**Unrestricted File Upload**

Thank you for visiting OWASP.org. We have migrated our community to a new web platform and regretably the content for this page needed to be programmatically ported from its previous wiki page. There’s still some work to be done.

**Description**

Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.

There are really two classes of problems here. The first is with the file metadata, like the path and file name. These are generally provided by the transport, such as HTTP multi-part encoding. This data may trick the application into overwriting a critical file or storing the file in a bad location. You must validate the metadata extremely carefully before using it.

The other class of problem is with the file size or content. The range of problems here depends entirely on what the file is used for. See the examples below for some ideas about how files might be misused. To protect against this type of attack, you should analyse everything your application does with files and think carefully about what processing and interpreters are involved.

**Risk Factors**

* The impact of this vulnerability is high, supposed code can be executed in the server context or on the client side. The likelihood of detection for the attacker is high. The prevalence is common. As a result the severity of this type of vulnerability is high.
* It is important to check a file upload module’s access controls to examine the risks properly.
* Server-side attacks: The web server can be compromised by uploading and executing a web-shell which can run commands, browse system files, browse local resources, attack other servers, or exploit the local vulnerabilities, and so forth.
* Client-side attacks: Uploading malicious files can make the website vulnerable to client-side attacks such as [XSS](https://owasp.org/www-community/vulnerabilities/Cross-site_Scripting_/(XSS/)) or Cross-site Content Hijacking.
* Uploaded files can be abused to exploit other vulnerable sections of an application when a file on the same or a trusted server is needed (can again lead to client-side or server-side attacks)
* Uploaded files might trigger vulnerabilities in broken libraries/applications on the client side (e.g. iPhone MobileSafari LibTIFF Buffer Overflow).
* Uploaded files might trigger vulnerabilities in broken libraries/applications on the server side (e.g. ImageMagick flaw that called ImageTragick!).
* Uploaded files might trigger vulnerabilities in broken real-time monitoring tools (e.g. Symantec antivirus exploit by unpacking a RAR file)
* A malicious file such as a Unix shell script, a windows virus, an Excel file with a dangerous formula, or a reverse shell can be uploaded on the server in order to execute code by an administrator or webmaster later – on the victim’s machine.
* An attacker might be able to put a phishing page into the website or deface the website.
* The file storage server might be abused to host troublesome files including malwares, illegal software, or adult contents. Uploaded files might also contain malwares’ command and control data, violence and harassment messages, or steganographic data that can be used by criminal organisations.
* Uploaded sensitive files might be accessible by unauthorised people.
* File uploaders may disclose internal information such as server internal paths in their error messages.

**Examples**

**Attacks on application platform**

* Upload .jsp file into web tree - jsp code executed as the web user
* Upload .gif file to be resized - image library flaw exploited
* Upload huge files - file space denial of service
* Upload file using malicious path or name - overwrite a critical file
* Upload file containing personal data - other users access it
* Upload file containing “tags” - tags get executed as part of being “included” in a web page
* Upload .rar file to be scanned by antivirus - command executed on a server running the vulnerable antivirus software

**Attacks on other systems**

* Upload .exe file into web tree - victims download trojaned executable
* Upload virus infected file - victims’ machines infected
* Upload .html file containing script - victim experiences [Cross-site Scripting (XSS)](https://owasp.org/www-community/vulnerabilities/Cross-site_Scripting_/(XSS/))
* Upload .jpg file containing a Flash object - victim experiences Cross-site Content Hijacking.
* Upload .rar file to be scanned by antivirus - command executed on a client running the vulnerable antivirus software

**Weak Protections and Bypassing Methods**

**Deny Listing File Extensions**

This protection might be bypassed by:

* Finding missed extensions that can be executed on the server side or can be dangerous on the client side (e.g. “.php5”, “.pht”, “.phtml”, “.shtml”, “.asa”, “.cer”, “.asax”, “.swf”, or “.xap”).
* Finding flaws in a web server configuration when it parses files with double extensions or it executes them by providing a sensitive extension after a delimiter such as “/” or “;” character (e.g. “/file.jpg/index.php” when the “file.jpg” file contains PHP code and has been uploaded)
  + In Apache, a php file might be executed using the double extension technique such as “file.php.jpg” when “.jpg” is allowed.
  + In IIS6 (or prior versions), a script file can be executed by using one of these two methods:
    - by adding a semi-colon character after the forbidden extension and before the permitted one (e.g. “file.asp;.jpg”)
    - by renaming a script file’s extension (e.g. “.asp”) to an allowed extension (e.g. “.txt”) in a folder that its name ends with the script’s extension (e.g. “folder.asp\file.txt”). In Windows, it is possible to create a directory by using a file uploader and ADS (Alternate Data Stream). In this method, a filename that ends with “::$Index\_Allocation” or “:$I30:$Index\_Allocation” makes the file uploader to create a directory rather than a file (e.g. “folder.asp::$Index\_Allocation” creates “folder.asp” as a directory).
* Changing a number of letters to their capital forms to bypass case sensitive rules (e.g. “file.aSp” or “file.PHp3”).
* Using Windows 8.3 feature, it is possible to replace the existing files by using their shortname (e.g. “web.config” can be replaced by “web~1.con” or “.htaccess” can be replaced by “HTACCE~1”)
* Finding characters that are converted to other useful characters during the file upload process. For instance, when running PHP on IIS, the “>”, “<”, and double quote “ characters respectively convert to “?”, “\*”, and “.” characters that can be used to replace existing files (e.g. “web<<” can replace the “web.config” file). In order to include the double quote character in the filename in a normal file upload request, the filename in the “Content-Disposition” header should use single quotes (e.g. filename=’web”config’ to replace the “web.config” file).
* Finding neutral characters after a filename such as trailing spaces and dots in Windows filesystem or dot and slash characters in a Linux filesystem. These characters at the end of a filename will be removed automatically (e.g. “file.asp … … . . .. ..”, “file.asp “, or “file.asp.”). Although slash or backslash characters are also normally problematic characters, they can be ignored in a normal file upload request as anything before these characters may count as the directory name on the server-side; that said, they should be tried for a thorough test (e.g. “test.php/” or “test.php.\”).
* Finding flaws in extension detection techniques. A web server may use the first extension after the first dot (“.”) in the provided filename or use a flawed algorithm to detect the extension when there is none or multiple dot characters (e.g. “file.txt.jpg.php”).
* Using control characters such as null character (0x00) after a forbidden extension and before a permitted one may lead to a bypass. In this method, all the strings after the Null character will be discarded when saving the files. Both URL-encoded and decoded version of the null character should be tried in a file upload request for a thorough test.
* Using NTFS alternate data stream (ADS) in Windows. In this case, a colon character “:” will be inserted after a forbidden extension and before a permitted one. As a result, an empty file with the forbidden extension will be created on the server (e.g. “file.asax:.jpg”). This file might be edited later using other techniques such as using its short filename. The “::$data” pattern can also be used to create non-empty files. Therefore, adding a dot character after this pattern might also be useful to bypass further restrictions (.e.g. “file.asp::$data.”)
* Flaws in the protection mechanism when it replaces dangerous extensions. For instance, “file.p.phphp” might be changed to “file.php” after going through this functionality.
* Flaws in the uploaded file usage for instance when a PHP application uses the “include” function to show the uploaded images.
* Combination of the above techniques.

**Beating getimagesize()**

The getimagesize() function will check if it is an image and will check “mime” to verify image type.

Insecure Configuration :

 <FilesMatch ".+\.ph(p([3457s]|\-s)?|t|tml)">  SetHandler application/x-httpd-php  </FileMatch>

Secure Configuration :

 <FilesMatch ".+\.ph(p([3457s]|\-s)?|t|tml)$">  SetHandler application/x-httpd-php  </FileMatch>

If the service is up an running with the Insecure Configuration, any one can beat the getimagesize function by writing comments in GIF file.

For that an end user needs to install an utility in Kali/Ubuntu OS named ‘gifsicle’

 For Kali Linux : apt-get install gifsicle  For Ubuntu : sudo apt-get install gifsicle

Once installed, the below commands will help writing the commands in a gif file.

 gifsicle < mygif.gif -- comment "

<?php echo ‘Current PHP version: ‘ . phpversion(); ?>

” > output.php.gif

The above command will create a file with the name “output.php.gif” which simply needs to be uploaded during the check of file upload vulnerability.

**Allow Listing File Extensions**

Applications that check the file extensions using an allow list method also need to validate the full filename to prevent any bypass.

* The list of permitted extensions should be reviewed as it can contain malicious extensions as well. For instance, in case of having “.shtml” in the list, the application can be vulnerable to SSI attacks.
* Some of the bypass techniques for the deny list methods such as using double extensions are also applicable here and should be checked.

**“Content-Type” Header Validation**

“Content-Type” entity in the header of the request indicates the Internet media type of the message content. Sometimes web applications use this parameter in order to recognise a file as a valid one. For instance, they only accept the files with the “Content-Type” of “text/plain”.

* It is possible to bypass this protection by changing this parameter in the request header using a web proxy.

**Using a File Type Detector**

Sometimes web applications intentionally or unintentionally use some functions (or APIs) to check the file types in order to process them further. For instance, when an application resize an image file, it may just show an error message when non-image files are uploaded without saving them on the server.

* If it reads the few first characters (or headers), it can be bypassed by inserting malicious code after some valid header or within the file’s metadata.
* Inserting code in the comments section or those section that have no effect on the main file can also lead to a bypass.
* The inserted data can be obfuscated or encoded if the application detects a malicious code using specific patterns or signatures.
* Uploaded file can be crafted to create a malicious code in case of being compressed by the application.

**Other Interesting Test Cases**

* Uploading a file when another file with the same name already exists. This may show interesting error messages that can lead to information disclosure. Logical flaws might be found if the application renames the new file to keep it on the server.
* Uploading a file when another folder with the same name already exists. This may show interesting error messages that can lead to information disclosure.
* Uploading a file with a long name. This may show interesting error messages that can lead to information disclosure.
* Uploading a file multiple times at the same time. This may show interesting error messages that can lead to information disclosure.
* Uploading valid and invalid files in different formats such as compressed or XML files to detect any possible processing on the server side.
* Uploading a file with “.”, “..”, or “…” as its name. For instance, in Apache in Windows, if the application saves the uploaded files in “/www/uploads/” directory, the “.” filename will create a file called “uploads” in the “/www/” directory.
* Uploading files that may not be deleted easily such as “…:.jpg” in NTFS that makes the “…” file (this file can be deleted using command line). This may show interesting error messages that can lead to information disclosure.
* Uploading a file in Windows with invalid characters such as |<>\*?” in its name. This may show interesting error messages that can lead to information disclosure.
* Uploading a file in Windows using reserved (forbidden) names such as CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9. This may show interesting error messages that can lead to information disclosure. Uploading a file with a reserved name may lead to denial of service if the application keeps the name and tries to save it with another extension (detecting it wrongly as an existing file).
* Cross-site content hijacking issues can be exploited by uploading a file with allowed name and extension but with Flash, PDF, or Silverlight contents.
* Uploading a “crossdomain.xml” or “clientaccesspolicy.xml” file can make a website vulnerable to cross-site content hijacking. These files should be uploaded to the root of the website to work. However, the “crossdomain.xml” file can be in a subdirectory as long as it is allowed in the root “crossdomain.xml” file.

**Important Notes in Testing File Uploaders**

* Do not try to replace the existing files during testing unless it is safe to proceed. For instance, replacing configuration files such as “web.config” or “.htaccess” file can lead to a denial of service attack for the whole website.

**Prevention Methods (Solutions to be more secure)**

In order to make a Windows server more secure, it is very important to follow the Microsoft security best practices first. For this purpose, some of the useful links are:

* IIS 6.0 Security Best Practices[<http://technet.microsoft.com/en-us/library/cc782762(WS.10>).aspx]
* Securing Sites with Web Site Permissions[<http://technet.microsoft.com/en-us/library/cc756133(WS.10>).aspx]
* IIS 6.0 Operations Guide[<http://technet.microsoft.com/en-us/library/cc785089(WS.10>).aspx]
* Improving Web Application Security: Threats and Countermeasures[1](http://msdn.microsoft.com/en-us/library/ms994921.aspx)
* Understanding the Built-In User and Group Accounts in IIS 7.0[2](https://learn.iis.net/page.aspx/140/understanding-the-built-in-user-and-group-accounts-in-iis-70/)
* IIS Security Checklist[3](http://windows.stanford.edu/docs/IISsecchecklist.htm)

And some special recommendations for the developers and webmasters:

* The file types allowed to be uploaded should be restricted to only those that are necessary for business functionality.
* Never accept a filename and its extension directly without having an allow list filter.
* The application should perform filtering and content checking on any files which are uploaded to the server. Files should be thoroughly scanned and validated before being made available to other users. If in doubt, the file should be discarded.
* It is necessary to have a list of only permitted extensions on the web application. And, file extension can be selected from the list. For instance, it can be a “select case” syntax (in case of having VBScript) to choose the file extension in regards to the real file extension.
* All the control characters and Unicode ones should be removed from the filenames and their extensions without any exception. Also, the special characters such as “;”, “:”, “>”, “<”, “/” ,”\”, additional “.”, “\*”, “%”, “$”, and so on should be discarded as well. If it is applicable and there is no need to have Unicode characters, it is highly recommended to only accept Alpha-Numeric characters and only 1 dot as an input for the file name and the extension; in which the file name and also the extension should not be empty at all (regular expression: ^\[a-zA-Z0-9\]{1,200}\\.\[a-zA-Z0-9\]{1,10}$).
* Limit the filename length. For instance, the maximum length of the name of a file plus its extension should be less than 255 characters (without any directory) in an NTFS partition.
* It is recommended to use an algorithm to determine the filenames. For instance, a filename can be a hash of the name of file plus the date of the day.
* Uploaded directory should not have any “execute” permission and all the script handlers should be removed from these directories.
* Limit the file size to a maximum value in order to prevent denial of service attacks (on file space or other web application’s functions such as the image resizer).
* Restrict small size files as they can lead to denial of service attacks. So, the minimum size of files should be considered.
* Use Cross Site Request Forgery protection methods.
* Prevent from overwriting a file in case of having the same hash for both.
* Use a virus scanner on the server (if it is applicable). Or, if the contents of files are not confidential, a free virus scanner website can be used. In this case, file should be stored with a random name and without any extension on the server first, and after the virus checking (uploading to a free virus scanner website and getting back the result), it can be renamed to its specific name and extension.
* Try to use POST method instead of PUT (or GET!)
* Log users’ activities. However, the logging mechanism should be secured against log forgery and code injection itself.
* In case of having compressed file extract functions, contents of the compressed file should be checked one by one as a new file.
* If it is possible, consider saving the files in a database rather than on the filesystem.
* If files should be saved in a filesystem, consider using an isolated server with a different domain to serve the uploaded files.
* File uploaders should be only accessible to authenticated and authorised users if possible.
* Write permission should be removed from files and folders other than the upload folders. The upload folders should not serve any
* Ensure that configuration files such as “.htaccess” or “web.config” cannot be replaced using file uploaders. Ensure that appropriate settings are available to ignore the “.htaccess” or “web.config” files if uploaded in the upload directories.
* Ensure that files with double extensions (e.g. “file.php.txt”) cannot be executed especially in Apache.
* Ensure that uploaded files cannot be accessed by unauthorised users.
* Adding the “Content-Disposition: Attachment” and “X-Content-Type-Options: nosniff” headers to the response of static files will secure the website against Flash or PDF-based cross-site content-hijacking attacks. It is recommended that this practice be performed for all of the files that users need to download in all the modules that deal with a file download. Although this method does not fully secure the website against attacks using Silverlight or similar objects, it can mitigate the risk of using Adobe Flash and PDF objects, especially when uploading PDF files is permitted.
* Flash/PDF (crossdomain.xml) or Silverlight (clientaccesspolicy.xml) cross-domain policy files should be removed if they are not in use and there is no business requirement for Flash or Silverlight applications to communicate with the website.
* Browser caching should be disabled for the crossdomain.xml and clientaccesspolicy.xml files. This enables the website to easily update the file or restrict access to the Web services if necessary. Once the client access policy file is checked, it remains in effect for the browser session so the impact of non-caching to the end-user is minimal. This can be raised as a low or informational risk issue based on the content of the target website and security and complexity of the policy file(s).
* CORS headers should be reviewed to only be enabled for static or publicly accessible data. Otherwise, the “Access-Control-Allow-Origin” header should only contain authorised addresses. Other CORS headers such as “Access-Control-Allow-Credentials” should only be used when they are required. Items within the CORS headers such as “Access-Control-Allow-Methods” or “Access-Control-Allow-Headers” should be reviewed and removed if they are not required.