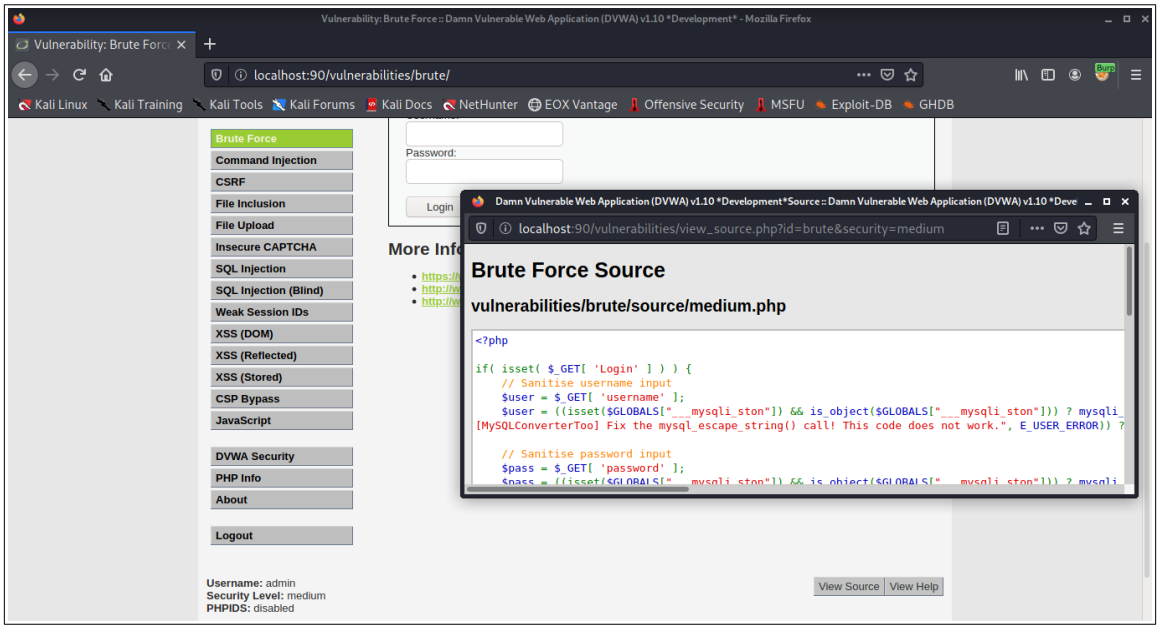
Firstly we will need to create two files for **Username** and **Password** respectively so that burpsuite can perform this attack. Creating a file or using an existing file with a list of potential credentials increases the probability of performing a successful brute force attack. A look at the webpage for Brute Force,



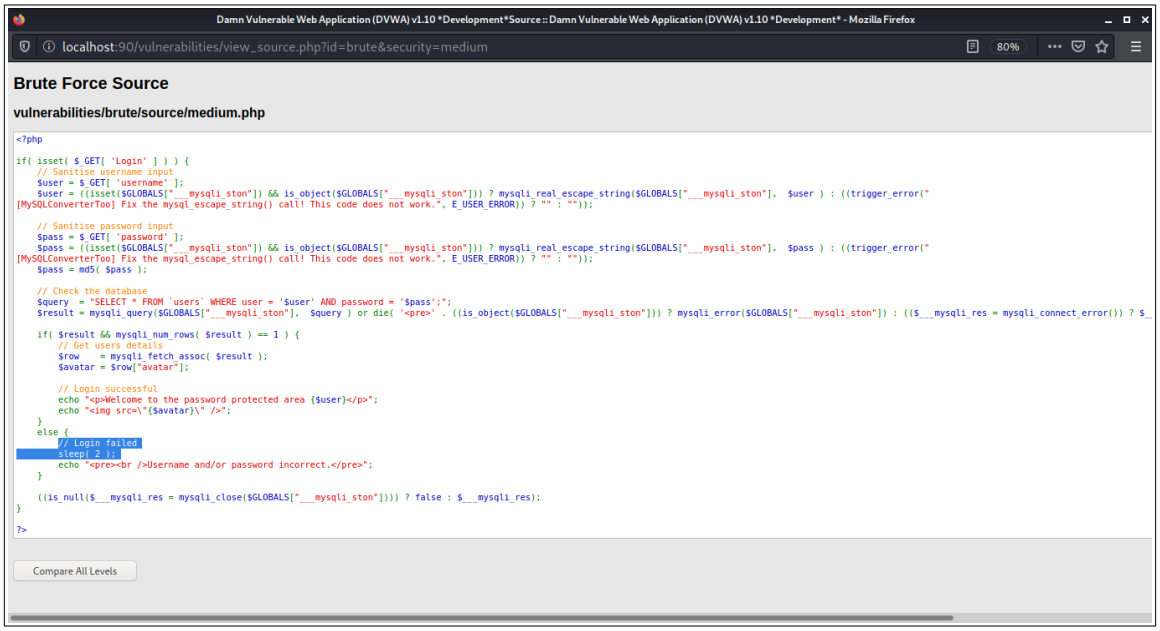
The **username** and **password** text files which will be used to perform the attack,



In the webpage, if we view the source code,

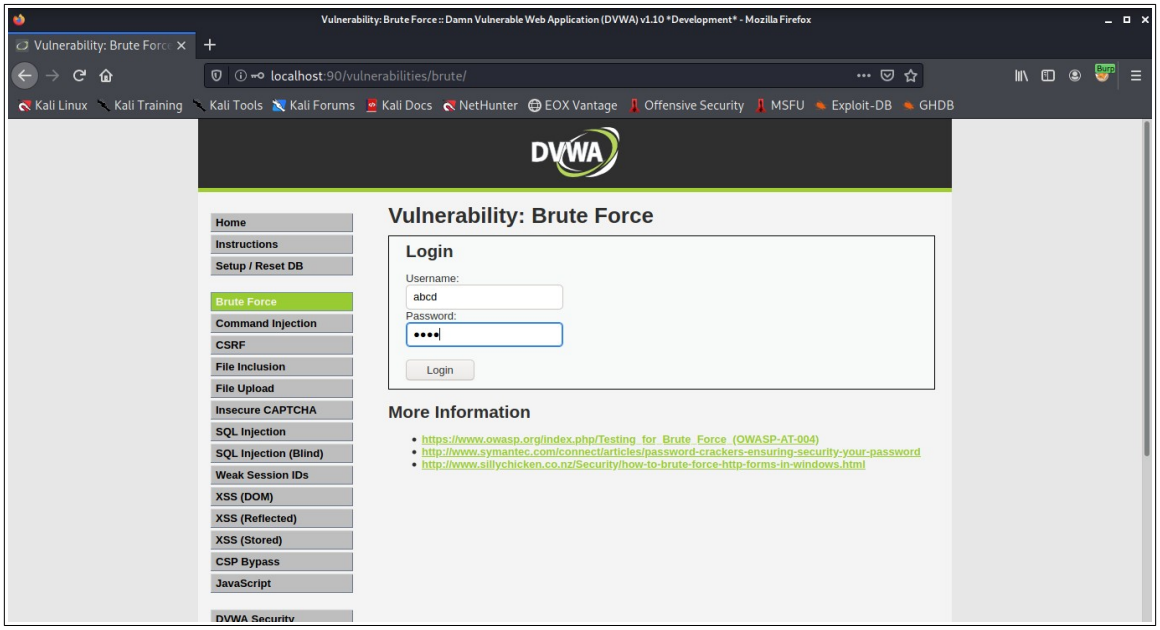


In the source code we can observe the Security feature which we have to tackle,

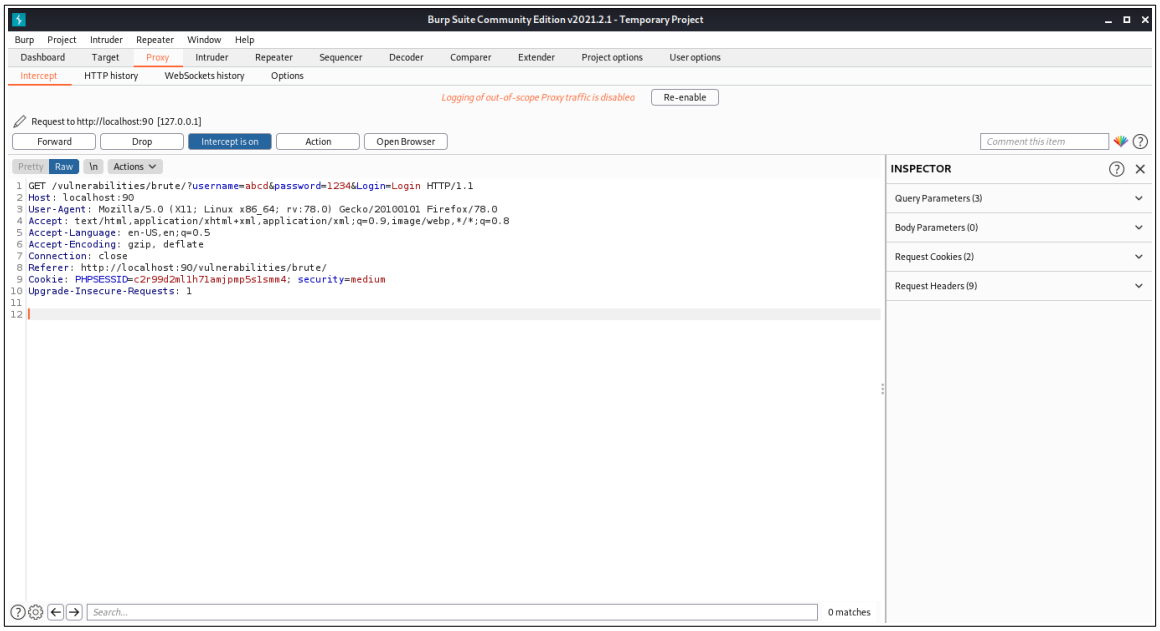


The highlighted text shows that if the login fails then it will will sleep for 2 seconds which means that while performing the brute force attack it will take some time to get the right combination of username and password as for every time the username and password possibilities from the text files shown above there will be a lag of 2 seconds thus this shows the security feature of the medium level in DVWA.

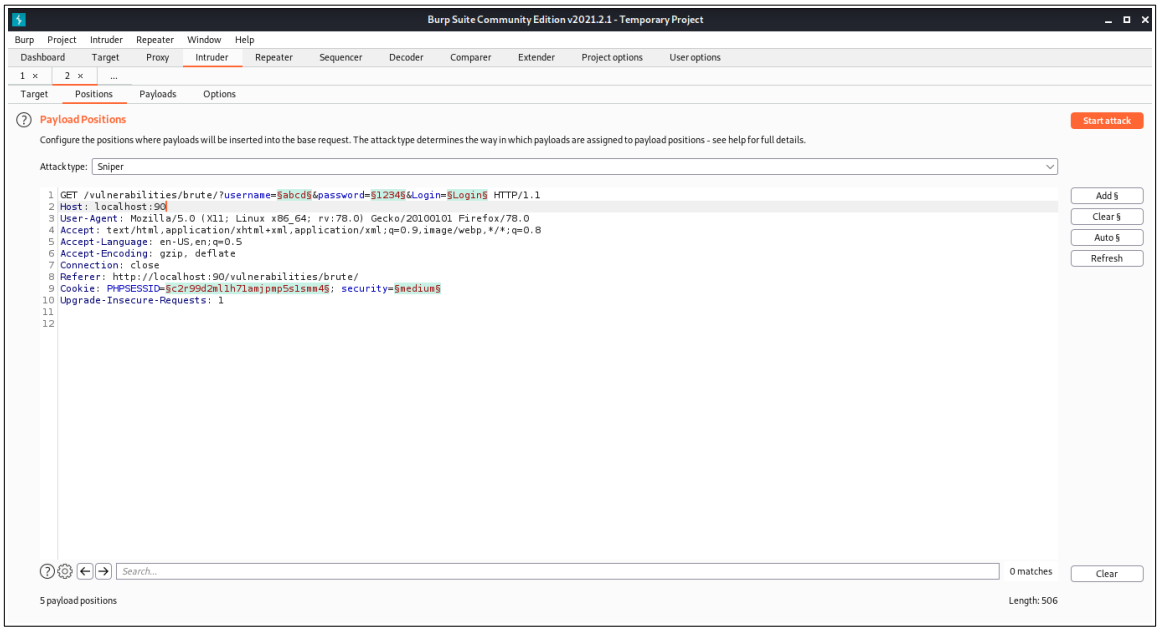
To perform the attack we can try any username and password combination as in this case we will be trying out with **username** as **abcd** and **password** as **1234** as we just a single intercept,



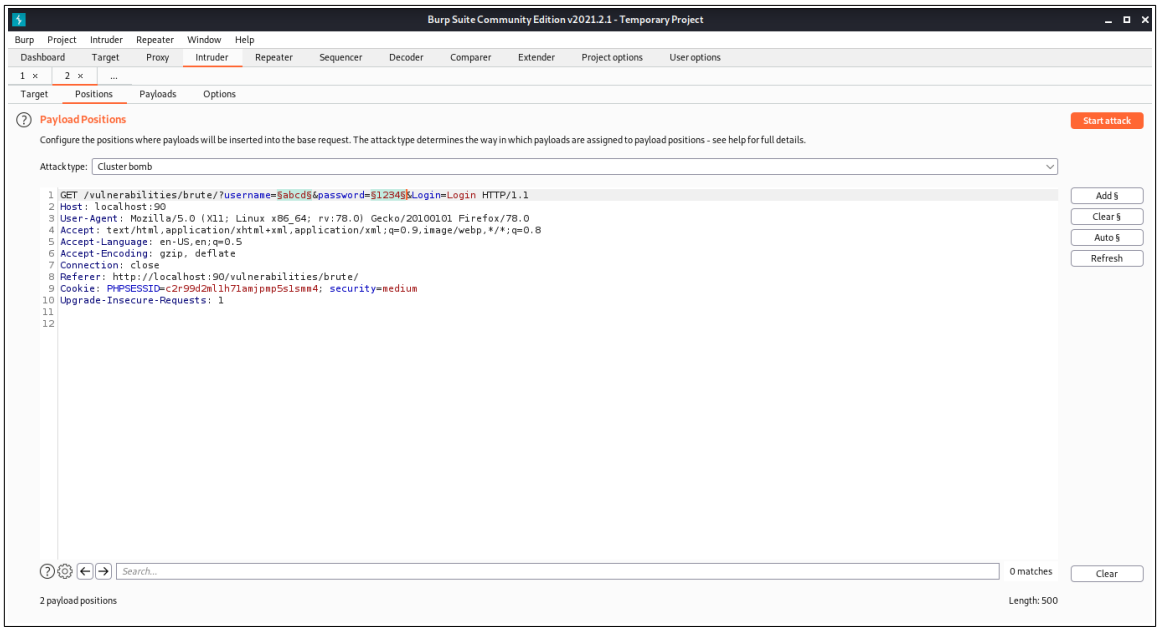
In burpsuite,



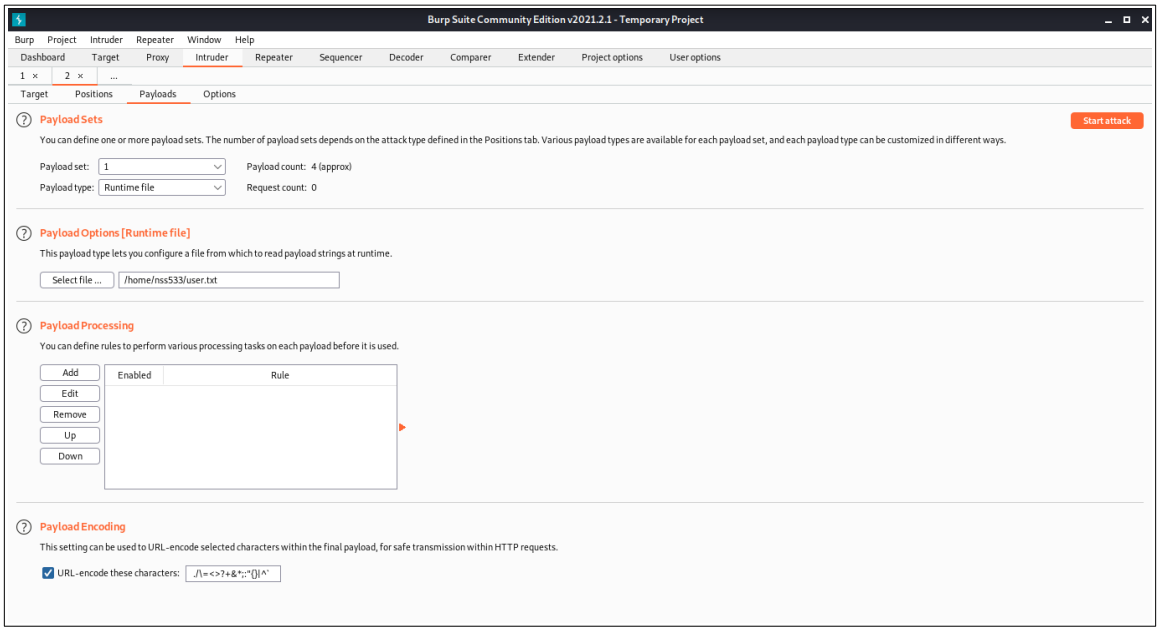
We send this intercept to the **Intruder** tab in burpsuite and turn the intercept off.



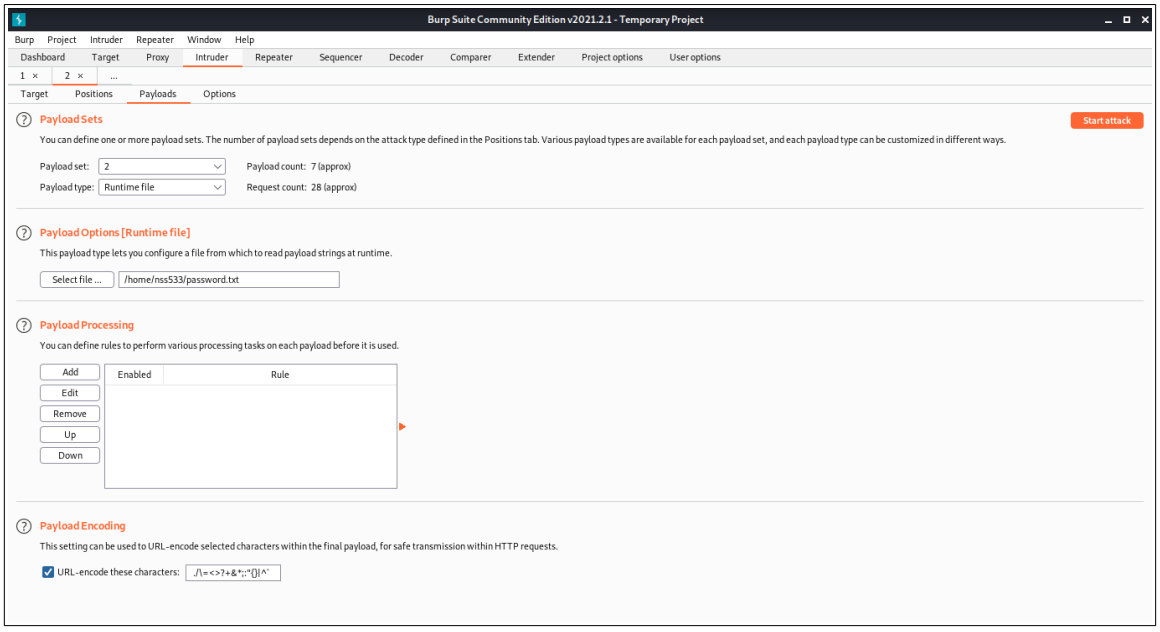
In the **Intruder** tab in the **Positions** sub-tab, we can see few highlighted texts, but we do not want to perform brute force on all these highlighted parts so we can hit the **Clear** button on the right to deselect them.
Now we can select the username and password field values(***abcd*** *and* ***1234***)



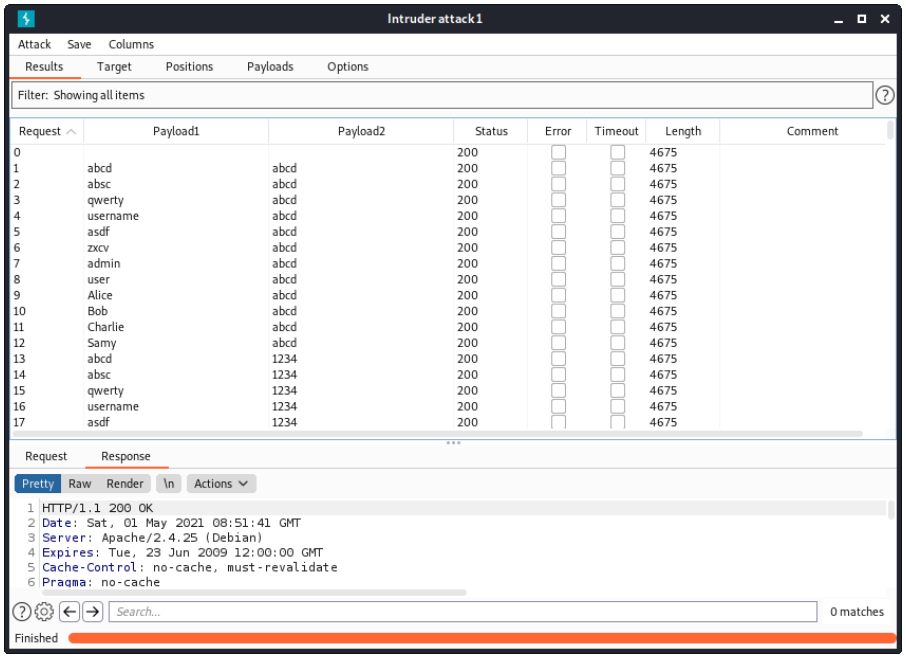
After selecting the username and password field values we need to select the attack type and **Cluster Bomb** attack type is used as shown above, since there are two files with potential usernames and password for burpsuite to work on.



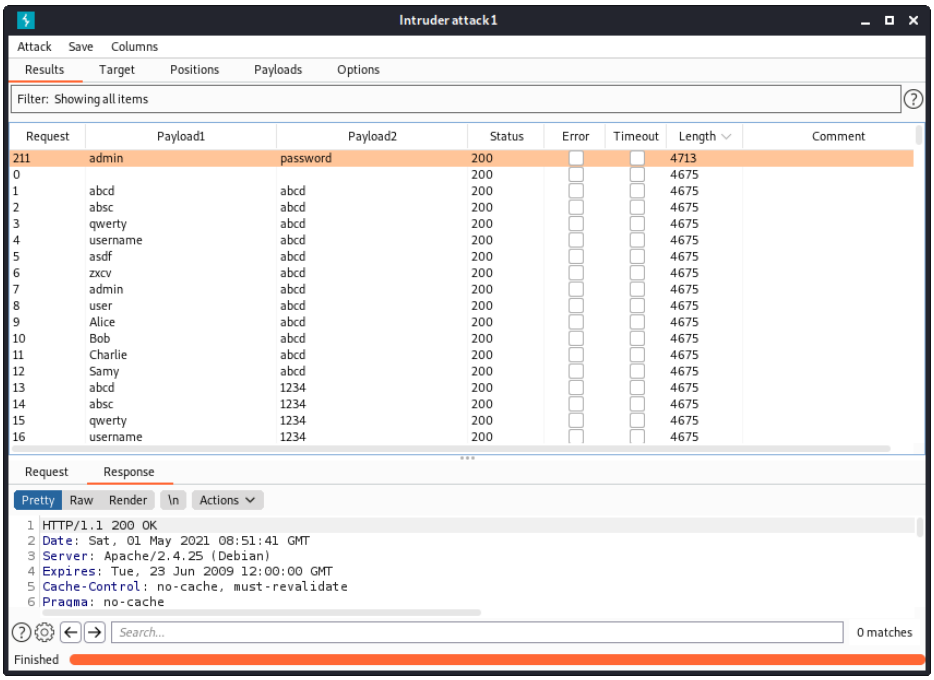
For this we select **Payloads** sub-tab and select **Payload 1** in **Payload set** option and choose **Runtime file** in **Payloadtype** and select the usernames file. This will be the same procedure for **Payload 2** with respect to the password file as shown below,



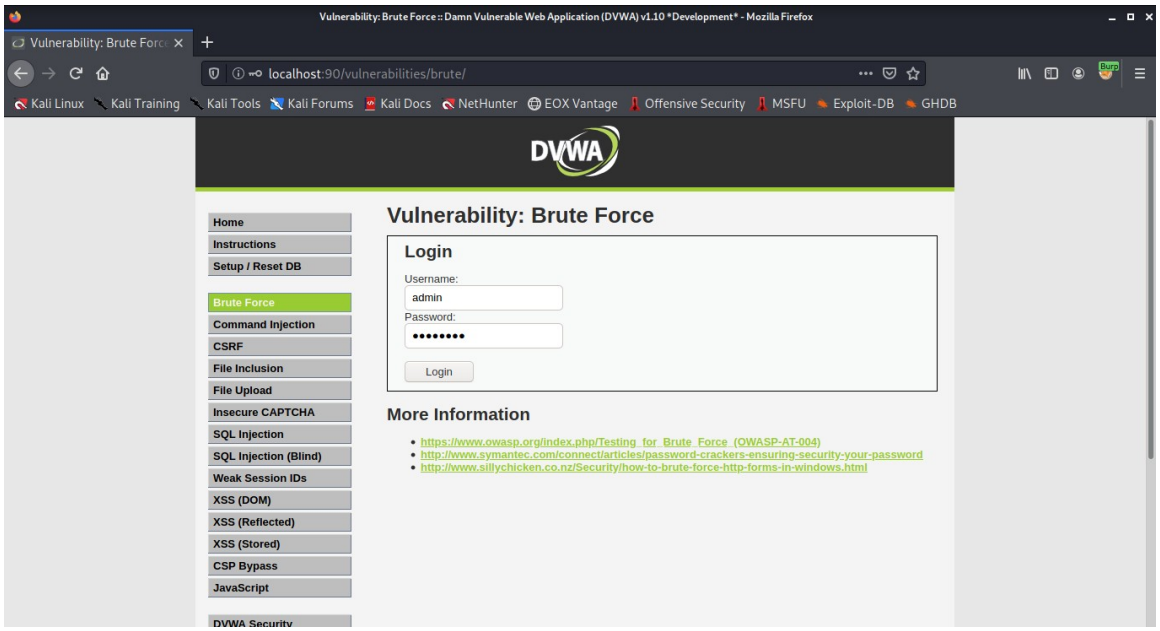
Finally, click the **Start Attack** button for burpsuite to perform the attack.
This attack will take some amount of time due to the **sleep()** observed earlier in the source code.



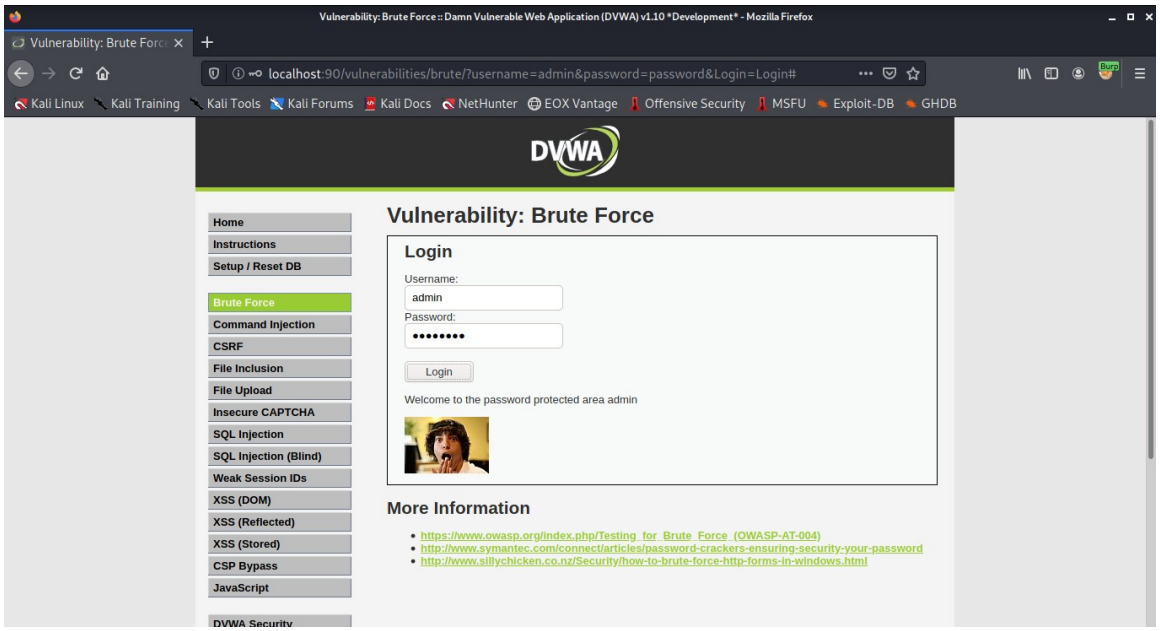
To know which combination of username and password is correct we observe the **0 request** as we need to observe the length of this **0 request** and which ever combination of **Payload 1** and **Payload 2** results with **length more than the 0 request row** is the correct credentials. Once we order the Results by the length we observe the correct credentials,



Now lets try this combination,



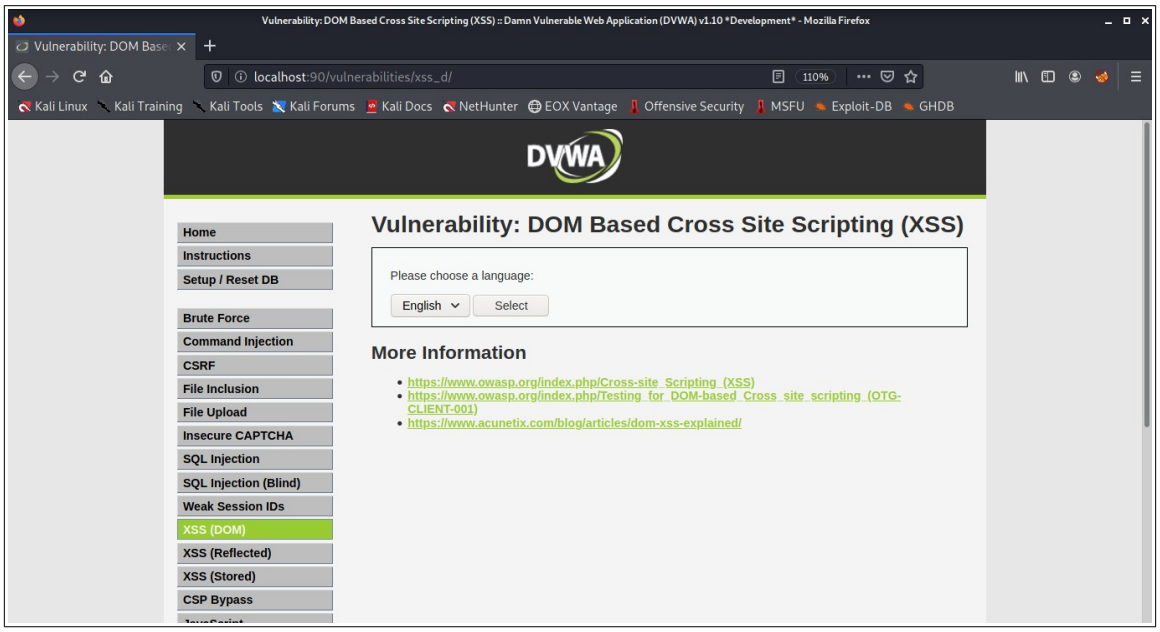
The combination turns out to be the correct combination of credentials,



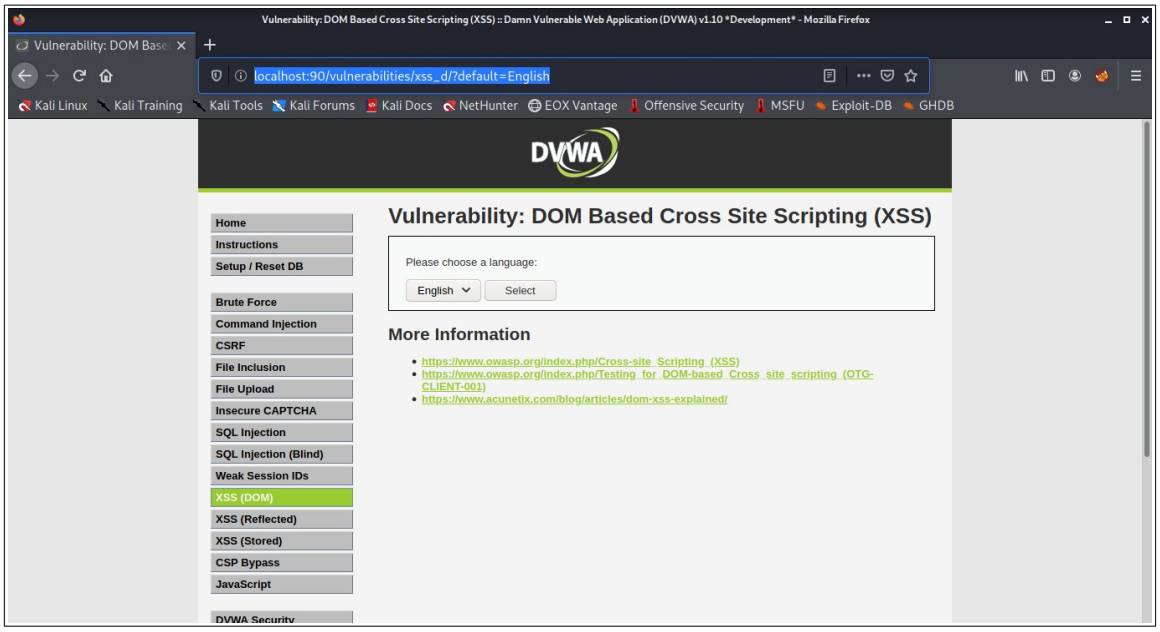
Thus the attack was successful though it took some amount of time.

Vulnerability: DOM Based Cross Site Scripting (XSS)

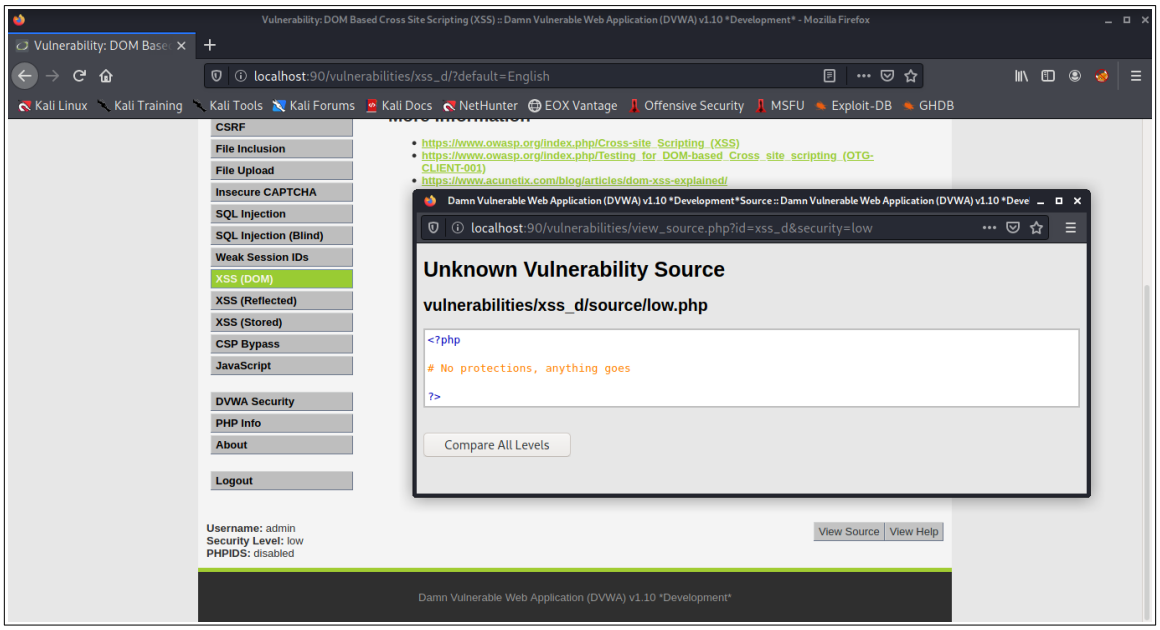
A look at the webpage for this attack,



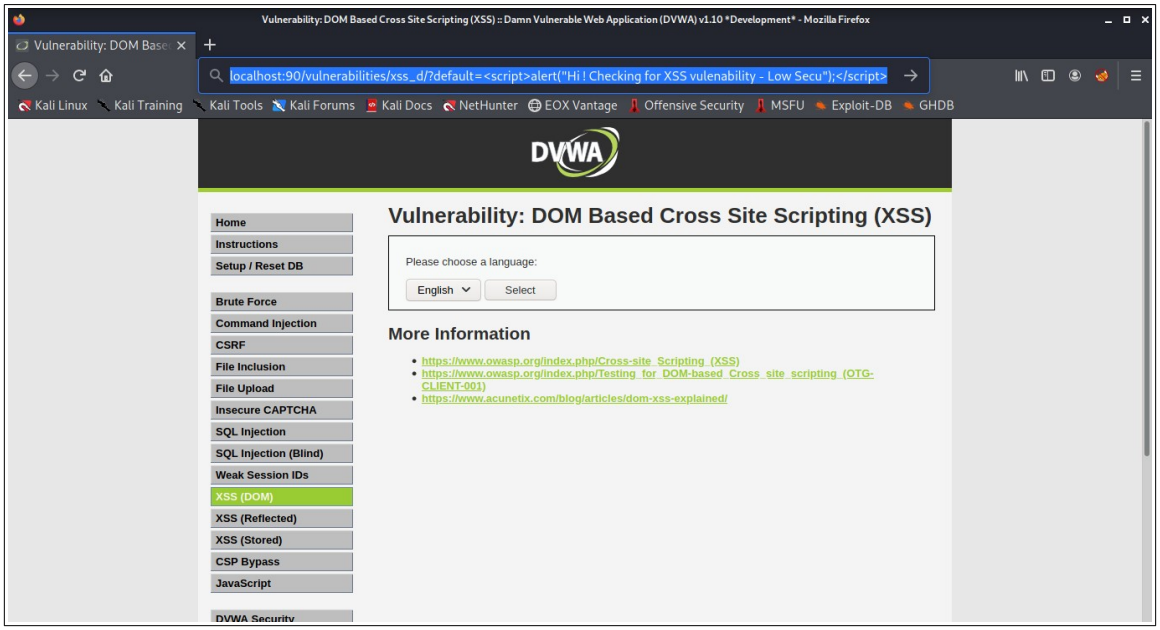
First, lets do this in low security.
So by selecting any language from the drop down as shown in the above pic, the changes will be observed in the URL section of the website,



Now we can tweak around in the URL section by introducing it with a malicious JavaScript. But lets have a look at the source code to see its vulnerability,



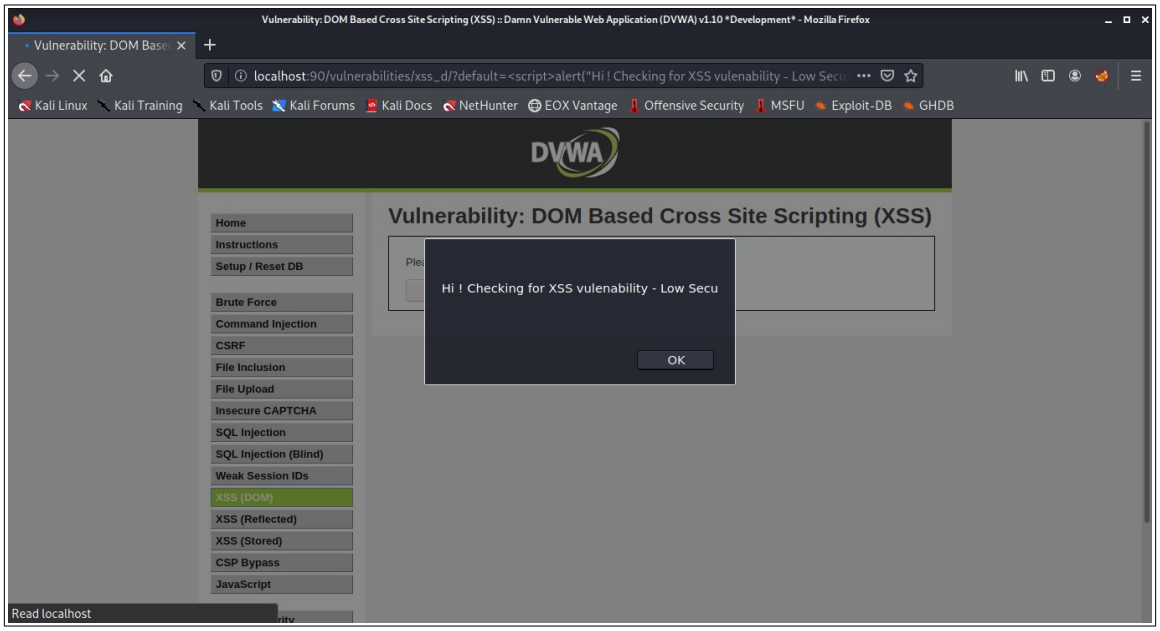
It can be clearly seen that there is no protection. Now it will be very easy to put in a simple payload say a JavaScript alert pop-up.



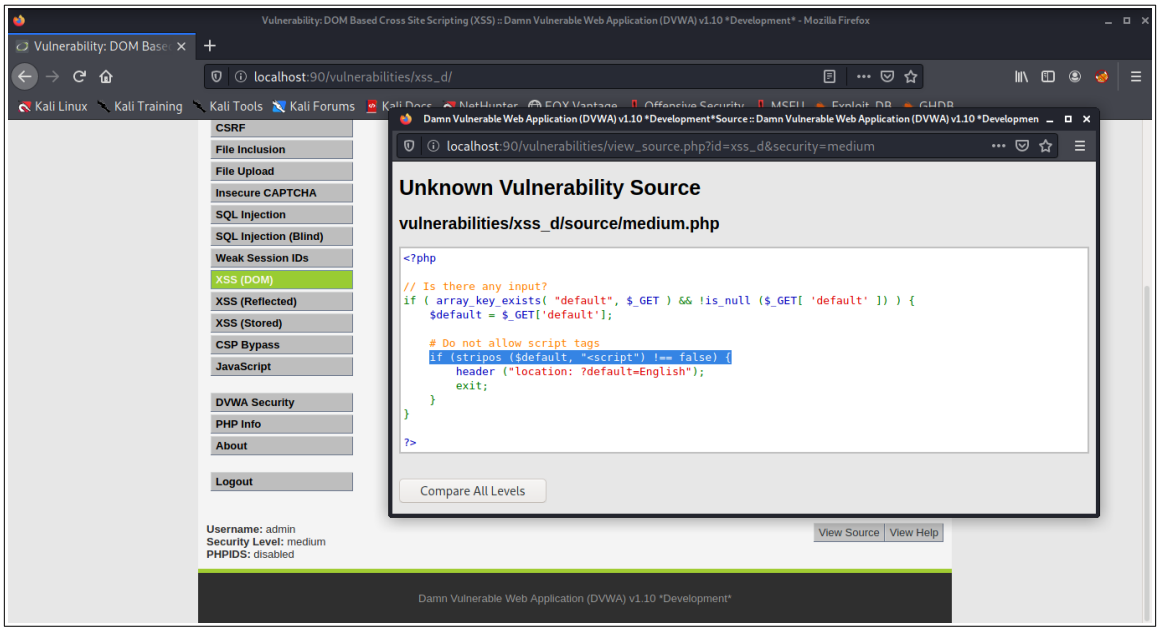
The JavaScript.

<script>alert("Hi ! Checking for XSS vulenability - Low Secu");</script>

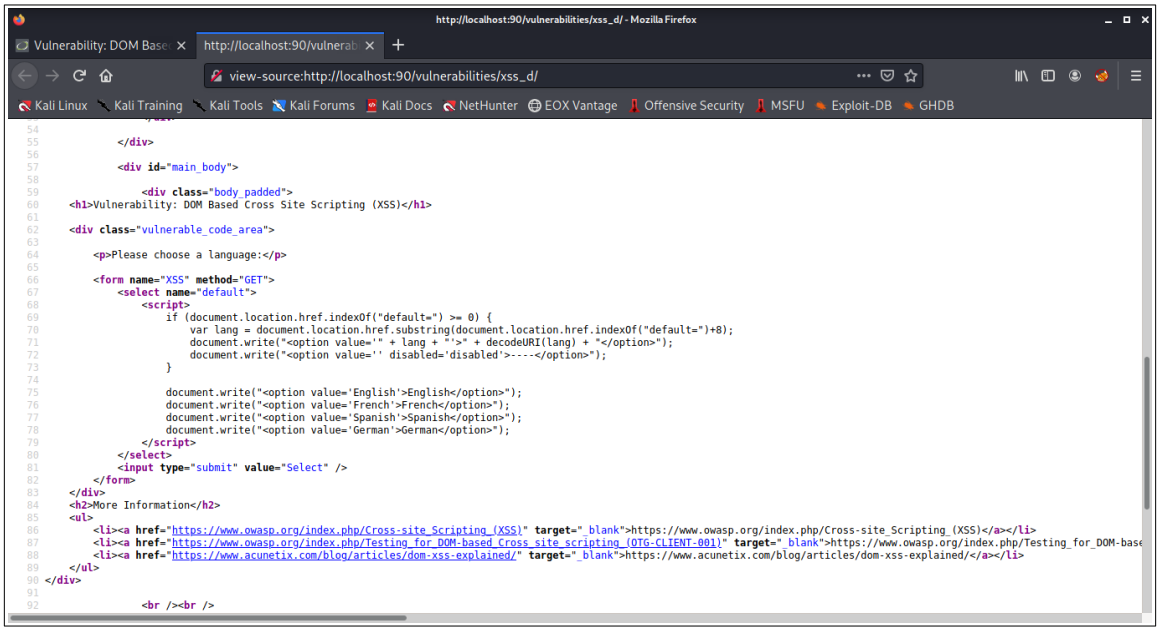
After clicking Enter, we should be able to observe a pop-up,



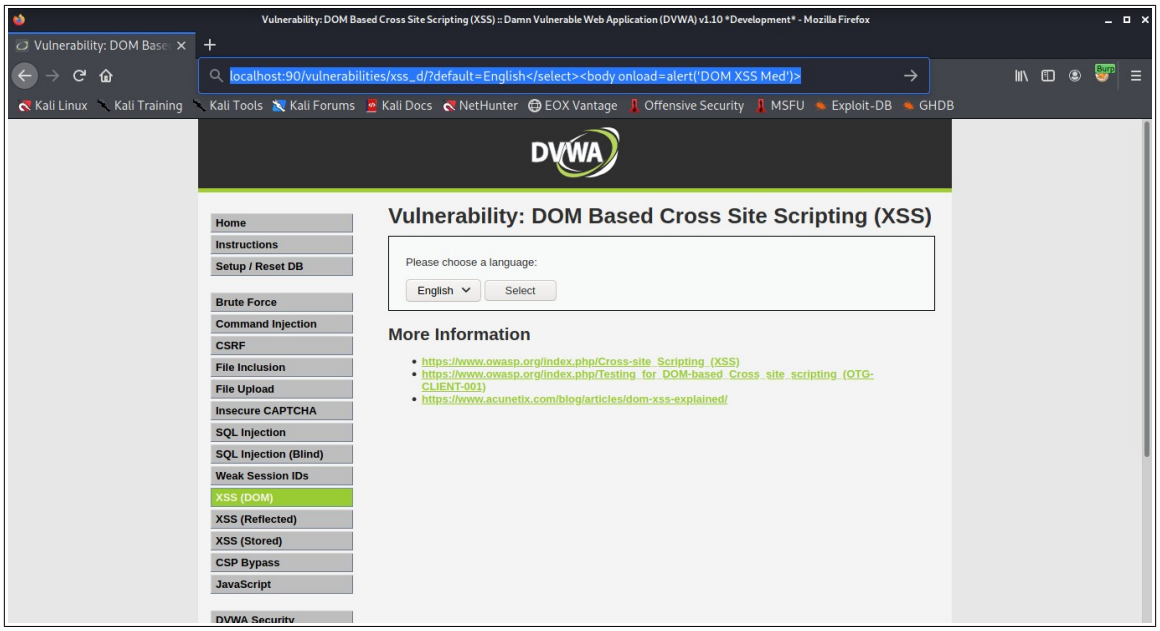
We can clearly see the pop-up with the contents of the script which shows the existence of an XSS vulnerability.
Now lets try this again with Medium Security,
First lets have a look at the source code,



Here we can see that the code is filtering any script tag introduced in the URL which mitigates the vulnerability which was seen previously thus not executing such URL's.
So lets look at the source code of the entire page source,

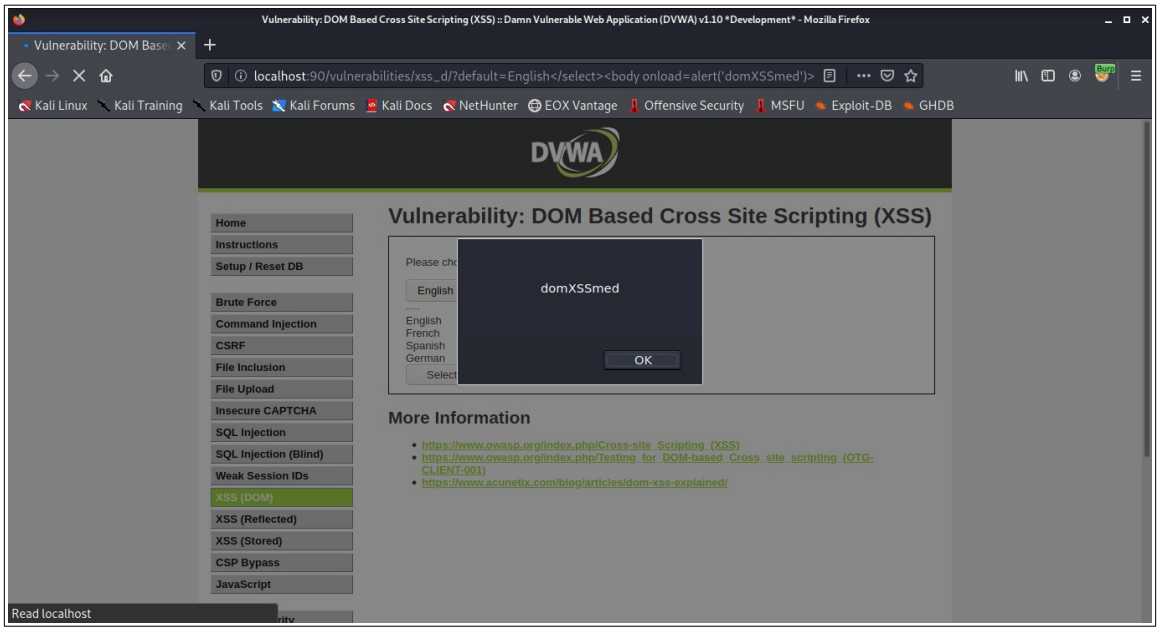


Here it can observed that the script tag is put after a select tag. So we can execute the XSS attack if the select tag is does not exist before the script. (*These are all present in the main body tag*)

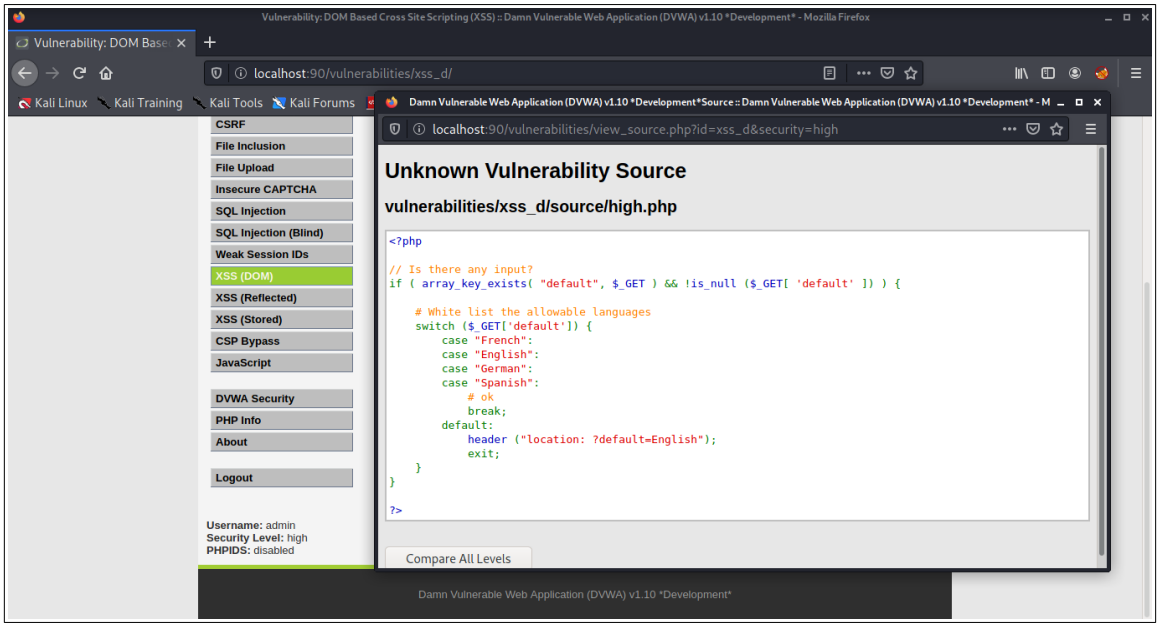


Here, if we close the **select** tag and create an **onload payload** the XSS attack should be successful.

</select><body onload=alert('domXSSmed')>

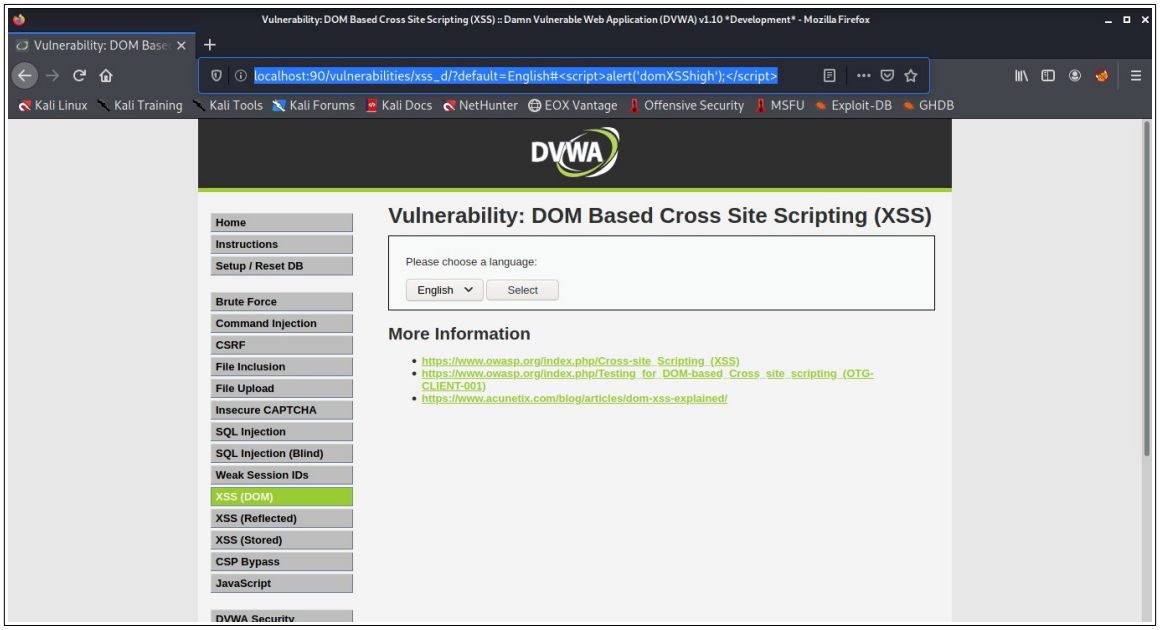


Thus the XSS attack was successful.
Let's try this out at high security as well.
Let's look at the source code,



Here we can see that the server is white listing only the 4 strings French, English, German and Spanish respectively.

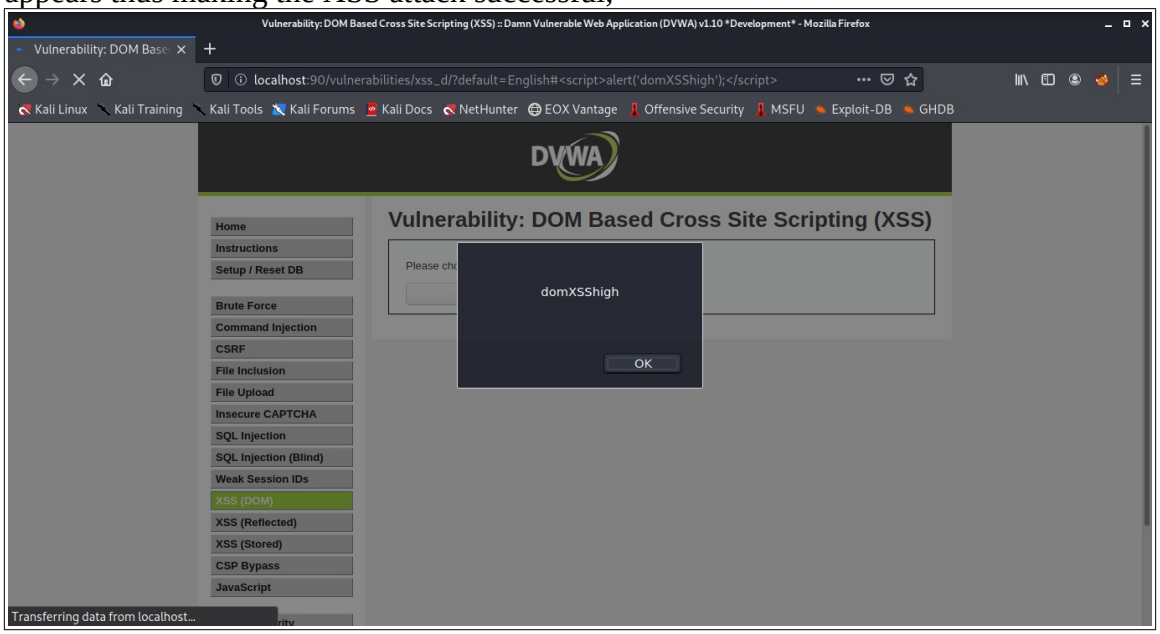
So here we can use the '#' while introducing the script in the URL as the '#' will not submit the code after it to the server and the script tag will also have to be taken into consideration along with the filtering of the 4 strings seen earlier.



In the URL after the **default=English** we can place the code,

#<script>alert('domXSShigh');</script>

After clicking Enter there will be no effect, but after reloading the page the alert pop-up appears thus making the XSS attack successful,



Closing Notes:

In this Bug Bounty Assignment, the DVWA website which had many other most common web vulnerabilities and with various levels of difficulty, the following vulnerabilities were tested and solved,

1. **SQL Injection**
2. **Cross Site Request Forgery – CSRF**
3. **Brute Force**
4. **Cross Site Scripting – XSS**(*DOM based*)

Since this website was hosted locally (***http://localhost:90)*** as discussed briefly in the beginning, there was no harm or damage caused to any person or organization with respect to the creation, distribution and utilization of the resources needed during the conduction of this **BUG BOUNTY** assignment.

The above listed vulnerabilities were selected completely with accordance to the knowledge gained from the university syllabus and few online resources only for ethical usage/educational purposes.

Nishanth S Shastry
B.TECH CSE
6th Semester
PESU