# File Upload Validator – Documentation

FUV supplies an easy-to-use set of tools that help to reduce the threats that a file upload server is exposed to when receiving files from un-trusted users. The vulnerabilities document covers the possible threats and the defend methods.

FUV was planned to be used by many different applications, running on many servers. It’s a java package distributed as a JAR file. Each application can configure it for its own needs according to its specific threats and the integration with the existing code is relatively simple, using a single method that receives the file to validate and returns true/false whether it should be accepted or not.

## API

### Filevalidator interface

The primary interface of the system.

**public boolean FileValidator.validate(File file) :**

File is the input file to validate.  
Returns true if all configured modules approved the file according to their configuration (may vary from one application to another). If at least one of the modules rejected the file – the method returns false.

**public FileNameGenerator getFileNameGenerator() :**

Returns a FileNameGenerator object which is configured with the global configuration file. The object is able to generate a new random file name or censor a given file name according to pre-defined rules.[See the FileNameGenerator section for examples.](#_File_Name_Generator)

## Engine

The whole point in FUV is that it can be used in many applications just by adding the JAR and calling the simple API that was described above. Each application will obviously need to configure the requested modules and their internal configuration differently. All modules and utilities configure themselves from a single XML configuration file. In order to the system to properly work, each application should supply such a file (named ‘file-validator-config.xml’) and make it visible by ClassLoader.getSystemResourceAsStream(), meaning, adding its containing directory to the classpath.

FUV uses JAXB to build the module classes from the XML configuration. A sketch of the configuration file and deeper explanations can be found below :

<file-validator-config>

<application-name>Application Name</application-name>

<archive-recursion-depth>7</archive-recursion-depth>

<modules>

…

</modules>

<file-name-generator>

…

</file-name-generator>

<max-file-size>15</max-file-size>

<char-strips>

…

</char-strips>

<types-collections>

…

</types-collections>

</file-validator-config>

Configuration:

1. application-name – the name of the application
2. archive-recursion-depth – the depth allowed for archive/compressed files. In case of “deeper” file – the file will be considered as invalid. The count starts from 0. Please see “[archive/compressed files](#_Archive/compressed_files)” section.
3. Modules – The enabled modules. Please see “[Modules](#_Modules)” section.
4. file-name-generator – utility for safe filename generation. Please see “[File Name Generator](#_File_Name_Generator)” section.
5. max-file-size – utility for size safe uploading file. Please see “[Size Bounded Input Stream](#_Size_Bounded_Input)” section
6. char-strips – Characters strips. Please see “[Char strips](#_Char-strips)” section, “[File Name module](#_File_Name_Module)” section and “[File name generator](#_File_Name_Generator)” section.
7. types-collections – Used to refer a group of file types as one (for example : word and word2007). Please see the “[Types Collections](#_types-collections)” section and the “[File Type Module](#_File_Type_Module)” section.

### Char-strips

Char strip is a list of characters we can use “[File Name module](#_File_Name_Module)” (as allowed characters in filename) and in the “[File name generator](#_File_Name_Generator)” (in order to create safe names).

Each strip has:

1. “stripKey” attribute – The ID of the key. The key is used to refer the strip.
2. “strip” element – the characters in the strip.

<char-strips>

<!-- Digits -->

<char-strip stripKey=*"D"*>

<strip>0123456789</strip>

</char-strip>

<!-- Characters -->

<char-strip stripKey=*"C"*>

<strip>ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz</strip>

</char-strip>

<!-- Others -->

<char-strip stripKey=*"O"*>

<strip>\_-)(</strip>

</char-strip>

</char-strips>

### Types-collections

A type collection is a collection of MIME types. Its purpose is to give the user the ability to refer a group of logically equal types as one and to specify allowed extensions for each type (optionally). For example: different versions of the same program often produce different file types but logically are considered the same (we would like to apply the same behavior for all PowerPoint files regardless of whether they had been created with PowerPoint 97 or 2010).  
 Another usage is due to the Apache-Tika package that FUV uses for MIME type recognition. At some cases, Tika has more than one MIME type for the same input, depending on whether it has a filename or not (for example – DOCX files will be identified as word files if they have a filename but as ‘msoffice’ if they haven’t).  
Allowed extensions, if provided, will be used by the FileTypeModule for a more strict validation (optional, please refer to “[FileTypeModule](#_File_Type_Module)”)

Configuration Example :   
**NOTE:** Type collection’s name and allowed extensions are case insensitive.

<types-collections>

<types-collection name=*"word"*>

<type allowed-exts=*"doc"*>application/msword</type>

<type allowed-exts=*"doc"*>application/x-tika-msoffice</type>

<type allowed-exts=*"docx"*>application/vnd.openxmlformats-officedocument.wordprocessingml.document</type>

<type allowed-exts=*"docx"*>application/x-tika-ooxml</type>

</types-collection>

<types-collection name=*"pdf"*>

<type allowed-exts=*"pdf"*>application/pdf</type>

</types-collection>

<types-collection name=*"JPG"*>

<type allowed-exts=*"jpg,JPEG"*>image/jpeg</type>

</types-collection>

</types-collections>

In this example there are 3 groups of types declared. “word”, “pdf” and “JPG”. The “word” group consists of 4 MIME types that covers all possibilities for MSWord documents. Each type has a list (only one item in these cases) of allowed extensions (“doc” or “docx”). The second group (“pdf”) serves the purpose of declaring the allowed type for a type, and the third one declares that the same type (“image/jpeg”) may have several accepted extensions.  
Notice that the second and third collections are pointless whether you decided not to use the strict extensions checking in the FileTypeModule. The first one actually groups a few types into a single collection.

### Archive/compressed files

In case the file we want to validate is an archive/compressed file, it is opened in temporary directory and the inner files can be scanned too. In case one of the inner files is archive/compressed file too, the same operation is done recursively.

The supported archive file formats are TAR and ZIP. The supported compressed file formats are GZIP and BZIP2.

Configuration:

The recursion maximum depth allowed (actually, the maximum file depth allowed) is configured in the engine:

<archive-recursion-depth>7</archive-recursion-depth>

The count starts from 0 (archive-recursion-depth=0 means that archive/compressed file are not allowed).

Important Notes:

* Inner **directories** are not scanned (but the inner files inside the directories are scanned)
* The inner file **name** inside compressed files (GZIP/BZIP2) is not scanned.

## Modules

The “modules” element contains all the enabled modules in the system. All modules have “scanInnerFiles” attribute (“true” by default) and unique configuration. In case “scanInnerFiles” is “true” and the validated file is archive/compressed file, the module will scan the inner files too.

<modules>

<!-- File name module -->

<file-name-module>

<max-file-name-length>50</max-file-name-length>

<allowedCharStrips>D C O</allowedCharStrips>

</file-name-module>

<!-- Anti Virus module -->

<anti-virus-module scanInnerFiles=*"false"*>

<anti-virus-path>bin/av\_wrapper.sh</anti-virus-path>

<success-rc>0</success-rc>

</anti-virus-module>

<!-- File type module -->

<file-type-module>

<allowed-types>word jpg application/x-gzip application/x-gtar application/zip text/plain application/x-bzip2</allowed-types>

<force-ext-check/>

</file-type-module>

<!-- File permissions module-->

<unix-file-permissions-module scanInnerFiles=*"false"*>

<user-max-permissions>rwx</user-max-permissions>

<group-max-permissions>r-x</group-max-permissions>

<all-max-permissions>r-x</all-max-permissions>

</unix-file-permissions-module>

</modules>

### File Type Module

Purpose: The module handles invalid file types according to a predefined set of accepted MIME types.  
This module uses [Apache-Tika](http://tika.apache.org/) for content analysis of the file.

The module is configured by a list of allowed types (and/or [types-collections](#_types-collections)) and by an optional element to enforce extension check (default is not to check).

Configuration:

<file-type-module>

<allowed-types>word jpg application/x-gzip application/x-gtar application/zip text/plain application/x-bzip2</allowed-types>

<force-ext-check/>

</file-type-module>

1. allowed-types : A list of space/comma separated values. Each value can be a MIME type (as Apache-Tika would give) or a pre-defined types-collection. Both an explicit type and a types-collection are case insensitive.
2. force-ext-check : An optional element. If not given – extension check will NOT take place. If given – extension check WILL take place unless it contains “false” or “no” as text value (case insensitive).

**NOTICE :**  FUV only knows allowed extensions for types that are in a types-collection (and allowed-exts attribute was set). It will accept any extension for a type without specific allowed-exts. For example : if you configure FileTypeModule with *<allowed-types>application/x-gzip</allowed-types>*, the module will accept all extensions for a gzip file even if *<force-ext-check/>* was set. If you want it to accept only a specific extension, you should use a type collection and specify in its allowed-exts attribute the extension. [Refer to types collection manual for further examples.](#_types-collections)

### File Name Module

Purpose: to handle invalid file names.

The module validates the file name string (only the simple file name without the extension or the full path):

1. Filename length – validation for filename length
2. Filename characters – validation for filename characters

Configuration:

The module configuration contains:

1. max-file-name-length – The maximum length allowed for simple filename without extension or full path
2. allowedCharStrips – IDs of all char strips allowed in the name, separated by whitespace. In the following example, the IDs are “D” (the “digits” strip) and “O” (the “others” strip). The union list of char strips in this field is the list of characters allowed. The validation is according to “whitelist”: All filename characters in the name must appear in the union list.

<file-name-module>

<max-file-name-length>50</max-file-name-length>

<allowedCharStrips>D O</allowedCharStrips>

</file-name-module>

The [characters strips](#_Char-strips) are configured on the engine part (and not inside the modules element) and for each strip, the stripKey is its ID.

### UNIX File Permissions Module

Important Note: this module can be enabled only in UNIX environment. (based on ‘ls’ command)

Purpose: The module validates that the file on the server has the appropriate permissions. It can be used to alert for a problem with server’s configuration that causes files to be stored in the disk with execute permissions (for example).

Configuration:

The module is configured by 3 “maximal” allowed permissions for the user, group and all (similar to UNIX file permissions).

<unix-file-permissions-module scanInnerFiles=*"false"*>

<user-max-permissions>rwx</user-max-permissions>

<group-max-permissions>r-x</group-max-permissions>

<all-max-permissions>r-x</all-max-permissions>

</unix-file-permissions-module>

The module will reject any file that has a permission that wasn’t allowed in at least one of the groups. In the case above, a file with only group-write permission (020) will be rejected (group max permissions doesn’t contain write), while a file with execute permission for all will be allowed.

Obviously, the configuration above is only for example purposes, you would probably want to use this module with restrictions such as user=rw , group=r , all=r (if any…).

### Anti-Virus Module

Purpose: To scan the file with Anti-Virus

Configuration:

1. anti-virus-path – The Anti-Virus binary/wrapper script path. We strongly recommend a wrapper script to easily follow the return code requirements.
2. success-rc – The “success” return code of the script.
3. scanInnerFiles="false" – there is no point scanning the inner files again after the original file was scanned.

<anti-virus-module scanInnerFiles=*"false"*>

<anti-virus-path>bin/av\_wrapper.sh</anti-virus-path>

<success-rc>0</success-rc>

</anti-virus-module>

Requirements from the AV binary/wrapper script :

1. Must be a runnable file on your host OS. Either a compiled binary, a script (with an appropriate shebang line) or anything that a default shell can executes (without aliases or startup scripts like .bashrc running before if it’s a UNIX environment).
2. Must return a constant specific return code for success. All others will be considered as a failure. If your AV may return more than one, wrap it as a script as shown in ‘bin/av\_wrapper.sh’.
3. Must receive a single argument of the file path to run on. The return code will refer to the given file.

## Utilities

Additional utilities the FUV package provides :

### File Name Generator

Purpose: To allow the user generate safe filenames. The Generator contains 2 methods:

1. censorFilename(String fileName) – Censors the given filename: limits the filename length and removes not-allowed characters. If the name after removing the not-allowed characters is longer than allowed, we take the max-file-name-length first characters. If the new filename is empty (none of the original characters was allowed), **FilenameGenerationException** is thrown. The max-file-name-length is not including the extension, and the extension will be added back to the new file.
2. generateNewRandomFilename() - Generates a random file name according to the pattern from the configuration. If the new filename is empty, **FilenameGenerationException** is thrown.

Configuration:

* Censor method:
  1. max-file-name-length – the maximum length of the new filename.
  2. charStripsToKeep – The characters from the original name we want to keep. The [characters strips](#_Char-strips) are configured on the engine part
* Generation method: name-pattern is the pattern of the name to generate. The pattern contain “part”s. each part has “strip” ID to use, and how many characters from the strip to use. . The [characters strips](#_Char-strips) are configured on the engine part.

<file-name-generator>

<!-- Censor method -->

<max-file-name-length>30</max-file-name-length>

<charStripsToKeep>C O</charStripsToKeep>

<!-- Generation method -->

<name-pattern>

<part><strip>C</strip><length>10</length></part>

<part><strip>D</strip><length>2</length></part>

</name-pattern>

</file-name-generator>

How to use:

* Create FileName generator:  
  FileValidator fv = FileValidatorImpl.getInstance();   
  FileNameGenerator gen = fv.getFileNameGenerator();
* Generate name:  
  String name1 = gen.generateNewRandomFilename();
* Censor name:  
  String name2 = gen.censorFilename("sf3fsf\_t54ha.doc");

### Size Bounded Input Stream

Purpose: To create safe way to upload a file without a problem with it size. SizeBoundedInputStream is an InputStream that warps the original InputStream and count the bytes the user read. In case the the number reached the maximum allowed, it returns -1 (EOF) and set the **limitReached** flag to “true”.

Configuration: max-file-size – Maximum file size in Bytes.

<max-file-size>1024</max-file-size>

How to use:

* public SizeBoundedInputStream(InputStream is) – Constructor – set the maximum size allowed to the one in configuration
* public SizeBoundedInputStream(InputStream is, long maxSize) – Constructor – set the maximum size allowed to the one in the parameters
* public int read() throws IOException – overrides InputStream
* public boolean hasReachedLimit() – returns the flag

## Logging

FUV uses SLF4J as logging façade and logback as the logging implementation. Logback’s configuration file can be found at src/main/resources/logback.xml and is a standard logback file. Logback supports a variety of logging features. [A logback manual can be found here](http://logback.qos.ch/manual/index.html).   
If for any reason you decide to abandon logback and go with other implementation (log4j?) you can easily switch by just replacing the dependencies. This was the primary reason why we decided to use SLF4J.

[SLF4J documentation can be found here.](http://www.slf4j.org/docs.html)

## Usage Examples

### Tika Type directory tester

Usage: TikaTypeDirectoryTester <dir path>

Receives a path of a directory, iterates over all inner files and prints the file name and the MIME type that tika discovered. It is useful when you want to add a new type to the XML configuration file and need to know the exact tika type name.

### File Validator Directory Tester

Usage: FileValidatorDirectoryTester <dir path>

Receives a path of a directory, iterates over all inner file and run FUV with the default configuration file. Prints the file name and whether it’s valid or not. It is useful when you want to check FUV on some specific files before you fully integrate into your system.

## Configuration file example

<file-validator-config>

<application-name>Application Name</application-name>

<!-- Starting from 0 (0 = archive/compressed file are not allowed) -->

<archive-recursion-depth>7</archive-recursion-depth>

<modules>

<!-- File name module -->

<file-name-module>

<max-file-name-length>50</max-file-name-length>

<allowedCharStrips>D C O</allowedCharStrips>

</file-name-module>

<!-- Anti Virus module -->

<anti-virus-module scanInnerFiles=*"false"*>

<anti-virus-path>bin/av\_wrapper.sh</anti-virus-path>

<success-rc>0</success-rc>

</anti-virus-module>

<!-- File type module -->

<file-type-module>

<allowed-types>word jpg application/x-gzip application/x-gtar application/zip text/plain application/x-bzip2</allowed-types>

<force-ext-check/>

</file-type-module>

<!-- File permissions module-->

<unix-file-permissions-module scanInnerFiles=*"false"*>

<user-max-permissions>rwx</user-max-permissions>

<group-max-permissions>r-x</group-max-permissions>

<all-max-permissions>r-x</all-max-permissions>

</unix-file-permissions-module>

</modules>

<file-name-generator>

<!-- Censor method -->

<max-file-name-length>30</max-file-name-length>

<charStripsToKeep>C O</charStripsToKeep>

<!-- Generation method -->

<name-pattern>

<part><strip>C</strip><length>10</length></part>

<part><strip>D</strip><length>2</length></part>

</name-pattern>

</file-name-generator>

<max-file-size>15</max-file-size>

<char-strips>

<!-- Digits -->

<char-strip stripKey=*"D"*>

<strip>0123456789</strip>

</char-strip>

<!-- Characters -->

<char-strip stripKey=*"C"*>

<strip>ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz</strip>

</char-strip>

<!-- Others -->

<char-strip stripKey=*"O"*>

<strip>\_-)(</strip>

</char-strip>

</char-strips>

<types-collections>

<types-collection name=*"word"*>

<type allowed-exts=*"doc"*>application/msword</type>

<type allowed-exts=*"doc"*>application/x-tika-msoffice</type>

<type allowed-exts=*"docx"*>application/vnd.openxmlformats-officedocument.wordprocessingml.document</type>

<type allowed-exts=*"docx"*>application/x-tika-ooxml</type>

</types-collection>

<types-collection name=*"pdf"*>

<type allowed-exts=*"pdf"*>application/pdf</type>

</types-collection>

<types-collection name=*"JPG"*>

<type allowed-exts=*"jpg,JPEG"*>image/jpeg</type>

</types-collection>

</types-collections>

</file-validator-config>