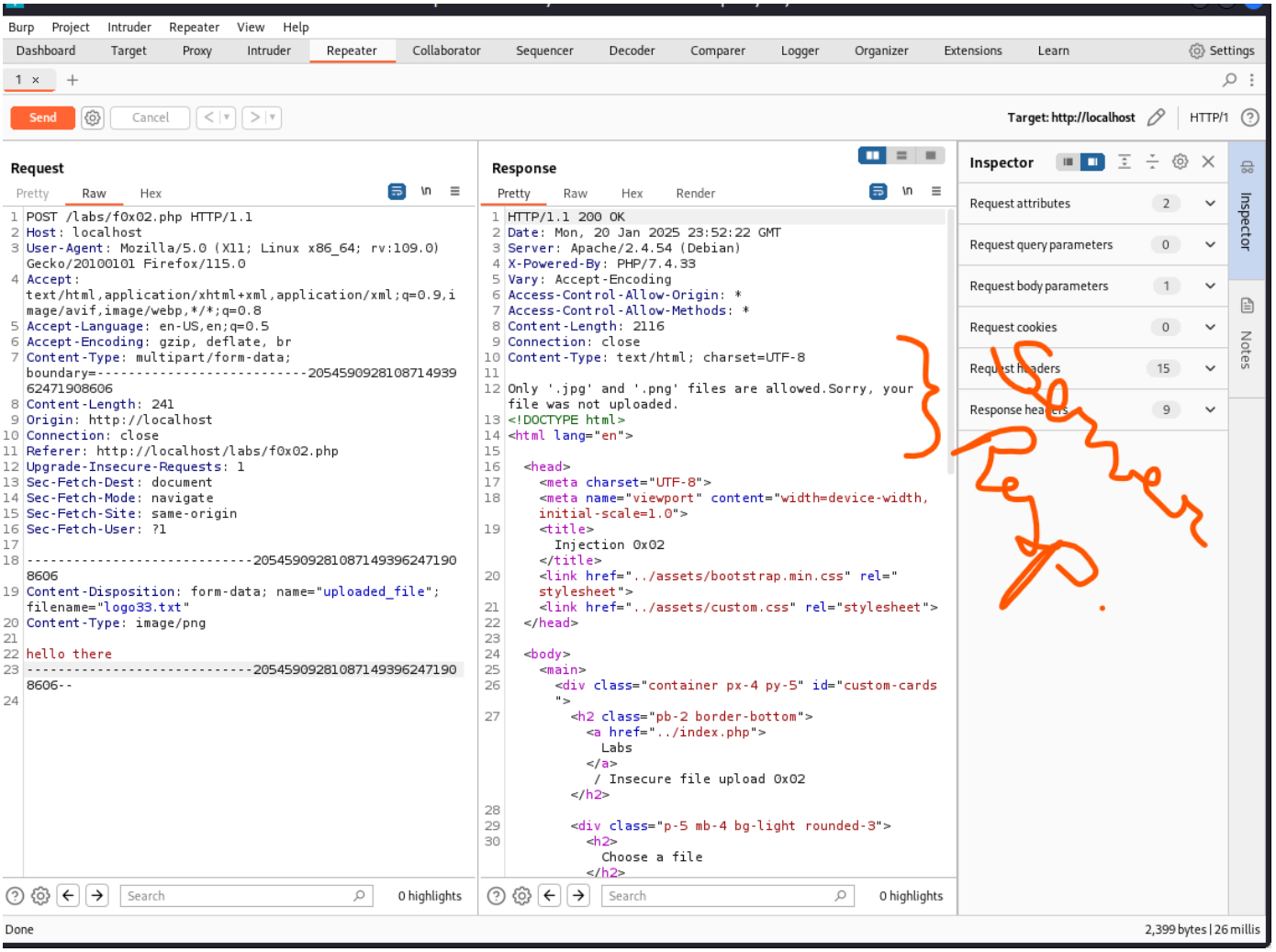
15 - Insecure File Upload - Magic Bytes

Here, the check is happening client and server side.

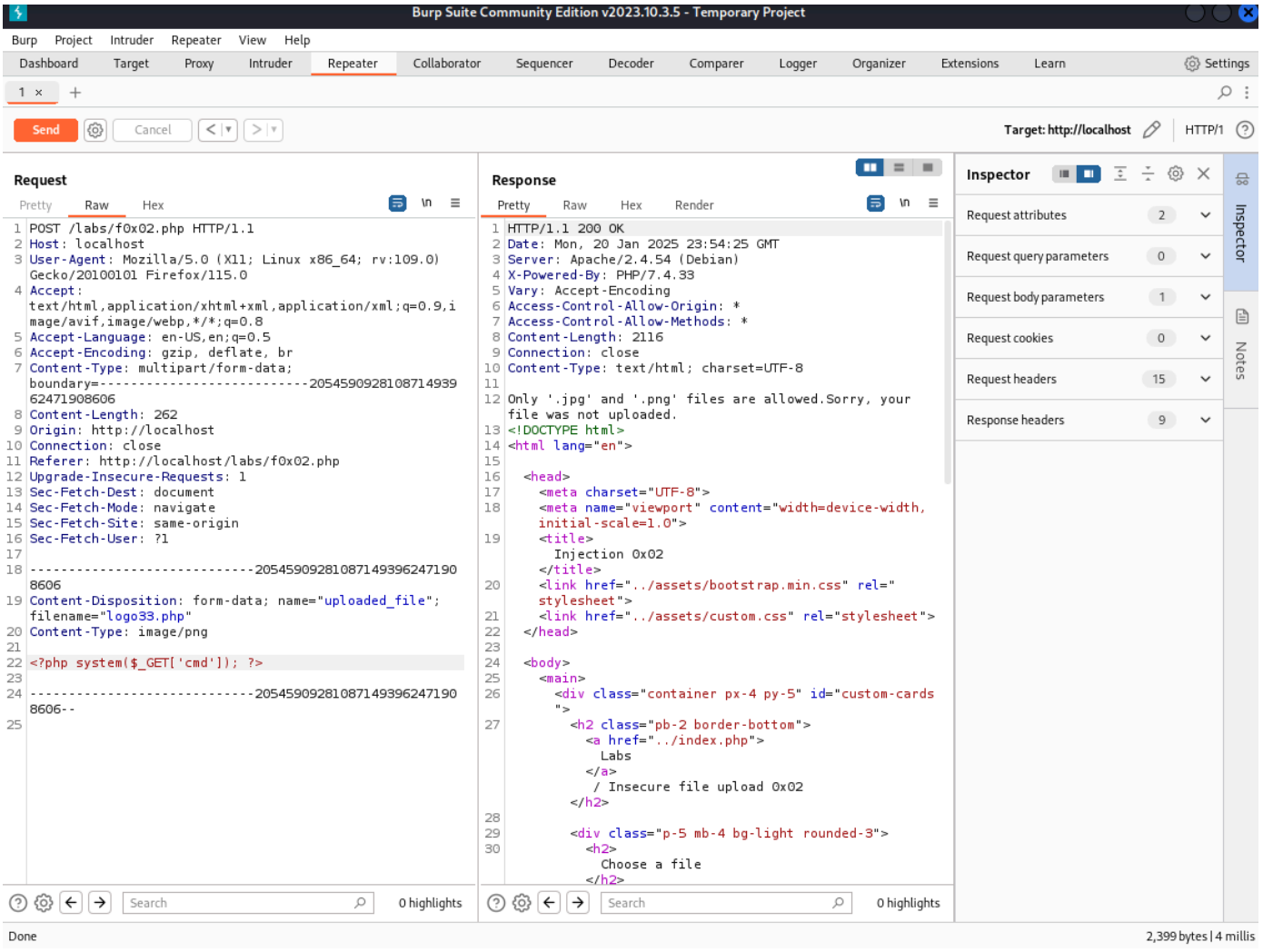
The process is going to be the same. We are always building upon what we know.

We see normal behavior and build on that.

Here, when we try to upload a txt, server tell us we cant.



Same for php.



Looks like we are going to be using magic bytes.

The filters range from checking the extension at the end of the file to actually reading the file to ensure it is actually the extension expected.

The instructor list a couple of options here:

-Adding a null byte and then the expected extension to the end of the file name: ".php%00.png"

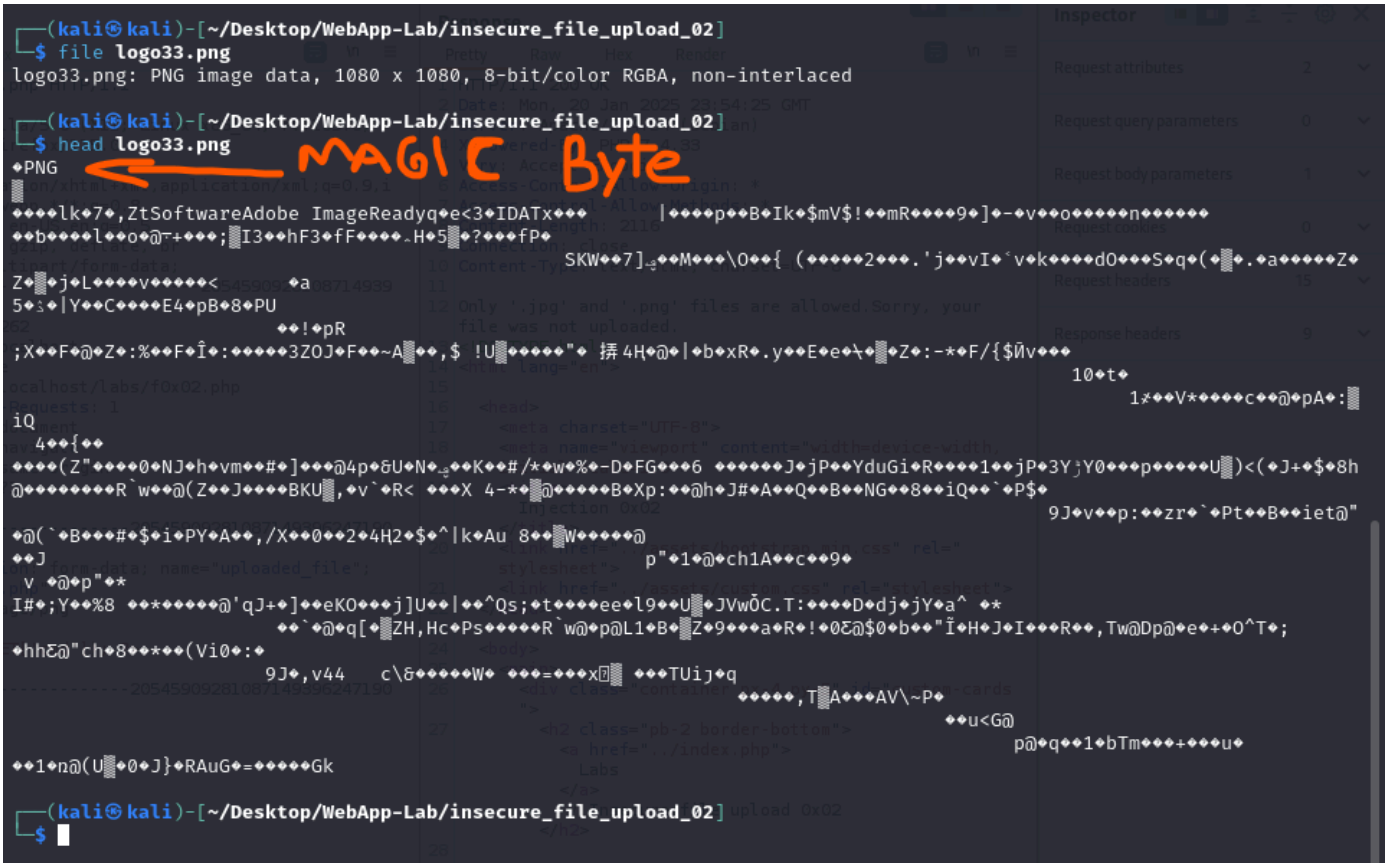
-"If an application is not configured correctly, for example the "htaccess" file is configured incorrectly, sometimes we can get execution just by adding the expected extension without the null byte: ".php.png"

-"The other way is to upload a htaccess file that allows us to execute files like ".asd" as if they were ".php"

We can check "Content-Type" parameter although it cannot impact how a file is stored or executed on the target server, so it is not worthy.

We can also check the magic bytes, which is going to tell the system what type of file it is. It is the first few bytes of a file. If we were to go to our kali Linux and use the "file" command to see more info on a

file, it would return the file type along with some other information as the type of file, the size, and some other info that is going to depend on the file type. Now, if we use the command head, we can see that the first few bytes identify the file as PNG. Lets take a look.

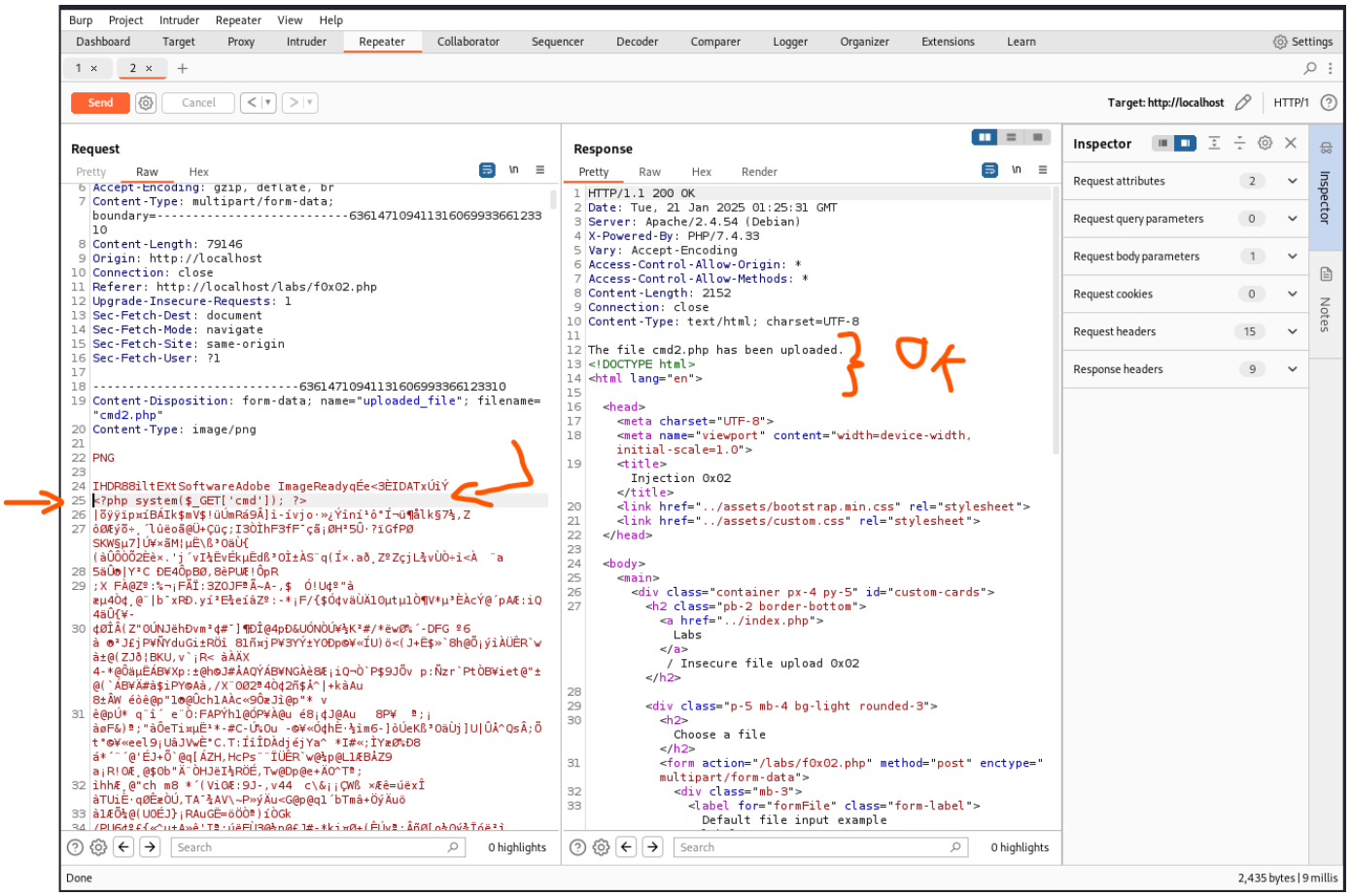


We can check online for a list of file types and its magic byte.

Now, we are going to insert our payload in the image we are sending, like it was embedded in the file.

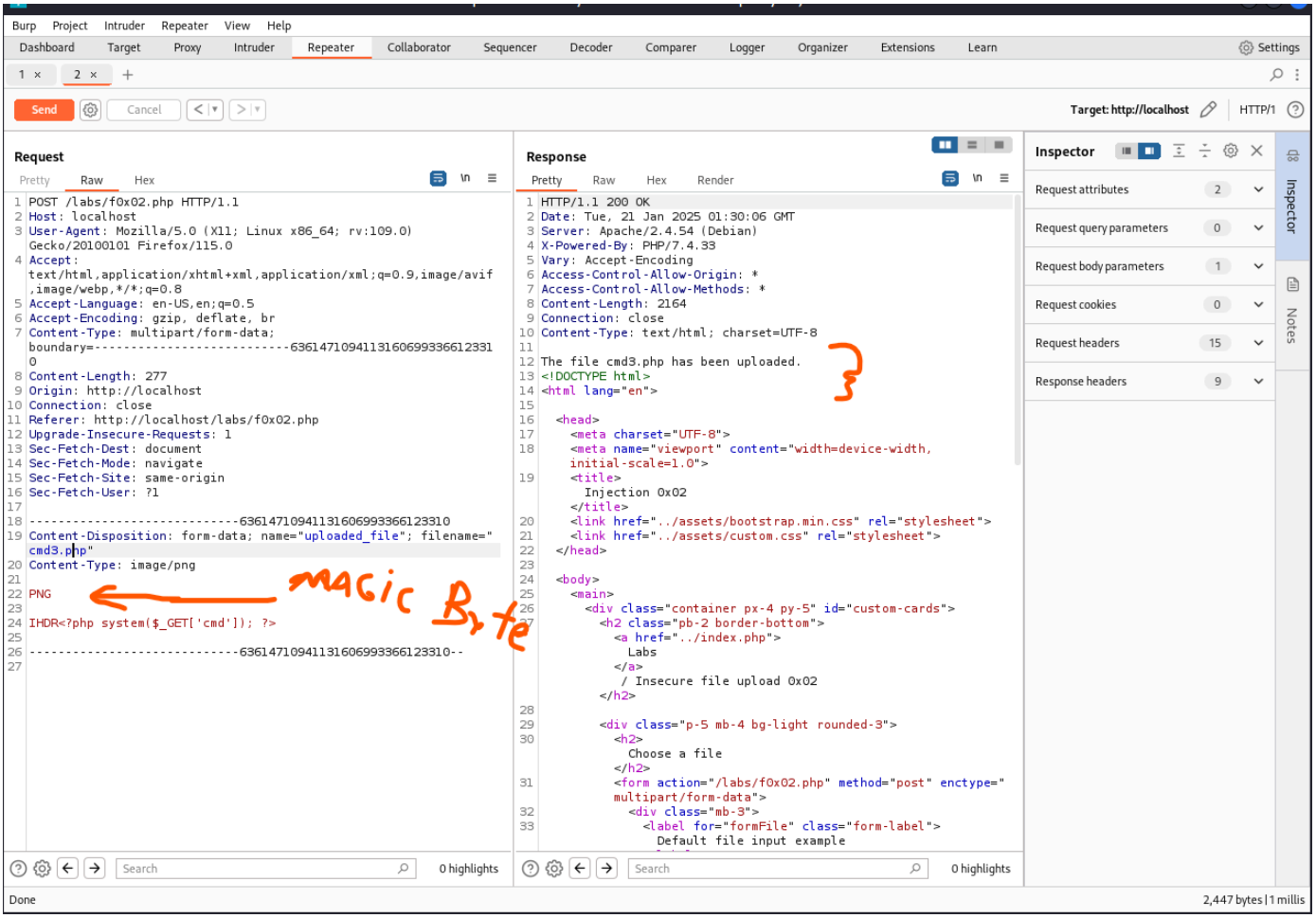
Check it out:

Go back to the .png file we successfully uploaded. In the image data, are going to put our payload between the first line of image data, and the second line of image data. And, we will need to change the file extension as well, so the website executes it.

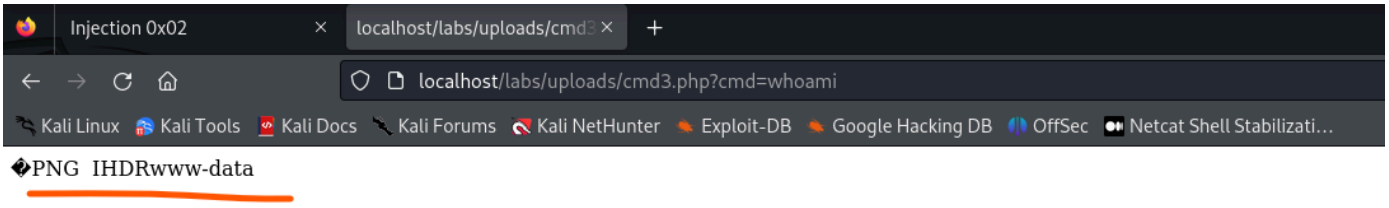


But, it still gives us errors.

Lets take out most of the image data.

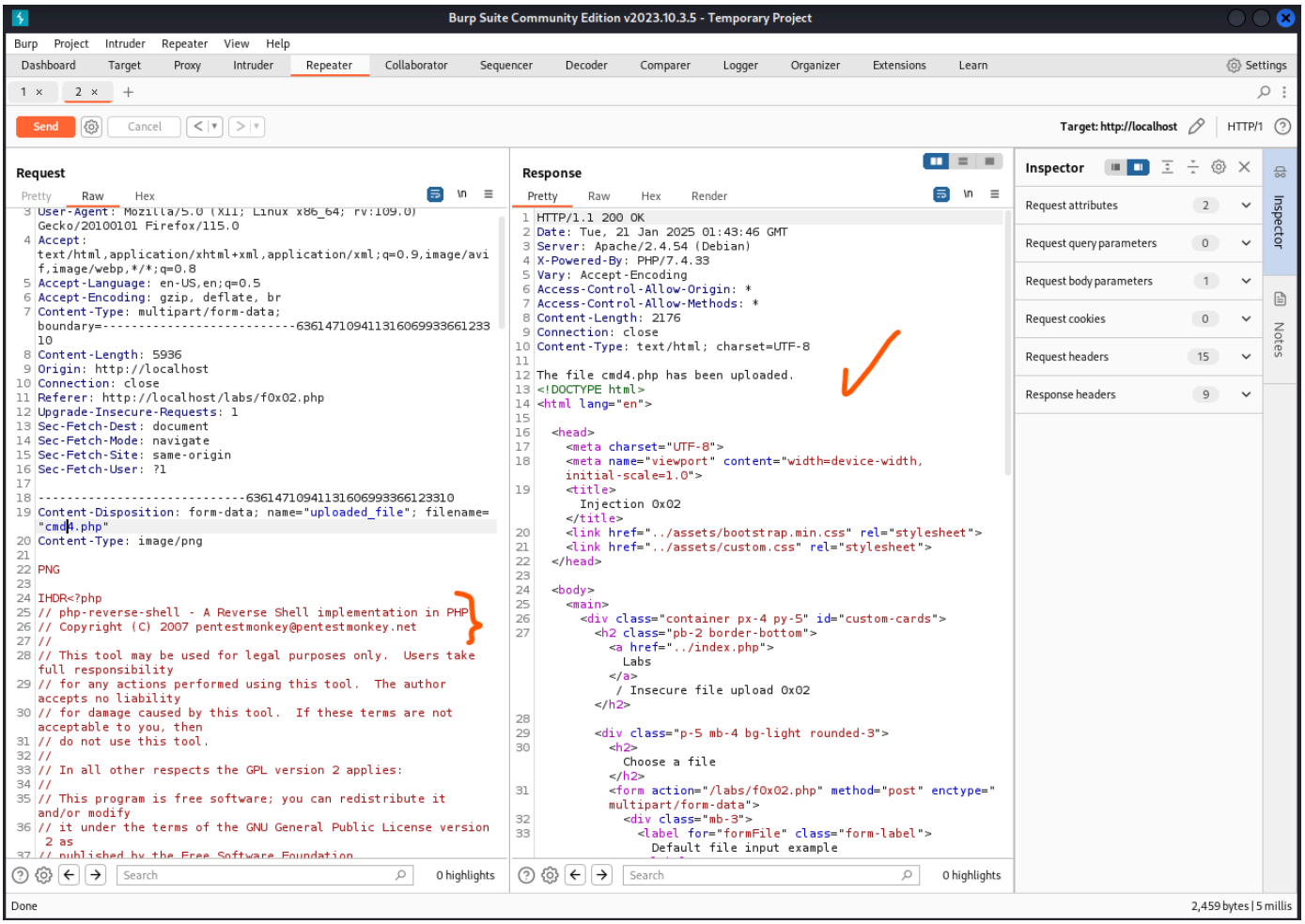


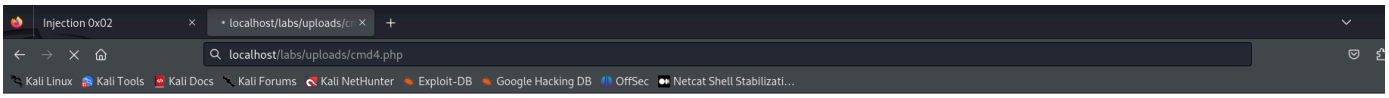
lets see if it works.



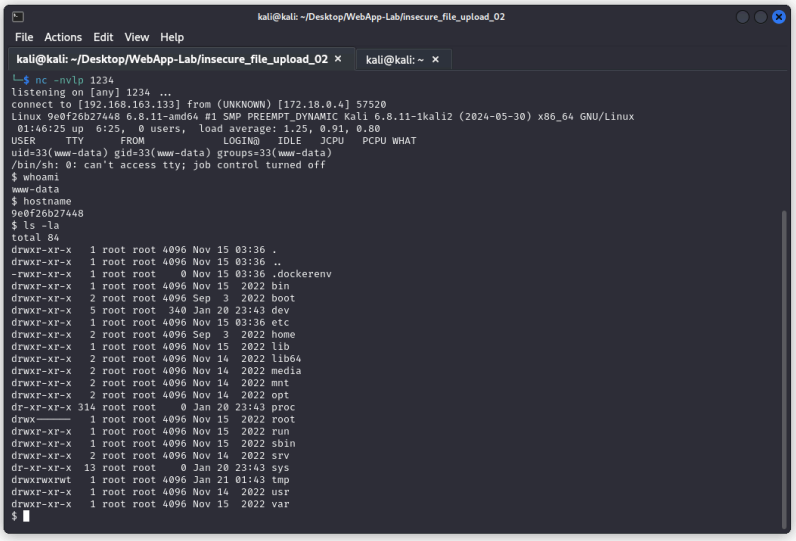
It is not really pretty but it works.

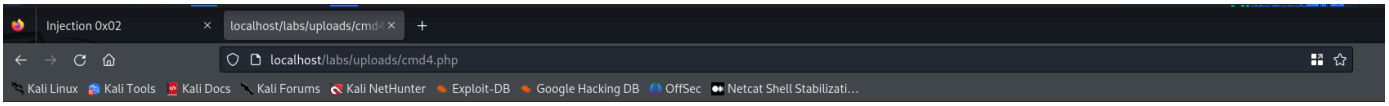
Just substitute the php code with the rev shell from pentest monkey, and be happy.





PNG IHDRwww-data





PNG IHDRWARNING: Failed to daemonise. This is quite common and not fatal. Successfully opened reverse shell to 192.168.163.133:1234 ERROR: Shell process terminated

Now, if there was a block list blocking .PHP files, we could also use other file extension.

If we google "valid php file extensions", we can see there are a couple options. We could perhaps upload a file with one of these extensions, and have the website execute it when we request the URL.

Here, the best secure idea would be to use an allow list rather than a block list.