Improve File Uploaders' Protections

Especially for Windows-based web applications

Revision: 1.0

Soroush Dalili
irsdl a.t yahoo d.o.t com
soroush.secproject.com
Feb. 2010

Improve File Uploaders' Protections

1 Introduction

Uploading files by using web applications is very common. However, there is always a high risk around this matter. In case of uploading a web-shell file which can be absolutely malicious, an attacker can get the same privilege of access as the web application to the server. In this paper, which is mostly around the Windows-based web applications, some general solutions for protecting against this type of attack have been suggested. Moreover, as a proof of concept, some of the most general protection methods and the way of bypassing them have been discussed.

This article is an educational article to improve the security of the web applications. And, the author of this article ("Soroush Dalili") does not accept and has no responsibility about the content or usage of this article in any other way. Any other usage of this article except the legal ones is completely prohibited.

Please respect the copyright and mention the name of the author ("Soroush Dalili") in case of using this article.

2 Prevention Methods (Solutions to be more secure)

As this paper wants to be around the Windows-based web applications, 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
- Securing Sites with Web Site Permissions
- IIS 6.0 Operations Guide
- Improving Web Application Security: Threats and Countermeasures
- Understanding the Built-In User and Group Accounts in IIS 7.0
- IIS Security Checklist

And some special recommendations for the developers and webmasters:

1) Never accept a filename and its extension directly without having a white-list filter.
2) 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 regard to the real file extension.
3) All the control characters[1] 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

[1] http://en.wikipedia.org/wiki/Control_character

and also the extension should not be empty at all (regular expression: **[a-zA-Z0-9]{1,200}\.[a-zA-Z0-9]{1,10}**).

4) 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.
5) It is recommended to use an algorithm to determine the filenames. For instance, a filename can be a MD5 hash of the name of file plus the date of the day.
6) Uploaded directory should not have any "execute" permission.
7) 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).
8) Restrict small size files as they can lead to denial of service attacks. So, the minimum size of files should be considered.
9) Use Cross Site Request Forgery protection methods.
10) Prevent from overwriting a file in case of having the same hash for both.
11) 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.
12) Try to use POST method instead of PUT (or GET!)
13) Log users' activities. However, the logging mechanism should be secured against log forgery and code injection itself.
14) In case of having compressed file extract functions, contents of the compressed file should be checked one by one as a new file.

3 Weak Protection Methods and Methods of Bypassing

3.1 Using Black-List for Files' Extensions:

Some web applications still use only a black-list of extensions to prevent from uploading a malicious file.

3.1.1 Bypass Method(s):

1) It is possible to bypass this protection by using some extensions which are executable on the server but are not mentioned in the list. (Example: "file.php5", "file.shtml", "file.asa", or "file.cer")
2) Sometimes it is possible to bypass this protection by changing some letters of extension to the capital form (example: "file.aSp" or "file.PHp3").
3) Using trailing spaces and/or dots at the end of the filename can sometimes cause bypassing the protection. These spaces and/or dots at the end of the filename will be removed when the file wants to be saved on the hard disk automatically. The filename can be sent to the server by using a local proxy or using a simple script (example: "file.asp ... ...").

4) In case of using insecure IIS6 (or prior versions), it might be possible to bypass this protection by adding a semi-colon after the forbidden extension and before the permitted extension (example: "file.asp;.jpg")[2].
5) This protection can be completely bypassed by using the most famous control character which is Null character (0x00) after the forbidden extension and before the permitted one. In this method, during the saving process all the strings after the Null character will be discarded. Putting a Null character in the filename can be simply done by using a local proxy or by using a script (example: "file.asp%00.jpg"). Besides, it would be perfect if the Null character is inserted directly by using the Hex view option of a local proxy such as Burpsuite or Webscarab in the right place (without using %).
6) It is also possible to create a file with a forbidden extension by using NTFS alternate data stream (ADS). In this case, a ":" sign will be inserted after the forbidden extension and before the permitted one. As a result, an empty file with the forbidden extension will be created on the server (example: "file.asp:.jpg"). Attacker can try to edit this file later to execute his/her malicious codes. However, an empty file is not always good for an attacker. So, there is an invented method by the author of this paper in which an attacker can upload a non-empty shell file by using the ADS. In this method, a forbidden file can be uploaded by using this pattern: "file.asp::$data.".
7) Sometimes combination of the above can lead to bypassing the protections.

3.2 Using White-List for Files' Extensions:

Many web applications use a white-list to accept the files' extensions. Although using white-list is one of the recommendations, it is not enough on its own. Without having input validation, there is still a chance for an attacker to bypass the protections.

3.2.1 Bypass Method(s):

1) The 3rd method of 3.1.1.
2) The 4th method of 3.1.1.
3) The 5th method of 3.1.1.
4) The 6th method of 3.1.1.
5) The list of permitted extensions should be reviewed as it can contain malicious extension as well. For instance, in case of having ".shtml" in the list, the application can be vulnerable to SSI attacks.

3.3 Using "Content-Type" from the Header:

"Content-Type" entity in the header of the request indicates the Internet media type of the message content[3]. Sometimes web applications use this parameter in order to recognize a file as a good one. For instance, they only accept the files with the "Content-Type" of "text/plain".

3.3.1 Bypass Method(s):

[2] IIS 6 Semi-Colon Vulnerability (http://soroush.secproject.com/downloadable/iis-semicolon-report.pdf)
[3] http://en.wikipedia.org/wiki/MIME#Content-Type

1) It is possible to bypass this protection by changing this parameter in the request header by using a local proxy.

3.4 Using File Type Recogniser:

Sometimes web applications intentionally or unintentionally use some functions (or APIs) to check the type of the file in order to do further process. For instance, in case of having image resizing, it is probable to have image type recogniser.

3.4.1 Bypass Method(s):

1) Sometimes the recognisers just read the few first characters (or header) of the files in order to check them. In this case, an attacker can insert the malicious code after some valid header.
2) There are always some places in the structure of the files which are for the comments section and have no effect on the main file. And, an attacker can insert malicious codes in these points.
3) Also, it is not impossible to think about a file modifier (for example an image resizer) which produces malicious codes itself in case of receiving special input.

4 Additional Note(s)

There are some other things that developers and webmasters should have a good knowledge about:

- In case of having folder (directory) creation function, it should be secured similar to the files' extensions and also it should be protected against the directory traversal attacks. Otherwise, an attacker might be able to create a directory which has execution permission and bypass the protections.
- In case of having "rename", "move", and "edit" functions for the files or directories, the same protection should be applied for all of them. In other words, it should be considered as a new file or folder in case of having "rename", "move", or "edit" command. Do not forget that there is no difference between rename and move functions as both of them can do the same thing. So, both of them should be protected against the directory traversal attacks as well.
- Sometimes the protections can be bypassed by creating a new directory first, and upload files in it. For instance, there is no ".htaccess" anymore to protect the new folder from executing "PHP" files. Moreover, sometimes there is no permission to upload a file, but it is possible to upload the files in a newly created directory.
- Sometimes protections can be bypassed by uploading a file, similar to an existing filename. For instance, uploading a file with the name of "default.aspx" may bypass the protections. Or as another example, an attacker may upload a ".htaccess" file in order to rewrite the protection rules.
- Usually an attacker renames the forbidden extension to a permitted one and tries to upload it by using a local proxy. Then, he/she changes the file extension to whatever he/she wants by using that local proxy before sending it to the server. In this method, the client side Javascripts can also be bypassed without any problem, and all request parameters would be in the correct format except the filename. They always assume that there is a value in the cookies which

should be set to a special value by a JavaScript on checking and submitting the form's information. So, never trust the client side's protections.

- Sometimes it is not possible to find the upload directory or temporary upload directory. An attacker can send a long file (or directory) name or use forbidden filenames[4] (such as "com5") in order to get an error to find the upload directories. And in case of having a temporary folder to modify the files before putting them in the right directory (for example for resizing the images), an attacker might be able to find this directory and execute the malicious files immediately after uploading them (before being deleted). So, the error messages should be hidden and also all the temporary directories should be out of web URL access.
- Alternate data stream feature can lead to some further issues in case of accepting ":" character. First, an attacker can hide some information inside the other files. This hidden data can be used later.

  Moreover, an attacker can create an irremovable file in order to keep his/her malicious code on the server. As an example: "irremovable.php.::$Data"

  It is also possible to create a directory by using ADS feature. For example the "test.directory::$index_allocation" ,as a file name, will create "test.directory" folder on the web server.

5 General Examples

As examples of a secured and insecure version of a file uploader, FCKEditor is the best one for both. Security functions of its file uploader in the new version cannot be bypassed by the methods which are discussed in this paper.

5.1 Insecure:

- For instance, FCKEditor 2.4.3 (and prior versions) uses a long list of the bad extensions in its ASP uploader:

```
44  ConfigDeniedExtensions.Add  "File",
    "html|htm|php|php2|php3|php4|php5|phtml|pwml|inc|asp|aspx|ascx|jsp|cfm|
    cfc|pl|bat|exe|com|dll|vbs|js|reg|cgi|htaccess|asis|sh|shtml|shtm|phtm"
```

  And it is vulnerable to:

  - The 1st method of 3.1.1. An attacker can upload an ASP file by renaming its extension to ".asa" or ".cer".
  - The 3rd method of 3.1.1. An attacker can upload a dangerous file by using trailing dots and spaces.
  - The 4th method of 3.1.1. Code execution depends on the IIS version and its patch.
  - The 5th method of 3.1.1. Null character should be inserted directly into the filename as it is shown here:

29	46	69	6c	65	22	3b	20	66	69	6c	65	6e	61	6d	65	3d	File"; filename=
2a	22	6d	61	6c	69	63	69	6f	75	73	2e	61	73	70	00	2e	"malicious.asp .
2b	67	69	66	22	0d	0a	43	6f	6e	74	65	6e	74	2d	54	79	gif"Content-Ty
2c	70	65	3a	20	69	6d	61	67	65	2f	67	69	66	0d	0a	0d	pe: image/gif

[4] http://en.wikipedia.org/wiki/Filename

- The 6th method of 3.1.1.

5.2 Secured:

It is nearly impossible to say that an application is completely secure. However, as a highly secured file uploader, ASP version of FCKFinder 2.6.6 is a good example (http://sourceforge.net/projects/fckeditor/files/). As it is secure against the methods which were mentioned here (revision 1.0).