

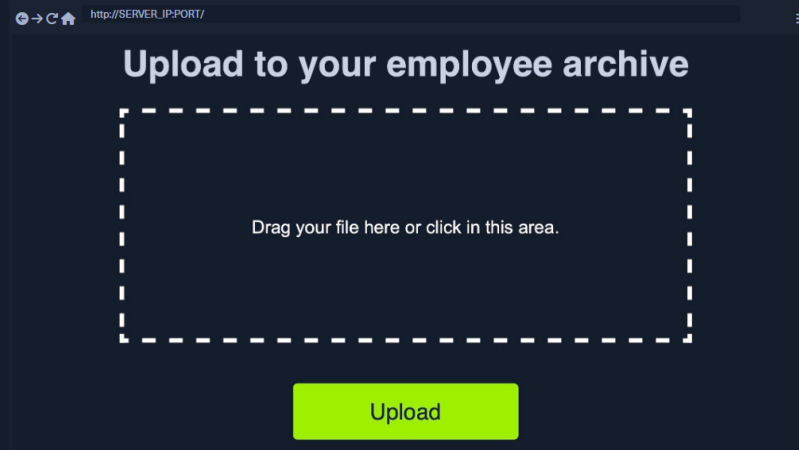
Absent Validation

The most basic type of file upload vulnerability occurs when the web application **does not have any form of validation filters** on the uploaded files, allowing the upload of any file type by default.

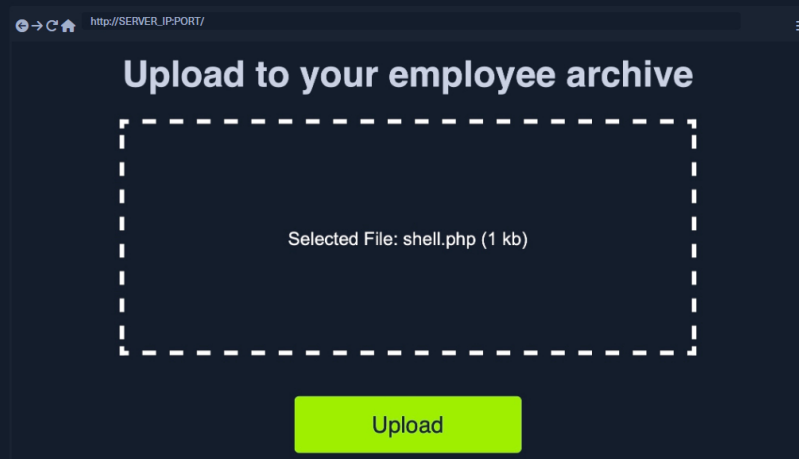
With these types of vulnerable web apps, we may directly upload our web shell or reverse shell script to the web application, and then by just visiting the uploaded script, we can interact with our web shell or send the reverse shell.

Arbitrary File Upload

Let's start the exercise at the end of this section, and we will see an **Employee File Manager** web application, which allows us to upload personal files to the web application:



The web application does not mention anything about what file types are allowed, and we can drag and drop any file we want, and its name will appear on the upload form, including **.php** files:



Furthermore, if we click on the form to select a file, the file selector dialog does not specify any file type, as it says **All Files** for the file type, which may also suggest that no type of restrictions or limitations are specified for the web application:

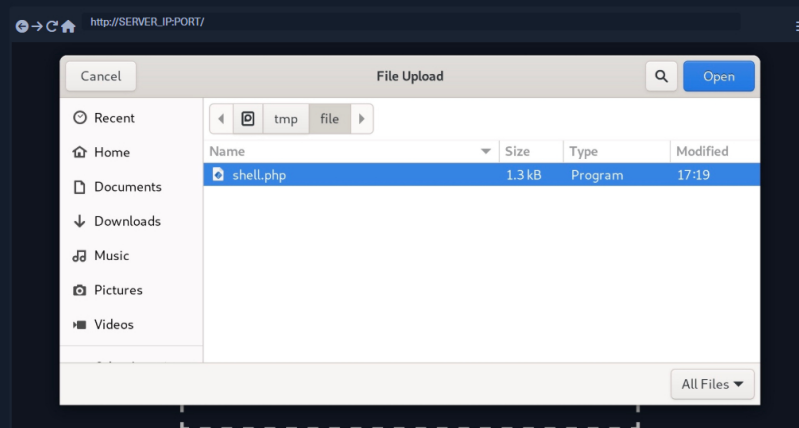
[📄 Cheat Sheet](#)[? Go to Questions](#)

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

Upload

All of this tells us that the program appears to have no file type restrictions on the front-end, and if no restrictions were specified on the back-end, we might be able to upload arbitrary file types to the back-end server to gain complete control over it.

Identifying Web Framework

We need to upload a malicious script to test whether we can upload any file type to the back-end server and test whether we can use this to exploit the back-end server. Many kinds of scripts can help us exploit web applications through arbitrary file upload, most commonly a **Web Shell** script and a **Reverse Shell** script.

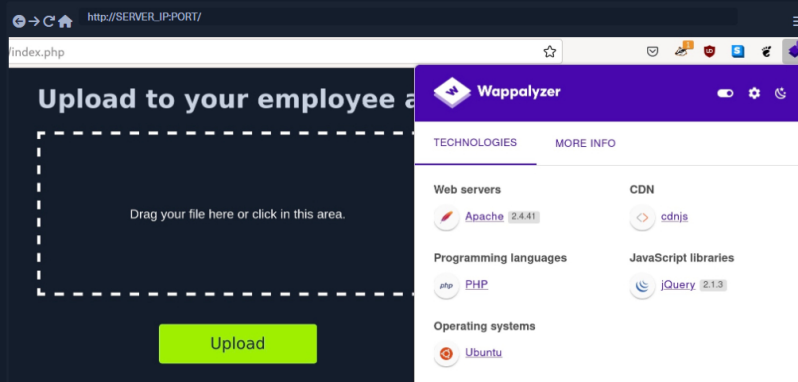
A Web Shell provides us with an easy method to interact with the back-end server by accepting shell commands and printing their output back to us within the web browser. A web shell has to be written in the same programming language that runs the web server, as it runs platform-specific functions and commands to execute system commands on the back-end server, making web shells non-cross-platform scripts. So, the first step would be to identify what language runs the web application.

This is usually relatively simple, as we can often see the web page extension in the URLs, which may reveal the programming language that runs the web application. However, in certain web frameworks and web languages, **Web Routes** are used to map URLs to web pages, in which case the web page extension may not be shown. Furthermore, file upload exploitation would also be different, as our uploaded files may not be directly routable or accessible.

One easy method to determine what language runs the web application is to visit the `/index.ext` page, where we would swap out `ext` with various common web extensions, like `php`, `asp`, `aspx`, among others, to see whether any of them exist.

For example, when we visit our exercise below, we see its URL as `http://SERVER_IP:PORT/`, as the `index` page is usually hidden by default. But, if we try visiting `http://SERVER_IP:PORT/index.php`, we would get the same page, which means that this is indeed a **PHP** web application. We do not need to do this manually, of course, as we can use a tool like Burp Intruder for fuzzing the file extension using a **Web Extensions** wordlist, as we will see in upcoming sections. This method may not always be accurate, though, as the web application may not utilize index pages or may utilize more than one web extension.

Several other techniques may help identify the technologies running the web application, like using the **Wappalyzer** extension, which is available for all major browsers. Once added to our browser, we can click its icon to view all technologies running the web application:



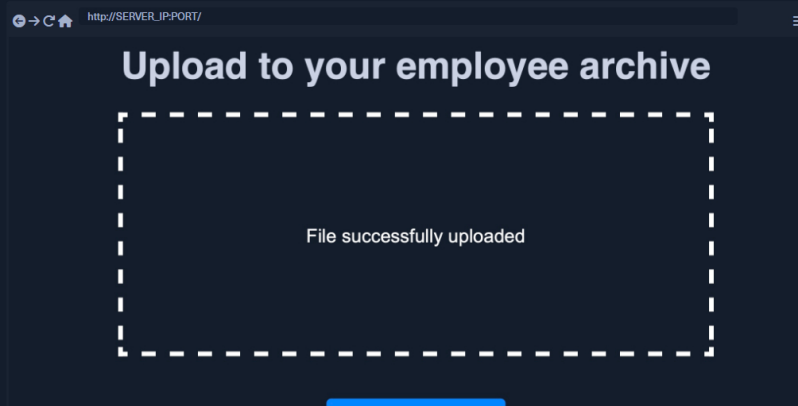
As we can see, not only did the extension tell us that the web application runs on **PHP**, but it also identified the type and version of the web server, the back-end operating system, and other technologies in use. These extensions are essential in a web penetration tester's arsenal, though it is always better to know alternative manual methods to identify the web framework, like the earlier method we discussed.

We may also run web scanners to identify the web framework, like Burp/ZAP scanners or other Web Vulnerability Assessment tools. In the end, once we identify the language running the web application, we may upload a malicious script written in the same language to exploit the web application and gain remote control over the back-end server.

Vulnerability Identification

Now that we have identified the web framework running the web application and its programming language, we can test whether we can upload a file with the same extension. As an initial test to identify whether we can upload arbitrary **PHP** files, let's create a basic **Hello World** script to test whether we can execute **PHP** code with our uploaded file.

To do so, we will write `<?php echo "Hello HTB";?>` to `test.php`, and try uploading it to the web application:




Download File

The file appears to have successfully been uploaded, as we get a message saying `File successfully uploaded`, which means that the web application has no file validation whatsoever on the back-end. Now, we can click the `Download` button, and the web application will take us to our uploaded file:




As we can see, the page prints our `Hello HTB` message, which means that the `echo` function was executed to print our string, and we successfully executed `PHP` code on the back-end server. If the page could not run `PHP` code, we would see our source code printed on the page.

In the next section, we will see how to exploit this vulnerability to execute code on the back-end server and take control over it.

**Connect to Pwnbox**
Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

UK 1.38PM

 Terminate Pwnbox to switch location

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 upload a PHP script that executes the `(hostname)` command on the back-end server, and submit the first word of it as the answer.

fileuploadsabseentification

Submit Hint

Previous Next Mark Complete & Next

