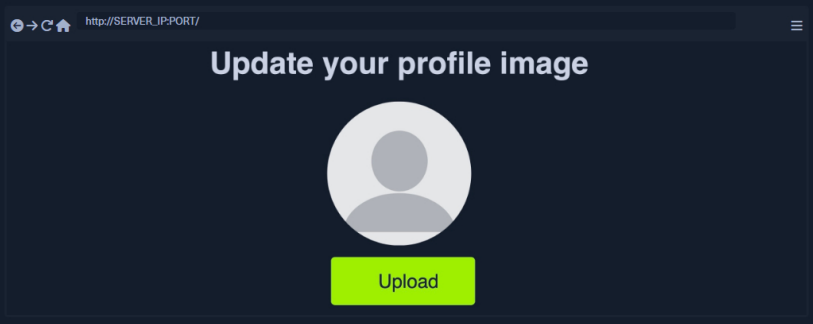
Client-Side Validation

Many web applications only rely on front-end JavaScript code to validate the selected file format before it is uploaded and would not upload it if the file is not in the required format (e.g., not an image).

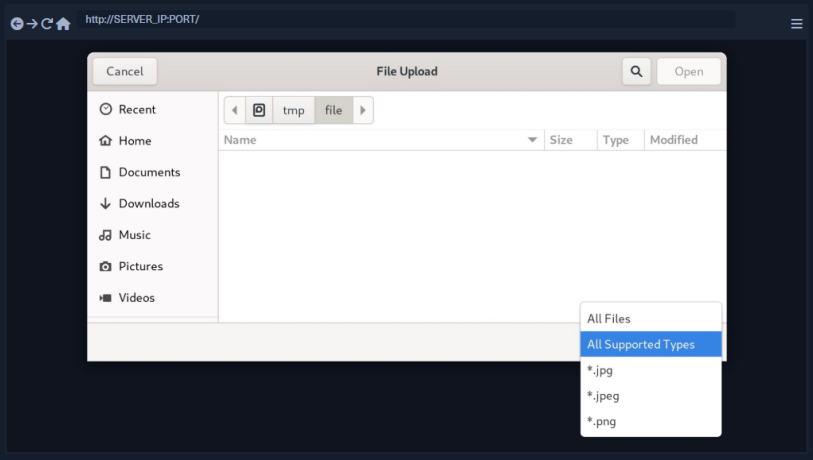
However, as the file format validation is happening on the client-side, we can easily bypass it by directly interacting with the server, skipping the front-end validations altogether. We may also modify the front-end code through our browser's dev tools to disable any validation in place.

Client-Side Validation

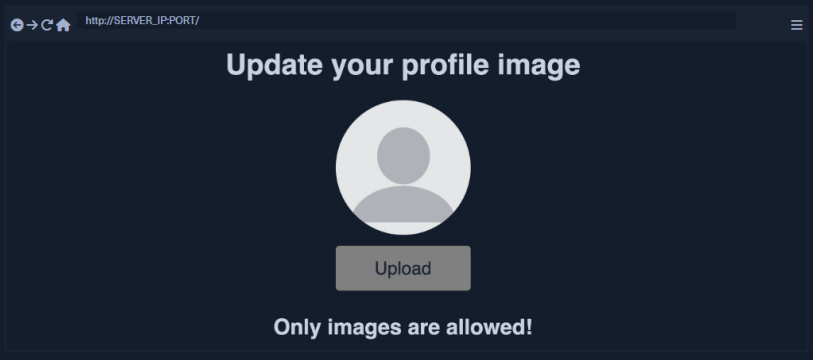
The exercise at the end of this section shows a basic `Profile Image` functionality, frequently seen in web applications that utilize user profile features, like social media web applications:



However, this time, when we get the file selection dialog, we cannot see our `PHP` scripts (or it may be greyed out), as the dialog appears to be limited to image formats only:



We may still select the `All Files` option to select our `PHP` script anyway, but when we do so, we get an error message saying (`Only images are allowed!`), and the `Upload` button gets disabled:



This indicates some form of file type validation, so we cannot just upload a web shell through the upload form as we did in the previous section. Luckily, all validation appears to be happening on the front-end, as the page never refreshes or sends any HTTP requests after selecting our file. So, we should be able to have complete control over these client-side validations.

Any code that runs on the client-side is under our control. While the web server is responsible for sending the front-end code, the rendering and execution of the front-end code happen within our browser. If the web application does not apply any of these validations on the back-end, we should be able to upload any file type.

As mentioned earlier, to bypass these protections, we can either `modify the upload request to the back-end server`, or we can `manipulate the front-end code to disable these type validations`

Cheat Sheet
Go to Questions

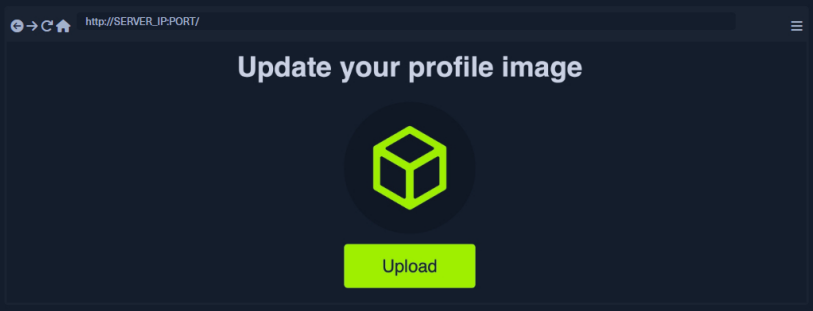
Table of Contents
Intro to File Upload Attacks
Basic Exploitation
Absent Validation
Upload Exploitation
Bypassing Filters
Client-Side Validation
Blacklist Filters
Whitelist Filters
Type Filters
Other Upload Attacks
Limited File Uploads
Other Upload Attacks
Prevention
Preventing File Upload Vulnerabilities
Skills Assessment
Skills Assessment - File Upload Attacks

My Workstation
OFFLINE
Start Instance
∞ / 1 spawns left

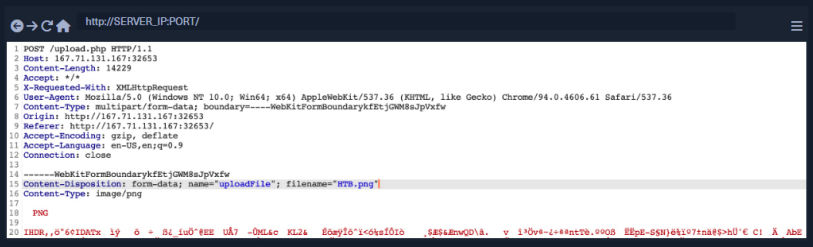
Manipulate the front-end code to disable these type validations.

Back-end Request Modification

Let's start by examining a normal request through `Burp`. When we select an image, we see that it gets reflected as our profile image, and when we click on `Upload`, our profile image gets updated and persists through refreshes. This indicates that our image was uploaded to the server, which is now displaying it back to us:



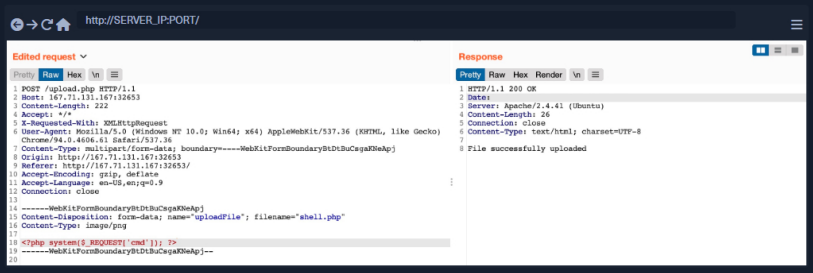
If we capture the upload request with `Burp`, we see the following request being sent by the web application:



The web application appears to be sending a standard HTTP upload request to `/upload.php`. This way, we can now modify this request to meet our needs without having the front-end type validation restrictions. If the back-end server does not validate the uploaded file type, then we should theoretically be able to send any file type/content, and it would be uploaded to the server.

The two important parts in the request are `filename="HTB.png"` and the file content at the end of the request. If we modify the `filename` to `shell.php` and modify the content to the web shell we used in the previous section; we would be uploading a `PHP` web shell instead of an image.

So, let's capture another image upload request, and then modify it accordingly:



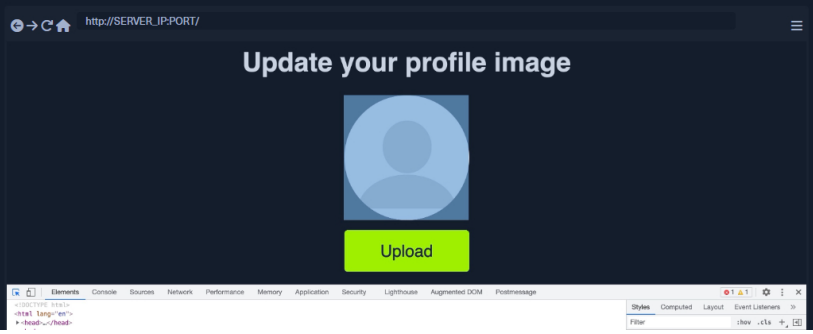
> **Note:** We may also modify the `Content-Type` of the uploaded file, though this should not play an important role at this stage, so we'll keep it unmodified.

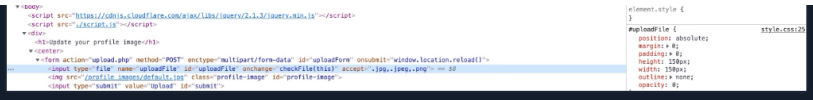
As we can see, our upload request went through, and we got `File successfully uploaded` in the response. So, we may now visit our uploaded file and interact with it and gain remote code execution.

Disabling Front-end Validation

Another method to bypass client-side validations is through manipulating the front-end code. As these functions are being completely processed within our web browser, we have complete control over them. So, we can modify these scripts or disable them entirely. Then, we may use the upload functionality to upload any file type without needing to utilize `Burp` to capture and modify our requests.

To start, we can click [`CTRL+SHIFT+C`] to toggle the browser's `Page Inspector`, and then click on the profile image, which is where we trigger the file selector for the upload form:





This will highlight the following HTML file input on line `18`:

Code: `html`

```
<input type="file" name="uploadFile" id="uploadFile" onchange="checkFile(this)" accept=".jpg,.jpeg,.png">
```

Here, we see that the file input specifies (`.jpg,.jpeg,.png`) as the allowed file types within the file selection dialog. However, we can easily modify this and select `All Files` as we did before, so it is unnecessary to change this part of the page.

The more interesting part is `onchange="checkFile(this)"`, which appears to run a JavaScript code whenever we select a file, which appears to be doing the file type validation. To get the details of this function, we can go to the browser's `Console` by clicking [`CTRL+SHIFT+K`], and then we can type the function's name (`checkFile`) to get its details:

Code: `javascript`

```
function checkFile(File) {
...SNIP...
    if (extension !== 'jpg' && extension !== 'jpeg' && extension !== 'png') {
        $('#error_message').text("Only images are allowed!");
        File.form.reset();
        $("#submit").attr("disabled", true);
    ...SNIP...
    }
}
```

The key thing we take from this function is where it checks whether the file extension is an image, and if it is not, it prints the error message we saw earlier (`Only images are allowed!`) and disables the `Upload` button. We can add `PHP` as one of the allowed extensions or modify the function to remove the extension check.

Luckily, we do not need to get into writing and modifying JavaScript code. We can remove this function from the HTML code since its primary use appears to be file type validation, and removing it should not break anything.

To do so, we can go back to our inspector, click on the profile image again, double-click on the function name (`checkFile`) on line `18`, and delete it:

Tip: You may also do the same to remove `accept=".jpg,.jpeg,.png"`, which should make selecting the `PHP` shell easier in the file selection dialog, though this is not mandatory, as mentioned earlier.

With the `checkFile` function removed from the file input, we should be able to select our `PHP` web shell through the file selection dialog and upload it normally with no validations, similar to what we did in the previous section.

Note: The modification we made to the source code is temporary and will not persist through page refreshes, as we are only changing it on the client-side. However, our only need is to bypass the client-side validation, so it should be enough for this purpose.

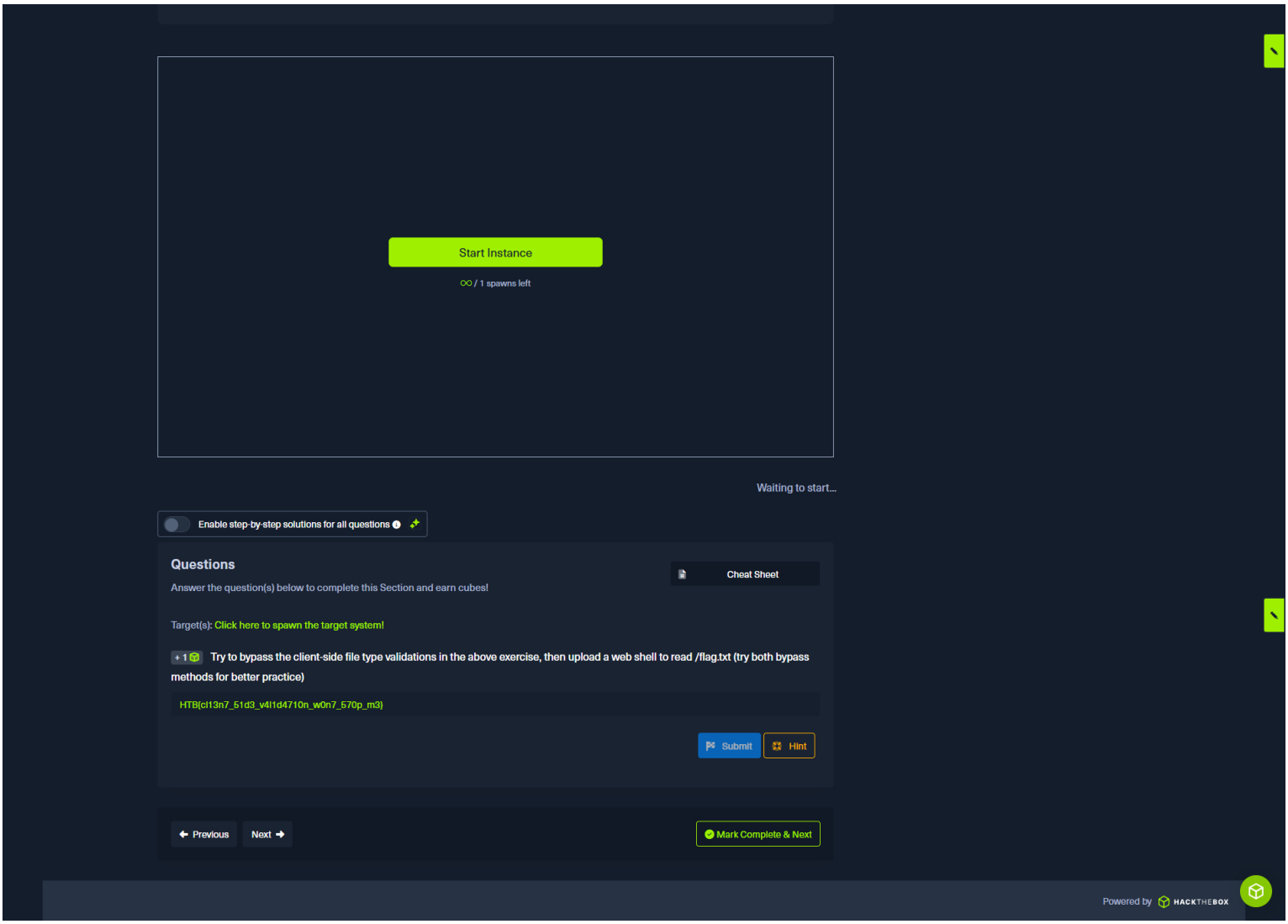
Once we upload our web shell using either of the above methods and then refresh the page, we can use the `Page Inspector` once more with [`CTRL+SHIFT+C`], click on the profile image, and we should see the URL of our uploaded web shell:

Code: `html`

```
<img src="/profile_images/shell.php" class="profile-image" id="profile-image">
```

If we can click on the above link, we will get to our uploaded web shell, which we can interact with to execute commands on the back-end server:

Note: The steps shown apply to Firefox, as other browsers may have slightly different methods for applying local changes to the source, like the use of `overrides` in Chrome.


Start Instance
∞ / 1 spawns left
Waiting to start...
Enable step-by-step solutions for all questions
Questions
Cheat Sheet
Answer the question(s) below to complete this Section and earn cubes!
Target(s): Click here to spawn the target system!
+ 1 Try to bypass the client-side file type validations in the above exercise, then upload a web shell to read /flag.txt (try both bypass methods for better practice)
HTB{cl13n7_51d3_v4l1d4710n_w0n7_570p_m3}
Submit
Hint
Previous
Next
Mark Complete & Next
Powered by HACKTHEBOX