Vault Command and Control Server [350]

Task description: ./description.txt

TL;DR: LFI previously uploaded PHP script with a “different” opening tag

This task is linked to the previous one in this category, **Bootleg Iodine**, and cannot be solved without it. Solving the first task gives us the link for this web challenge.

We find an aparent minimalistic website, with an upload functionality. First thoughts: upload a shell and find the flag. Not that easy. The upload form wasn’t restricting us on anything, I even uploaded a few PHP files, but they were useless, since viewing them in the upload folder returns the text as raw instead of executing it. When viewing the source, after uploading, we could see some response messages to our upload actions, that linked to a file named view.php that we could use to view the uploaded files. Now, it was kind of obvious that there is something to be done with this script, because why would we need a helper for viewing our uploaded files since we can already access them directly using the **uploads/** dir?

This part was the most frustrating one, I’ve tried so many methods and nothing seemed to be working. At first, it looked like the script replaces some characters in our file (actually deletes them), so we couldn’t use a PHP file with the script (anyway, it seemed to do file_get_contents on it, not include, so it wouldn’t be useful). I have tried to upload files containing different payloads and try to view them using view.php but nothing important happened, none of them worked, since most of the characters were deleted. I even found one kind of bug which using the data:// wrapper (which also didn’t work locally so it seemed like a good idea), would decode some string of mine and output a few random unprintable characters, and I thought I could try and find some combinations of PHP opening tags using it, but realized that the replacing is probably done after the entire page is read / constructed, and I decided to look for another solution and come back to this if I don’t find something better. After trying a lot of PHP wrappers, it seemed that the **php://filter** using **string.rot13** is actually giving some output for loading view.php itself. Indeed, after applying rot13 again to the strings, we find the PHP source of view.php (with many of the characters deleted because of it, so it was kind of unreadable, but I had something!). **convert.base64-encode** from **php://filter** seemed not to work, but **string.rot13** did, so what if we chained them together? like **php://filter/convert.base64-encode/string.rot13** ? This actually DID the job, and we got the entire source after base64 decoding the string (after also applying rot13 on it) and we can understand the functionality behind it. (as seen in the screenshot on the right)

```
if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    $cont_eva = file_get_contents($_GET['p']);
    $page = $cont_eva;
    if(!preg_match('#string.rot13#i', $_GET['p'])){
        $page = str_replace('_',"",$page);
        if(preg_match('#' . implode('|',get_defined_functions()['internal']) . '#i',$page) != 0)
            die("Don't like some functions! something is fishy!");
        if(preg_match('#zlib|bzip|convert|consumed|phar|zip|ftp|http|input|output#i',$page) != 0)
            die("Don't like this file! something is fishy!");
    }
    $page = str_replace(array('(', ')', ';', '[', ']', '$', '_', '<', '>', '/', '=', ' '), "", $page);
    if (!ctype_alnum($page)){
        print $page;
        die("Don't like this file! something is fishy!");
    }
    include ($_GET['p']);
}else{
    print 'Too easy!';
}
```

So the only method allowed is $_GET and our request must (should) contain “**string.rot13**” in it. As I thought before, it only does file_get_contents on the parameter, BUT we can see in the source that it also does include the file, IF it is only alphanumeric after removing the characters “**();[]$_<>/=** “ from it. Well, since http is allowed, and if allow_url_include would’ve been activated (which, unfortunately isn’t), a race condition would’ve been a nice solution (maybe adding a delay after file_get_contents) where our remote file would change it’s contents from something alphanumeric at first, to a PHP script in the second request. But this isn’t the case, we actually need to upload a PHP script that has “**string.rot13**” in it’s name (so it won’t conflict), that will bypass these conditions. Pretty simple right? Except of the PHP tags, which actually count the most.. “**?**” isn’t deleted from the file, so we can’t use **<?php** or **<?** as the opening tags for PHP, then how are we supposed to make the script valid? Well, a less known opening tag exists, (which btw, will be deprecated with php7), and it’s existance is kind of unwanted by the PHP community, even the official docs only list it in the changelog now: **<script language=”php”>php_code();</script>**. This seems like our way to go, but there is only one problem, the quotes aren’t deleted as well. However, since PHP parses literals as constants, and if they are undefined, they will be parsed as strings, I tried without quotes: **<script language=php>eval($_GET[csc]);</script>** and it worked! We can now use system() and cat the flag from **/flag_is_here**.

Flag: **CSC2016{073b0507fa5dfd45d9f9b2e4162f3d759e1245eab0cf251c4de050a10145274e}**