What are file upload vulnerabilities?

File upload vulnerabilities arise when a server allows users to upload files without validating their names, size, types, content etc. In this article, we will learn common attack vectors that can be used to exploit improper file upload functionality and bypass common defense mechanisms.

How file upload requests work

Before digging deep into the exploitation part of the file upload functionality, let's take a deeper look at file upload requests. There are 5 parts to a file upload request, as seen below:

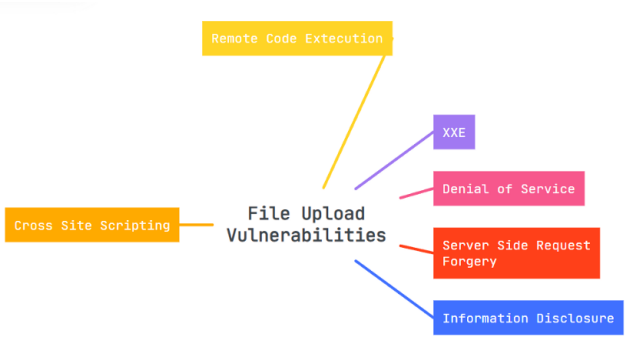
1. Filename: This is the filename of the uploaded file.
2. Content-Type: Content-Type header is used to indicate the content type of the uploaded file. For instance: For a png file it is "image/png", for a text file it is "text/plain".
3. Magic Number: Magic number is a series of hexadecimal values present at the beginning of the file that determines the type of content of the file.
4. Content of the file: This is the main content of the file.

Now that we understand the basics of file upload requests and their components, let's explore some techniques used to exploit file uploads.

Why are file upload vulnerabilities a problem?

File upload vulnerabilities tend to be labelled high-severity. Here are some of the risks that come with improper implementation of a file upload functionality:

- Server Side Attacks: File upload vulnerabilities can be compromised by uploading a malicious web-shell which allows an attacker to run arbitrary commands, browse local files, etc.
- Client Side Attacks: File upload vulnerabilities also makes applications vulnerable to cross site scripting attack or cross site content hijacking.
- DoS Attacks: Improper implementation of file upload functionality also leads to Denial of Service attacks.
- File upload pages sometimes disclose internal sensitive information such as server internal paths in error messages.

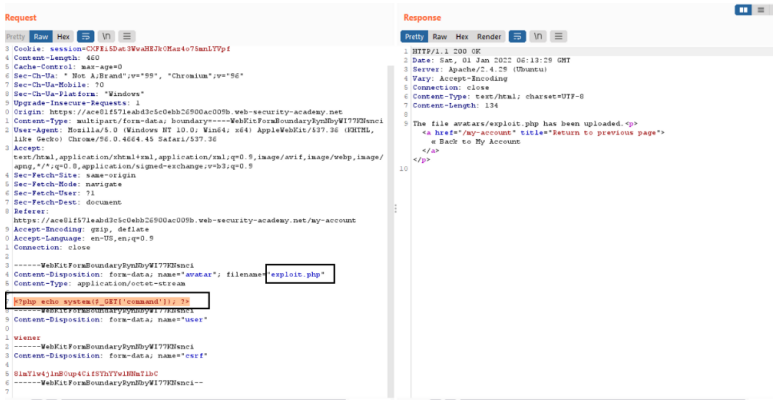


File upload vulnerabilities are, in a sense, a 'gateway vulnerability' to many other security flaws that could seriously compromise your application. Now we'll look at some specific techniques attackers use to exploit this vulnerability.

Exploit #1: Through file contents

Remote Code Execution (Web Shell Upload)

Let's start with a basic web shell upload that allows you to run arbitrary commands on the vulnerable server. In the example below, that application has no security on the vulnerable server, which allows the attacker to upload a malicious php file with the following payload:<?php echo system($_GET['command']); ?>



Now they can run arbitrary commands on the web server by using the URL, i.e. https://endpointofuploadedfile?command={command}. In most cases, the application doesn't allow users to upload malicious files to the application. But these security

measures can be bypassed using the techniques you see below.

Bypassing blacklisting protection

The main problem with blacklisting is that you can't really blacklist every possible vector.

Case 1: By changing the blocked extension's name to uppercase

Let's start with the basic bypass. Suppose a developer blocks php, html, exe and other extensions in a smaller case. In those scenarios, we can change the case of the extensions to bypass the file upload restrictions. For example, a php file can be uploaded by changing the extensions to PHP.

Case 2: Bypassing using alternative extensions that might not be blocked

For instance, a developer might block php, html, exe extensions. Attackers can still use alternative extensions like php1, php2, php3, php4, php5, php6, phtml, etc. to bypass the restrictions and upload malicious files.

The equivalent for Windows IIS servers is if a developer blocked asp extension. Attackers can still bypass this protection using extensions like "asa" & "cer". IIS 7.5 and under have both *.asp and *.cer mapped to asp.dll, thus executing ASP code.

Case 3: Upload file using File Traversal sequences

Sometimes servers restrict executing scripts in the user controlled directories as they are explicitly configured to do that. If the server doesn't match the expected content-type they might return some kind of error. These restrictions depend on directories to directories basis.

For instance: file.jpg gets uploaded in the /users/images directory. It might happen that even if the attacker is able to upload a malicious script to this directory, the server doesn't execute the script. We can bypass this using file traversal sequence "../exploit.php" (if the server is not validating the filename).

Another instance for exploiting using File traversal sequences is to upload the sequences with the name of local file of the server (uploading the file with a name like "../../logo.png" to change the logo of the application)

Case 4: Bypassing the content-type check

Developers might check the type of the file using the content-type header. For instance, when you try to upload a php file, the content-type will be "application/x-httpd-php". The value of content-type can still be altered by using any proxy tool.

For bypassing this protection, intercept the request and change the value of the content-type to the acceptable ones(image/png, image/jpeg).

Case 5: Bypassing the file type check

The web application might check the signature of the file in order to verify the type of file. In those cases, we can bypass this check either by adding a magic number of file (to the start of the payload) or by adding our code in a comment in the file using exiftool.

Bypassing exploit dictionaries

Bypassing whitelist protection

Whitelist protection is when developers allow the user to upload only certain types of files. This seems to be easier to implement than blacklisting, but there still are ways to bypass whitelisting protection.

Case 1: Bypassing the file extensions check

The application only validates that the uploaded file contains valid extensions, but not validating if that the file ends with that valid extension or not. For instance, we can bypass this by using filename "exploit.jpg.php"

Case 2: Null Byte Injection

If the application has validations written in high-level languages like PHP or JavaScript, but the server processes the file in languages like C or C++ then this can create confusion in how a file is treated. Protection can be bypassed using the null byte(exploit.jpg%00.php or exploit.jpg\00.php).

Exploit #2: Through SVG files

Some applications allow users to upload SVG files which are later processed on the server side. Because SVG format uses XML, an attacker can create a malicious file to exploit vulnerabilities like SSRF and XXE.

Case 1: SSRF

For this exploit, an attacker needs to create an SVG file with the below content and change the controlled server URL to its own server:

<svg xmlns:svg=" http://www.w3.org/2000/svg " xmlns=" http://www.w3.org/2000/svg " xmlns:xlink=" http://www.w3.org/1999/xlink " width="200" height="200">

<image height="30" width="30"

xlink:href="https://controlledserver.com/pic.svg" />

</svg>

If they upload this file to the application, there will be a callback if the application is vulnerable.

Case 2: XXE

Here, attackers create an SVG file with the content shown below, and if the server is vulnerable the content of the local file is visible either in the response or in the image itself.

<?xml version="1.0" standalone="yes"?><!DOCTYPE test [<!ENTITY xxe SYSTEM "file:///etc/hostname" >]><svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"><text font-size="16" x="0" y="16">&xxe;</text></svg>

Case 3: Cross Site Scripting

In this case, attackers need to create an SVG file with the below content. If the server is vulnerable, they will see a pop-up showing that it is.

```
<svg xmlns=" http://wwww3.org/2000/svg  ">
<script>alert('XSS');</script>
<rect x="0" height="100" width="100" style="fill: #cccccc"/>
<line x1="20" y1="80" x2="80" y2="80" style="stroke: #ff0000; stroke-width: 5;"/>
</svg>
```

Exploit #3: Through filename

Case 1: DoS (Denial of Service)

When there is no text limitation for the name of the uploaded file, it's possible for an attacker to create a file with a long, heavy name, causing a denial of service attack on the application.

For reference, you can view this report.

Case 2: XSS (Cross Site Scripting)

The filename sometimes itself is reflected in the page, so simply by altering the filename can trigger cross site scripting on the vulnerable server.

For this exploit, simply rename the file as "/><svg onload = alert(document.cookie)> and upload the file on the server.

Exploit #4: Through size of file

Case 1: Denial of Service

Web applications sometimes do not validate the file size of the uploaded files. In this case the web application might be vulnerable to Denial of Service Attack which can be exploited by uploading many large files which will exhaust the server hosting space. This vulnerability is also referred to as pixel flood attack.

For reference, you can look into this report.

How to prevent and mitigate file upload vulnerabilities

- Your application should always check the content of the uploaded file. If it detects anything malicious, the file must be discarded.
- Maintain a whitelist of allowed extensions. Make sure that a file does not contain more than one extension.
- Make sure that the filename must not contain any special characters like ";", ":", ">", "<", "/", "\", "", "*", "%" etc.
- Limit the size of the file name.
- Limit the size (minimum & maximum) of file upload to prevent DOS attacks.
- Make sure to disable execute permission on the directories where all the uploaded files are stored.
- Ensure the uploaded files do not replace the local files of the server.