

Blacklist Filters

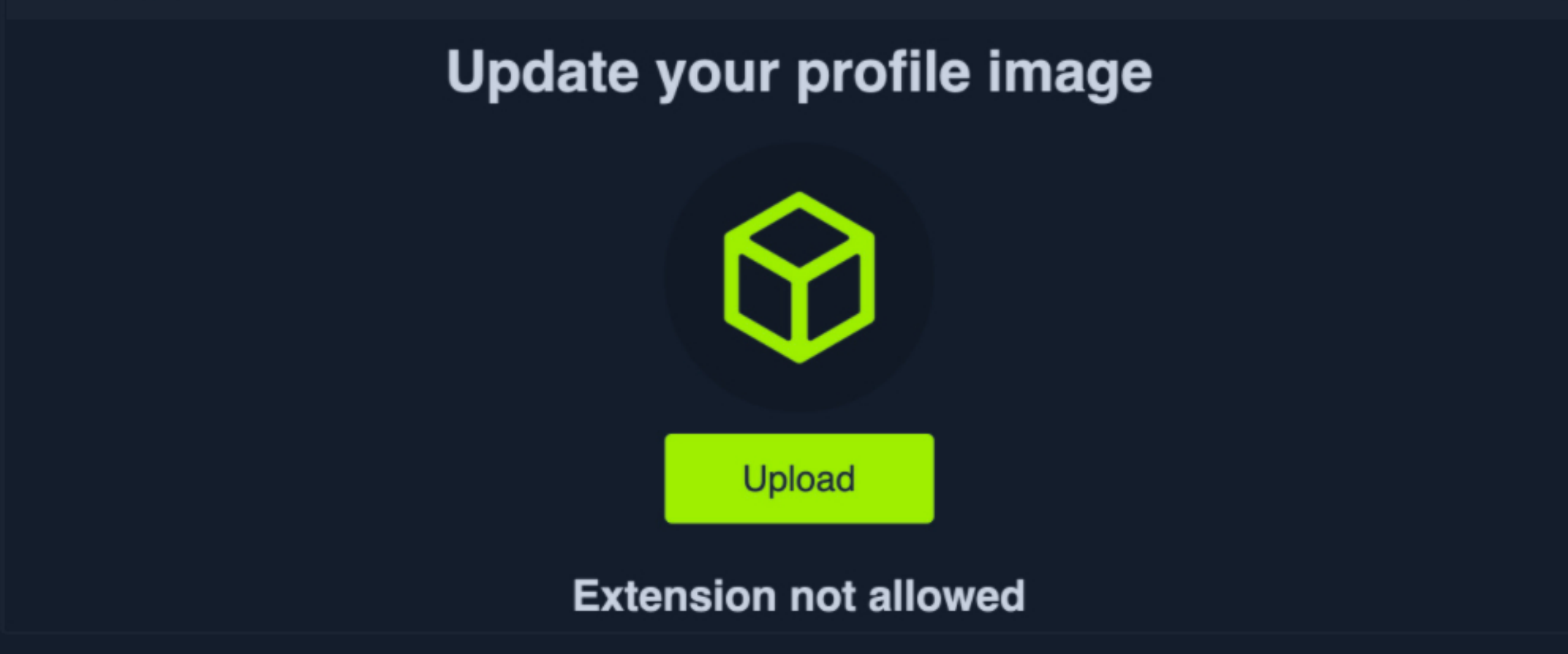
In the previous section, we saw an example of a web application that only applied type validation controls on the front-end (i.e., client-side), which made it trivial to bypass these controls. This is why it is always recommended to implement all security-related controls on the back-end server, where attackers cannot directly manipulate it.

Still, if the type validation controls on the back-end server were not securely coded, an attacker can utilize multiple techniques to bypass them and reach PHP file uploads.

The exercise we find in this section is similar to the one we saw in the previous section, but it has a blacklist of disallowed extensions to prevent uploading web scripts. We will see why using a blacklist of common extensions may not be enough to prevent arbitrary file uploads and discuss several methods to bypass it.

Blacklisting Extensions

Let's start by trying one of the client-side bypasses we learned in the previous section to upload a PHP script to the back-end server. We'll intercept an image upload request with Burp, replace the file content and filename with our PHP script's, and forward the request:



As we can see, our attack did not succeed this time, as we got **Extension not allowed**. This indicates that the web application may have some form of file type validation on the back-end, in addition to the front-end validations.

There are generally two common forms of validating a file extension on the back-end:

- Testing against a **blacklist** of types
- Testing against a **whitelist** of types

Furthermore, the validation may also check the **file type** or the **file content** for type matching. The weakest form of validation amongst these is **testing the file extension against a blacklist of extension** to determine whether the upload request should be blocked. For example, the following piece of code checks if the uploaded file extension is **PHP** and drops the request if it is:

```
Code: php

$fileName = basename($_FILES["uploadFile"]["name"]);
$extension = pathinfo($fileName, PATHINFO_EXTENSION);
$blacklist = array('php', 'php7', 'phps');

if (in_array($extension, $blacklist)) {
    echo "File type not allowed";
    die();
}
```

The code is taking the file extension (**\$extension**) from the uploaded file name (**\$fileName**) and then comparing it against a list of blacklisted extensions (**\$blacklist**). However, this validation method has a major flaw. **It is not comprehensive**, as many other extensions are not included in this list, which may still be used to execute PHP code on the back-end server if uploaded.

Tip: The comparison above is also case-sensitive, and is only considering lowercase extensions. In Windows Servers, file names are case insensitive, so we may try uploading a **php** with a mixed-case (e.g. **PHP**), which may bypass the blacklist as well, and should still execute as a PHP script.

So, let's try to exploit this weakness to bypass the blacklist and upload a PHP file.

Fuzzing Extensions

As the web application seems to be testing the file extension, our first step is to fuzz the upload functionality with a list of potential extensions and see which of them return the previous error message. Any upload requests that do not return an error message, return a different message, or succeed in uploading the file, may indicate an allowed file extension.

There are many lists of extensions we can utilize in our fuzzing scan. **PayloadsAllTheThings** provides lists of extensions for **PHP** and **.NET** web applications. We may also use **SecLists** list of common **Web Extensions**.

We may use any of the above lists for our fuzzing scan. As we are testing a PHP application, we will download and use the above **PHP** list. Then, from **Burp History**, we can locate our last request to **/upload.php**, right-click on it, and select **Send to Intruder**. From the **Positions** tab, we can **Clear** any automatically set positions, and then select the **.php** extension in **filenames="HTB.php"** and click the **Add** button to add it as a fuzzing position:

