File Upload Cheat Sheet

Introduction

File upload is becoming a more and more essential part of any application, where the user is able to upload their photo, their CV, or a video showcasing a project they are working on. The application should be able to fend off bogus and malicious files in a way to keep the application and the users safe.

In short, the following principles should be followed to reach a secure file upload implementation:

- **List allowed extensions. Only allow safe and critical extensions for business functionality**
  - **Ensure that input validation is applied before validating the extensions.**
- **Validate the file type, don't trust the Content-Type header as it can be spoofed**
- **Change the filename to something generated by the application**
- **Set a filename length limit. Restrict the allowed characters if possible**
- **Set a file size limit**
- **Only allow authorized users to upload files**
- **Store the files on a different server. If that's not possible, store them outside of the webroot**
  - **In the case of public access to the files, use a handler that gets mapped to filenames inside the application (someid -> file.ext)**
- **Run the file through an antivirus or a sandbox if available to validate that it doesn't contain malicious data**
- **Ensure that any libraries used are securely configured and kept up to date**
- **Protect the file upload from CSRF attacks**

File Upload Threats

In order to assess and know exactly what controls to implement, knowing what you're facing is essential to protect your assets. The following sections will hopefully showcase the risks accompanying the file upload functionality.

Malicious Files

The attacker delivers a file for malicious intent, such as:

1. Exploit vulnerabilities in the file parser or processing module (*e.g.* ImageTrick Exploit, XXE)
2. Use the file for phishing (*e.g.* careers form)
3. Send ZIP bombs, XML bombs (otherwise known as billion laughs attack), or simply huge files in a way to fill the server storage which hinders and damages the server's availability
4. Overwrite an existing file on the system
5. Client-side active content (XSS, CSRF, etc.) that could endanger other users if the files are publicly retrievable.

Public File Retrieval

If the file uploaded is publicly retrievable, additional threats can be addressed:

1. Public disclosure of other files
2. Initiate a DoS attack by requesting lots of files. Requests are small, yet responses are much larger
3. File content that could be deemed as illegal, offensive, or dangerous (*e.g.* personal data, copyrighted data, etc.) which will make you a host for such malicious files.

File Upload Protection

There is no silver bullet in validating user content. Implementing a defense in depth approach is key to make the upload process harder and more locked down to the needs and requirements for the service. Implementing multiple techniques is key and recommended, as no one technique is enough to secure the service.

Extension Validation

Ensure that the validation occurs after decoding the file name, and that a proper filter is set in place in order to avoid certain known bypasses, such as the following:

- Double extensions, *e.g.* `.jpg.php`, where it circumvents easily the regex `\.jpg`
- Null bytes, *e.g.* `.php%00.jpg`, where `.jpg` gets truncated and `.php` becomes the new extension
- Generic bad regex that isn't properly tested and well reviewed. Refrain from building your own logic unless you have enough knowledge on this topic.

Refer to the Input Validation CS to properly parse and process the extension.

List Allowed Extensions

Ensure the usage of *business-critical* extensions only, without allowing any type of *non-required* extensions. For example if the system requires:

- image upload, allow one type that is agreed upon to fit the business requirement;
- cv upload, allow `docx` and `pdf` extensions.

Based on the needs of the application, ensure the **least harmful** and the **lowest risk** file types to be used.

Block Extensions

Identify potentially harmful file types and block extensions that you regard harmful to your service.

Please be aware that blocking specific extensions is a weak protection method on its own. The Unrestricted File Upload vulnerability article describes how attackers may attempt to bypass such a check.

Content-Type Validation

The Content-Type for uploaded files is provided by the user, and as such cannot be trusted, as it is trivial to spoof. Although it should not be relied upon for security, it provides a quick check to prevent users from unintentionally uploading files with the incorrect type.

Other than defining the extension of the uploaded file, its MIME-type can be checked for a quick protection against simple file upload attacks.

This can be done preferably in an allow list approach; otherwise, this can be done in a block list approach.

File Signature Validation

In conjunction with content-type validation, validating the file's signature can be checked and verified against the expected file that should be received.

> This should not be used on its own, as bypassing it is pretty common and easy.

Filename Sanitization

Filenames can endanger the system in multiple ways, either by using non acceptable characters, or by using special and restricted filenames. For Windows, refer to the following MSDN guide. For a wider overview on different filesystems and how they treat files, refer to Wikipedia's Filename page.

In order to avoid the above mentioned threat, creating a **random string** as a file-name, such as

generating a UUID/GUID, is essential. If the filename is required by the business needs, proper input validation should be done for client-side (*e.g.* active content that results in XSS and CSRF attacks) and back-end side (*e.g.* special files overwrite or creation) attack vectors. Filename length limits should be taken into consideration based on the system storing the files, as each system has its own filename length limit. If user filenames are required, consider implementing the following:

- Implement a maximum length
- Restrict characters to an allowed subset specifically, such as alphanumeric characters, hyphen, spaces, and periods
    - If this is not possible, block-list dangerous characters that could endanger the framework and system that is storing and using the files.

File Content Validation

As mentioned in the Public File Retrieval section, file content can contain malicious, inappropriate, or illegal data.

Based on the expected type, special file content validation can be applied:

- For **images**, applying image rewriting techniques destroys any kind of malicious content injected in an image; this could be done through randomization.
- For **Microsoft documents**, the usage of Apache POI helps validating the uploaded documents.
- **ZIP files** are not recommended since they can contain all types of files, and the attack vectors pertaining to them are numerous.

The File Upload service should allow users to report illegal content, and copyright owners to report abuse.

If there are enough resources, manual file review should be conducted in a sandboxed environment before releasing the files to the public.

Adding some automation to the review could be helpful, which is a harsh process and should be well studied before its usage. Some services (*e.g.* Virus Total) provide APIs to scan files against well known malicious file hashes. Some frameworks can check and validate the raw content type and validating it against predefined file types, such as in ASP.NET Drawing Library. Beware of data leakage threats and information gathering by public services.

File Storage Location

The location where the files should be stored must be chosen based on security and business requirements. The following points are set by security priority, and are inclusive:

1. Store the files on a **different host**, which allows for complete segregation of duties between the application serving the user, and the host handling file uploads and their storage.
2. Store the files **outside the webroot**, where only administrative access is allowed.
3. Store the files **inside the webroot**, and set them in write permissions only.
4. If read access is required, setting proper controls is a must (*e.g.* internal IP, authorized user, etc.)

Storing files in a studied manner in databases is one additional technique. This is sometimes used for automatic backup processes, non file-system attacks, and permissions issues. In return, this opens up the door to performance issues (in some cases), storage considerations for the database and its backups, and this opens up the door to SQLi attack. This is advised only when a DBA is on the team and that this process shows to be an improvement on storing them on the file-system.

> Some files are emailed or processed once they are uploaded, and are not stored on the server. It is essential to conduct the security measures discussed in this sheet before doing any actions on them.

User Permissions

Before any file upload service is accessed, proper validation should occur on two levels for the user uploading a file:

- Authentication level
  - The user should be a registered user, or an identifiable user, in order to set restrictions and limitations for their upload capabilities
- Authorization level
  - The user should have appropriate permissions to access or modify the files

Filesystem Permissions

> Set the files permissions on the principle of least privilege.

Files should be stored in a way that ensures:

- Allowed system users are the only ones capable of reading the files
- Required modes only are set for the file
  - If execution is required, scanning the file before running it is required as a security best practice, to ensure that no macros or hidden scripts are available.

Upload and Download Limits

The application should set proper size limits for the upload service in order to protect the file storage capacity. If the system is going to extract the files or process them, the file size limit should be considered after file decompression is conducted and by using secure methods to calculate zip files size. For more on this, see how to Safely extract files from ZipInputStream, Java's input stream to handle ZIP files.

The application should set proper request limits as well for the download service if available to protect the server from DoS attacks.

Java Code Snippets

Document Upload Protection repository written by Dominique for certain document types in Java.