Intro to File Upload Attacks

Uploading user files has become a key feature for most modern web applications to allow the extensibility of web applications with user information. A social media website allows the upload of user profile images and other social media, while a corporate website may allow users to upload PDFs and other documents for corporate use.

However, as web application developers enable this feature, they also take the risk of allowing end-users to store their potentially malicious data on the web application's back-end server. If the user input and uploaded files are not correctly filtered and validated, attackers may be able to exploit the file upload feature to perform malicious activities, like executing arbitrary commands on the back-end server to take control over it.

File upload vulnerabilities are amongst the most common vulnerabilities found in web and mobile applications, as we can see in the latest CVE Reports. We will also notice that most of these vulnerabilities are scored as `High` or `Critical` vulnerabilities, showing the level of risk caused by insecure file upload.

Types of File Upload Attacks

The most common reason behind file upload vulnerabilities is weak file validation and verification, which may not be well secured to prevent unwanted file types or could be missing altogether. The worst possible kind of file upload vulnerability is an `unauthenticated arbitrary file upload` vulnerability. With this type of vulnerability, a web application allows any unauthenticated user to upload any file type, making it one step away from allowing any user to execute code on the back-end server.

Many web developers employ various types of tests to validate the extension or content of the uploaded file. However, as we will see in this module, if these filters are not secure, we may be able to bypass them and still reach arbitrary file uploads to perform our attacks.

The most common and critical attack caused by arbitrary file uploads is `gaining remote command execution` over the back-end server by uploading a web shell or uploading a script that sends a reverse shell. A web shell, as we will discuss in the next section, allows us to execute any command we specify and can be turned into an interactive shell to enumerate the system easily and further exploit the network. It may also be possible to upload a script that sends a reverse shell to a listener on our machine and then interact with the remote server that way.

In some cases, we may not have arbitrary file uploads and may only be able to upload a specific file type. Even in these cases, there are various attacks we may be able to perform to exploit the file upload functionality if certain security protections were missing from the web application.

Examples of these attacks include:

- Introducing other vulnerabilities like `XSS` or `XXE`.
- Causing a `Denial of Service (DoS)` on the back-end server.
- Overwriting critical system files and configurations.
- And many others.

Finally, a file upload vulnerability is not only caused by writing insecure functions but is also often caused by the use of outdated libraries that may be vulnerable to these attacks. At the end of the module, we will go through various tips and practices to secure our web applications against the most common types of file upload attacks, in addition to further recommendations to prevent file upload vulnerabilities that we may miss.

Next ➜ ✓ Mark Complete & Next

Cheat Sheet

Table of Contents

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left