# **File Upload Validator – How to avoid file upload attacks**

In order to avoid file-upload attacks, there are several rules the application should follow.

## check File Type

### **Block dangerous extensions**

Make a white-list of file extensions to be allowed for upload. Try to keep away from executable files and scripts in the white-list.

White list is better than black list because it’s almost impossible to create a black list that includes all possible extensions that an attacker can use.

The white list will allow the upload of supported file types only, and prevent attacks using script files.

### **Mime Type validation**

Don’t just upload the files based on their extensions. File extension can be changed very easily. Always verify the file headers before uploading the file.

Verify that the file type according to the headers is matching the white list you’ve created. You can also verify that the file type is matching the extension.

#### **Double extensions**

Double extension (e.g. filename.doc.txt) is a problem in case the developer is extracting the file extension by looking for the ‘.’ character in the filename, and extracting the string after the ‘.’ character.

The method used to bypass this approach is a bit more complicated, but still realistic. First, let’s have a look at how Apache handles files with multiple extensions. A quote from the Apache manual states:

“Files can have more than one extension, and the order of the extensions is normally irrelevant. For example, if the file welcome.html.fr maps onto content type text/html and language French then the file welcome.fr.html will map onto exactly the same information. If more than one extension is given which maps onto the same type of meta-information, then the one to the right will be used, except for languages and content encodings. For example, if .gif maps to the MIME-type image/gif and .html maps to the MIME-type text/html, then the file welcome.gif.html will be associated with the MIME-type text/html.”

Therefore a file named ‘filename.php.123’, will be interpreted as a PHP file and will be executed. This only works if the last extension (in our case .123), is not specified in the list of mime-types known to the web server. Web developers, usually are not aware of such ‘feature’ in Apache, which can be very dangerous for a number of reasons. Knowing this, an attacker can upload a file named shell.php.123 and bypass the file upload form protection. The script will compute the last extension (.123), and concludes that this extension is not in the list of dangerous extension. Having said that, it is impossible to predict all the possible random extensions a malicious user will use to be able to upload a file on your web server.

## **use Random filenames and folder** NAME

While allowing users to upload the files, we allow them to specify the name the files should be referred to. Application should validate these file names for any XSS attacks (and filenames should be escaped).

Do not allow user input to specify the destination directory or file name of uploaded documents. Good practice is to rename the document to some random value. This way, guessing the name of the uploaded file is more difficult for the attacker.

Another important security measure is to use system-generated file names instead of the names supplied by users when storing files on the file system. This will prevent local file inclusion attacks and also make any kind of file name manipulation by the user impossible.

## **keep Upload Directory Security**

The most important safeguard is to keep uploaded files where they cannot be directly accessed by the users via a direct URL. This can be done either by storing uploaded files outside of the web root or configuring the web server to deny access to the uploads directory.

Set proper folder permissions. Do not allow user to choose the upload folder. Avoid giving writable permissions to users.

## **scan file with Anti Virus**

Scan the uploaded files for any virus or malicious content.

## Limit File size – avoid Denial of service attack

Set minimum size and maximum size for file upload. This will prevent denial-of-service attack by uploading huge files (consume all available disk space).

Even better solution is to implement a limit on the size and number of files one user can upload in a given period (a day for example). In this way, the attacker won’t be able to upload a lot of small files instead of one big file.

# Sources

<http://blogs.securiteam.com/>

<http://www.acunetix.com/>

<http://www.scanit.be/>

<http://www.developershome.com>