# Other Upload Attacks

In addition to arbitrary file uploads and limited file upload attacks, there are a few other techniques and attacks worth mentioning, as they may become handy in some web penetration tests or bug bounty tests. Let's discuss some of these techniques and when we may use them.

## Injections in File Name

A common file upload attack uses a malicious string for the uploaded file name, which may get executed or processed if the uploaded file name is displayed (i.e., reflected) on the page. We can try injecting a command in the file name, and if the web application uses the file name within an OS command, it may lead to a command injection attack.

For example, if we name a file `file$(whoami).jpg` or `file`whoami`.jpg` or `file.jpg||whoami`, and then the web application attempts to move the uploaded file with an OS command (e.g. `mv file /tmp`), then our file name would inject the `whoami` command, which would get executed, leading to remote code execution. You may refer to the Command Injections module for more information.

Similarly, we may use an XSS payload in the file name (e.g. `<script>alert(window.origin);</script>`), which would get executed on the target's machine if the file name is displayed to them. We may also inject an SQL query in the file name (e.g. `file';select+sleep(5);--.jpg`), which may lead to an SQL injection if the file name is insecurely used in an SQL query.

## Upload Directory Disclosure

In some file upload forms, like a feedback form or a submission form, we may not have access to the link of our uploaded file and may not know the uploads directory. In such cases, we may utilize fuzzing to look for the uploads directory or even use other vulnerabilities (e.g., LFI/XXE) to find where the uploaded files are by reading the web applications source code, as we saw in the previous section. Furthermore, the Web Attacks/IDOR module discusses various methods of finding where files may be stored and identifying the file naming scheme.

Another method we can use to disclose the uploads directory is through forcing error messages, as they often reveal helpful information for further exploitation. One attack we can use to cause such errors is uploading a file with a name that already exists or sending two identical requests simultaneously. This may lead the web server to show an error that it could not write the file, which may disclose the uploads directory. We may also try uploading a file with an overly long name (e.g., 5,000 characters). If the web application does not handle this correctly, it may also error out and disclose the upload directory.

Similarly, we may try various other techniques to cause the server to error out and disclose the uploads directory, along with additional helpful information.

## Windows-specific Attacks

We can also use a few `Windows-Specific` techniques in some of the attacks we discussed in the previous sections.

One such attack is using reserved characters, such as (`|`, `<`, `>`, `*`, or `?`), which are usually reserved for special uses like wildcards. If the web application does not properly sanitize these names or wrap them within quotes, they may refer to another file (which may not exist) and cause an error that discloses the upload directory. Similarly, we may use Windows reserved names for the uploaded file name, like (`CON`, `COM1`, `LPT1`, or `NUL`), which may also cause an error as the web application will not be allowed to write a file with this name.

Finally, we may utilize the Windows 8.3 Filename Convention to overwrite existing files or refer to files that do not exist. Older versions of Windows were limited to a short length for file names, so they used a Tilde character (`~`) to complete the file name, which we can use to our advantage.

For example, to refer to a file called (`hackthebox.txt`) we can use (`HAC~1.TXT`) or (`HAC~2.TXT`), where the digit represents the order of the matching files that start with (`HAC`). As Windows still supports this convention, we can write a file called (e.g. `WEB~1.CON`) to overwrite the `web.conf` file. Similarly, we may write a file that replaces sensitive system files. This attack can lead to several outcomes, like causing information disclosure through errors, causing a DoS on the back-end server, or even accessing private files.

## Advanced File Upload Attacks

In addition to all of the attacks we have discussed in this module, there are more advanced attacks that can be used with file upload functionalities. Any automatic processing that occurs to an uploaded file, like encoding a video, compressing a file, or renaming a file, may be exploited if not securely coded.

Some commonly used libraries may have public exploits for such vulnerabilities, like the AVI upload vulnerability leading to XXE in `ffmpeg`. However, when dealing with custom code and custom libraries, detecting such vulnerabilities requires more advanced knowledge and techniques, which may lead to discovering an advanced file upload vulnerability in some web applications.

There are many other advanced file upload vulnerabilities that we did not discuss in this module. Try to read some bug bounty reports to explore more advanced file upload vulnerabilities.

Previous Next

Mark Complete & Next

Cheat Sheet

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