



File Upload Considerations

White Paper

Prepared by: Kirk Jackson
Security Consultant
Aura Information Security
kirk@aurainfosec.com

Version: 0.1 - DRAFT

Date: 7 July 2011



Table of Contents

- 1 Introduction.....3**
 - 1.1 Reason for this paper3**
 - 1.2 Limitations3**
 - 1.3 Structure3**
- 2 File Upload Problems4**
- 3 Common File Types5**
 - 3.1 Image files5**
 - 3.2 PDF files5**
 - 3.3 Microsoft Office files6**
 - 3.4 HTML and plain-text files7**
 - 3.5 Active Web Content files7**
 - 3.6 Binary files8**
 - 3.7 Web server logic files.....8**
 - 3.8 Other files9**
- 4 Uploading Files to the Server 10**
 - 4.1 File selection and POST..... 10**
 - 4.2 Server receiving the file 11**
 - 4.3 Code receiving the file 12**
 - 4.4 Parse & Scan the file contents 13**
- 5 Providing Files for Download 15**
 - 5.1 Download method and domain 15**
 - 5.2 Mime types 16**
 - 5.3 Content-Disposition header 16**
 - 5.4 URL authorisation and format 17**
- 6 Conclusion 19**

1 INTRODUCTION

1.1 Reason for this paper

Many modern websites allow users to upload files for storage and later display – HR sites allow CVs to be uploaded, photo sites allow images to be shared.

In our work as security consultants, the team at Aura Information Security finds ourselves giving similar advice to each customer on how to protect their sites and users from malicious intent. This report attempts to provide a first draft at generalising that advice so that other customers and companies may benefit.

1.2 Limitations

We can provide no warranty or guarantee that the information we provide will protect your site, or even that it will make it harder to break into. In our experience, each web application is built with different technologies, by different teams of people and are hosted in different environments – necessarily our advice to customers requesting our consulting services will be deliberate and based on an evaluation of their own circumstances.

If you would like validation that you have followed this advice correctly, and that your site is protected against the attacks detailed, we would welcome the opportunity to work with you on a consulting or penetration testing engagement.

1.3 Structure

The first section talks about the problems you might encounter when building a site with file upload ability, and the attacks that may arise.

The second section discusses common file types that are supported by web applications, and the issues that may be involved with accepting those types.

The third section will cover “uploading” – putting files up onto the server via a file upload control.

The fourth section will cover “downloading” – allowing files to be retrieved from the server by a user.

2 FILE UPLOAD PROBLEMS

It's fair to say that the World Wide Web was built in simpler times, before a lot of the security issues surrounding publishing and viewing other user's content had been discovered.

Allowing file uploads to your website exposes your application, server and users to attack if not secured correctly. In some cases there's no known protection against an issue, so allowing a file upload may mean accepting a business risk.

File uploads can contribute to many of the issues in the OWASP Top 10¹, in particular Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). Uploaded files may allow javascript to run in the context of the website, stealing user's credentials, or POSTing data to the server with malicious intent.

Uploaded files may also affect the server – exhausting resources, or executing with server privileges; or when downloaded to a client machine, they may affect the user's computer itself.

As motivation, consider a few attacks that we have witnessed or developed as proof-of concept in our time as security consultants:

- The HR page on a public website allows a CV to be uploaded in Microsoft Word format. When viewed on the corporate network, a macro has access to the LAN.
- Illegal / immoral content is embedded in valid files, and stored on a company's website for download by other conspirators.
- Uploading a malformed image file causes a server process to crash with an out of memory exception.

3 COMMON FILE TYPES

The most common file types that web applications accept for upload are Image files (jpeg, GIF etc), PDF files (Adobe Acrobat) and Microsoft Office document formats (Word, Excel, PowerPoint).

Some file formats “by design” allow for active content such as Javascript to run within them, and other file formats are ‘containers’ that allow other types of content to be embedded inside them.

3.1 Image files

Image files have been shared on the internet ever since, well, the beginning.

There is a rich history of image parsing exploits, where the code within web browsers or applications such as Word will run an attackers code when a maliciously crafted image file is opened by the application. Most platforms now have fairly robust image handling libraries that have been hardened over time using code inspection, fuzzing and other secure development techniques.

Considerations:

- Processing images can be CPU and memory-intensive on your web server, especially if your image library converts a compressed image file into a height x width bitmap in memory.
- Images can contain additional metadata in extra fields, such as the EXIF fields in a jpeg file. Extra data may be stored inside a legitimate file.
- Some image file formats are read from the beginning of the file, while others read from the end – which means it may be possible to have a valid image file that is also a valid file in another format (see the GIFAR attackⁱⁱ for an example).

Image-specific advice:

As well as other advice in this document:

- Consider how to move image processing logic out of the Request / Response lifecycle, e.g. by queuing and processing on a back-end server.
- Consider loading images into a server object, and then saving out only the resolution and meta-data that you control, rather than allowing the original file to be downloaded.

3.2 PDF files

Adobe Acrobat introduced the PDF format in 1993, and subsequently the format has been extended and standardised – the documentation for the format is now in excess of 750 pages.ⁱⁱⁱ

Love it or hate it, the PDF format is the de-facto format for sharing print-ready documents on the internet, and is used in preference to the other contenders (Word, Postscript and XPS?).

PDF files are not just static text, they may contain media files (images, movies), Javascript, 3D artwork (e.g. CAD diagrams), audio and data entry fields. Each of these brings along their own rendering logic, some of which may not have had a solid testing!

The Javascript functionality in PDF files is the most obviously ‘active’. If a PDF is loaded within the browser window, it may be possible to invoke actions back on the source web server.

See this example of the “World’s smallest PDF” that contains javascript:^{iv}

```
%PDF-1.
trailer <</Root<<
/Pages <<>>
/OpenAction <<
/S /JavaScript
/JS (app.alert({cMsg: 'Javascript!'});)>>>>>>
```

(Note that it’s possible to make a smaller one 😊)

Considerations:

- All PDF files may contain active content.
- Many users browse the internet with old versions of Adobe Reader installed, and there are easily obtained exploits for these.

PDF-specific advice:

Receiving untrusted PDF content is a hard problem to tackle.

- Consider virus / malware scanning all uploaded PDFs to make sure that known vulnerabilities are not let through.
- Consider repeating the scan each time the PDFs are downloaded, or when new AV databases are released.
- Ensure your corporate users are running up-to-date and patched browsers and Adobe Acrobat versions.
- Consider browser sniffing to check users have an up-to-date version of their PDF software before serving files to them.

3.3 Microsoft Office files

The latest versions of Microsoft Office store their files as zipped containers of XML and other content (e.g. .docx, .xlsx and .pptx).

As Office applications support active content through the use of Macros, traditionally embedded macros within documents have given the most issues, although in the newer versions of Office files must be saved with a different extension to allow macros to run (.docm, .xlsm, .pptm).

Office documents allow other forms of content that may cause issues, such as embedded media files (potential parser issues) and urls to external location (potential phishing, XSRF attacks etc).

Being a zip file, it is also possible for an attacker to stash other content inside a document file without it being noticed. For example, we were able to round-trip a Microsoft Word document that contained a fully functional Silverlight application through Google Docs.^v

Considerations:

- Newer versions of Microsoft Office and the newer file formats give greater protection against malicious macro execution.
- Malicious or secret content may be stored inside documents that you host on your site, without you being aware.
- Microsoft Office Isolated Conversion Environment may help remove malicious content from legacy Office documents.^{vi}

Microsoft Office-specific advice:

- Ensure your employees run the latest version and patches for Microsoft Office products, and are updated as soon as possible.
- Only use the recent Office file formats.

3.4 HTML and plain-text files

Allowing HTML files to be uploaded and stored in your web application carries obvious and inherent risk – HTML files may contain Javascript or other active content, and if opened in a web browser will execute as if it was code you placed on the server yourself.

Plain-text files also carry the same risk for site users running certain Internet Explorer versions – due to a feature of the browser called “MIME sniffing” (see Section 5.2), and must be handled in a special way to try and protect those users from Cross-Site scripting attacks.

HTML-specific advice:

- Seriously consider not allowing upload of HTML files.

3.5 Active Web Content files

Various plugins exist that will execute within the web browser window. For example Adobe Flash, Microsoft Silverlight and Java Applets allow a full application to run within the confines of a browser.

Applet-style plugins typically allow their applications to be referenced by any HTML on the same site, or by pages on other sites as long as the MIME type is correct. The applet may have privileges to perform actions against your server by virtue of the same-origin-policy defined by each plugin.

Therefore allowing upload of Flash (.swf), Silverlight (.xap), Java applet (.jar) or other plugin file formats without the correct precautions being taken may allow a malicious third-party site to reference a malicious applet uploaded on your site and perform cross-site scripting or cross-domain request forgery style attacks.

Active Web Content-specific advice:

- Ensure your application does not allow applets to be uploaded and served back to users.

3.6 Binary files

With the exception of Office macros, the above file formats have mainly considered web content executing in the context of a web browser.

Allowing upload of application code, such as executables, dmgs, zip files containing applications or any other kinds of binary content that will execute when run on the user's computer is inherently high risk. The user's computer may be infected with a virus, malware may be installed, or any other nasty thing could happen.

Binary file-specific advice:

- Prevent applications or executables from being uploaded to your web server.
- Ensure any binary content on your web site is regularly virus scanned, and that any tampering will be noticed.

3.7 Web server logic files

Most web servers are configured to execute code when requests match the name of a file on the server's hard drive. For example, if /index.php is requested, then PHP code within that file will be executed.

If an attacker is able to upload server-side code into a location where it can execute, then they will be able to perform any action that the web server can – such as database or file system access, or perhaps escalating to root / Administrator.

An example of where this might occur is an /uploads directory where image files are uploaded – if the server allows .php files to be uploaded into that directory, the web server may allow them to execute when the url is entered.

Considerations:

- Most web servers make execute permission based on the file extension of the files on disk (e.g. .php or .aspx), or on the location (e.g. /cgi-bin).
- Some web servers (particularly older versions) are configured to allow execution of file types that you may not realise. E.g. your server may allow .shtml files to run without you realising.

Web server logic-specific advice:

- Don't allow user content to be uploaded into the web root.
- Put specific directives in place to prohibit execution of any user-supplied content.
- Check your configuration to ensure only intended file formats will execute.

3.8 Other files

No list of file formats would be complete without a disclaimer: There are lots of file formats supported by your server and user's browsers. Allowing upload of any file format without understanding the consequences is a risky endeavour and should be treated with care 😊

4 UPLOADING FILES TO THE SERVER

This section covers the duration in time between when a user selects a file on their computer to upload, and the web application stores the file for later display.

For now, we will only consider uploads using the standard HTML file input field, uploaded as a MIME multipart form.

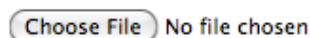
It should be noted that there are other ways for files to be uploaded to web applications, such as via Flash or Java applets, or even via non-HTTP mechanisms – some of these recommendations may apply to those forms of upload, and there may be other considerations to be aware of that aren't covered here.

4.1 File selection and POST

To allow a file to be uploaded a web page must have a file input tag inside a form that is multipart/formdata encoding type and that will POST to the server:

```
<form
  method="post"
  enctype="multipart/form-data"
  ... >
  <input type="file" name="myfile" ... />
</form>
```

The input control gets displayed in the browser similarly to this:



(Your browser may vary)

After the user selects a file and submits the form, the file contents are encoded and sent to the server:

```
POST http://site.com/ HTTP/1.1
Content-Type: multipart/form-data; boundary=-----1680728247
Content-Length: 19369

-----1680728247
Content-Disposition: form-data; name="myfile";
  filename="file.txt"
Content-Type: image/png

<file data goes here>

-----1680728247--
```

The browser separates each input value with a boundary field (normally longer than this example).

The name="myfile" data posted to the server refers to the original field name on the page. The filename field refers to the file on the user's disk, and is browser-dependant (some may post the full path to the file). The content-type is the browser's guess at the MIME type for the file.

All of the data posted to the web server should be treated as suspicious!
(This is a standard web security principle)

You cannot trust the file contents, the file name, the mime type or even the content-length header. Everything posted to the server is suspicious.

Considerations:

- If you wish to use the filename provided by the browser, validate it using a white-list of safe characters (e.g. A-z,0-9), and be sure to remove OS-specific characters, trailing whitespace and double-extensions.
- Don't trust the mime type sent in the content-type header – different browsers and operating systems will submit different values anyway.
- Don't trust that the file content matches that you'd expect from the filename or content-type, you will need to do your own file content checking.

Advice:

- Trust no-one ☺

4.2 Server receiving the file

Your web server will typically handle receiving the uploaded files, and will probably add uploaded files into an in-memory structure such as ASP.NET's Request.Files collection.

Before the file arrives at your code, there are a few things to consider:

Considerations:

- Large file uploads may swamp server resources – most web servers allow you to restrict the maximum amount of data that can be sent to the server in one request.
- Uploaded files may be stored temporarily on disk, and if large files are uploaded, they may exhaust disk space.
- Can your web server perform some content filtering for you (e.g. XSS detection)?
- A malicious user may upload the file very slowly, tying up server resources for the duration of the upload. This is a form of application-level denial of service attack.

Advice:

- Configure your web server to allow a sensible maximum file upload size.
- Configure temporary storage so that it is isolated from the system drive (e.g. on a secondary spindle / user-level quotas)
- Consider how to protect against slow uploads, perhaps by configuring a request timeout (e.g. Apache mod_reqtimeout).

4.3 Code receiving the file

Once the server has passed the file on to your code, then your processing can begin. Most applications attempt to write the file to the destination storage as quickly as possible, or process asynchronously using a queue, to avoid tying up server resources.

The following are some considerations if you are using the file system to store your uploaded files:

File System Considerations:

- Files may be web-accessible via the website when they shouldn't be.
- Provided filenames may trigger behaviour on your server – e.g. files named .php may be executed if placed within your website.

Advice:

- Store files outside of the web root, and separate fully-processed files from those that are still being processed.
- Restrict filenames to safe characters only (see Section 4.1).
- Be aware of file traversal issues, where a malicious user may be able to escape out of the intended directory.

If you are using a database to store your files, such as a relational SQL database, then the following considerations may apply:

Database Considerations:

- Different 'column types' in a given database may give differing performance or security characteristics. For example, in a relational database, some column types may truncate inserted data and could turn a valid file into an invalid one.
- Database growth may accelerate rapidly once binary data is stored in it. If an attacker is able to upload as many files as they want, they may fill up an important database volume.
- Traditional "INSERT INTO" syntax for database access may not perform well with your database engine. There may be alternate ways to insert or update binary data.
- Databases are often physically separate from web servers. Sending large files across a network boundary may require many in memory copies and may fill the network capacity.

Advice:

- Use an appropriate storage type for the binary data you are storing.
- Consider database engine mechanisms for restricting the impact of database growth on other functions. You might consider separating uploaded files from other database tables and logging information by writing them to a different spindle.
- Explore how to insert binary data to your database in the most efficient manner.

4.4 Parse & Scan the file contents

A malicious user can upload any content that they wish to your web application, so safely parsing it for well-formedness and scanning it for malware and viruses is important before you store it in your environment, and allow it to be downloaded.

Some file formats have established specifications, and you may be able to find a library or function that can test the uploaded file matches the structure expected. Other file formats are not so simple, and you may need to resort to magic number checking using file signatures.^{vii} If the magic number algorithm returns a MIME type, store that alongside the file, and don't rely on the file extension.

As discussed in Section 3, some file formats allow malicious content by design. If it is absolutely essential to allow these formats, you may need to build a parser that only allows the safe subset of the content.

Anti-virus software and malware scanners may provide some protection against files that exploit known vulnerabilities. Keeping the database up to date so that issues can be caught as soon as they are publicised is import, as is the ability to be able to re-scan collected content some time after it is uploaded, in case it was uploaded before the AV vendor released their detection.

If you must build your own parser, take care – especially if using regular expressions, which can perform very poorly under certain input. Building a parser is a project upon itself, and you'll want to make sure you "fuzz" your parser to make it robust under all input.

Considerations:

- Parsing and virus scanning may be CPU or memory intensive, and such software may not be built with a web environment in mind.
- Files may contain other content, perhaps in meta-data or appended to the start or end of the file (see Section 3).
- Building your own parser should be a last resort.

Advice:

- Implement a parser to ensure files are well-formed.

DRAFT - File Upload Considerations - DRAFT

- Implement virus and malware scanning of all uploaded files.
- Consider implement a background queue to process and scan files to restrict the chance of CPU / memory spikes.
- Do not support file formats that you cannot guarantee are safe.
- Take care when using regular expressions to parse input.

5 PROVIDING FILES FOR DOWNLOAD

This section assumes that you now have an uploaded file on your web server, and you wish to provide it to users to view / download.

The following information should be known about the file by this point:

- The MIME type
- The size of the file
- A safe filename to show the user
- That it is well-formed, contains no viruses or malware

The way that a web application serves files to the user can make it more secure for them to receive them. The aim is to protect the web application (e.g. from XSS), the user, and the user's browser and computer from malicious activity.

5.1 Download method and domain

To initiate a file download, the user's browser makes either a GET or a POST request to the server. Apart from the method of the request, the server response in both cases is the same.

Example:

The user requests to download the `http://example.com/Foo.pdf` from the web server:

```
GET /Foo.pdf HTTP/1.1
Host: example.com
Cookie: Adventure=XYZZY
```

The server responds with the file size, type and contents:

```
HTTP/1.1 200 OK
Content-Length: 140857
Content-Type: application/pdf
```

```
<file binary content comes here>
```

(Irrelevant headers have been removed)

If the file is hosted on a domain that uses cookies for authentication, then the cookies may be exposed to malicious content if it executes on the server or in the user's browser.

Also, consider the level of trust in the user's browser that the site has. Intranet sites typically have a higher level of trust and can perform more malicious actions if there is a security issue. By hosting user-uploaded files in the "Internet Zone", the

browser will have the maximum level of protection offered by the browsers segregation features.

Advice:

- Consider hosting user-generated content on a different domain name, so that cookies can never leak to malicious code, and cross-domain restrictions apply.
- Do not host user-generated files in the Intranet or other trusted zones.

5.2 Mime types

Allowing a file to be downloaded with the 'incorrect' MIME type can cause issues, such as allowing it to run in an HTML context and execute javascript, or allowing a file to run as a Java applet.

In Section 4.4 we discussed scanning the file and ensuring it matches the expected MIME type. It must be downloaded with the type that was discovered in that step.

Microsoft Internet Explorer browsers have a 'backward compatibility' feature that attempts to sniff the MIME type of a file and display it differently if it thinks the server issued the file with the wrong MIME type. This non-intuitive behaviour means that if you serve a file as text/plain, the web browser may 'sniff' the MIME type as text/html based on the presence of HTML tags, and allow it to run as HTML within the browser.

Starting with IE8, it is now possible to disable this sniffing behaviour, protecting users from having 'safe' content run as HTML. This can be done by setting the following header in the response:

```
X-Content-Type-Options: nosniff
```

This header does not affect the behaviour of non-IE browsers, so we recommend setting it in all cases when files are downloaded.

MIME-sniffing affects files served with text/plain, application/octet-stream, empty or null Content-Types.^{viii} If you intend to host files of these types, you should consider the possible impact of users using older versions of IE accessing them.

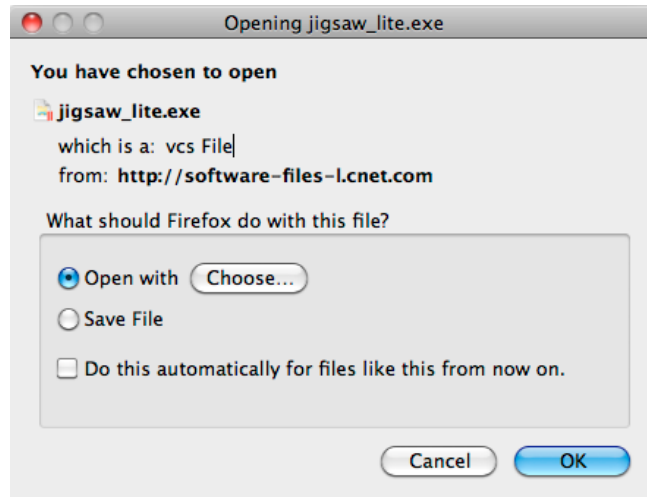
Advice:

- Ensure files are served with the correct Content-Type.
- Always serve files with the X-Content-Type-Options: nosniff header.
- Consider how to protect users running old versions of Internet Explorer.

5.3 Content-Disposition header

The Content-Disposition header can be sent by the server to force downloaded files to pop up the familiar Open / Save dialog:

```
HTTP/1.1 200 OK
Content-Length: 140857
Content-Disposition: attachment; filename=Foo.pdf
Content-Type: application/pdf
```



If the file is 'saved' to the user's desktop, then it will not open in the browser window with the context of the current user's login to that site – so XSS issues are moot. However, once the file is on the user's computer, it could still contain other malicious content that executes with their local privileges, so it's not a complete panacea.

If the user chooses to 'Open' the file, some browsers may still allow it to run within the context of the website, allowing the regular attacks to proceed. To protect against this, recent browsers now support an X-Download-Options header to disable the "Open" option in the dialog:

```
X-Download-Options: noopen
```

The Content-Disposition header also allows a filename to be set. It should be set to the safe filename generated earlier.

Advice:

- Serve files with the Content-Disposition: attachment header.
- Add the X-Download-Options: noopen header to responses.
- Ensure a safe filename is set by the server.

5.4 URL authorisation and format

The url to download files may be tampered by a malicious user, so application developers should ensure that permissions are checked before the user can

download a file – in case they are able to successfully guess the url for another user's file.

An attacker may also want to determine if a file is present or not by trying different urls and seeing if the 'unauthorised' response is different to the 'file not found' response. This information could be used by an attacker to mount a later attack.

If files are stored in a file system directory, care must be taken to isolate the code serving the files from file system traversal logic – in case an attacker can manipulate the url to download other files on the server.

Advice:

- Ensure correct authorisation checks are performed before serving files.
- Ensure that user's cannot guess other user's file urls.
- Ensure that directory listings are disabled and directory traversal not possible.

6 CONCLUSION

Our hope is that this document provided you with some items for consideration when building your own web application.

Security is a moving target, following any recommendations blindly without understanding them is risky, and new attacks are invented every week. We recommend a regular process of security review be implemented in your application lifecycle – preferably as early in the process as possible.

Hope this helps!

Kirk Jackson
Security Consultant
Aura Information Security

For corrections, suggestions or criticism, please email me at kirk@aurainfosec.com

ⁱ OWASP Top 10 project: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

ⁱⁱ Billy Rios – SUN Fixes GIFARs: <http://xs-sniper.com/blog/2008/12/17/sun-fixes-gifars/>

ⁱⁱ Billy Rios – SUN Fixes GIFARs: <http://xs-sniper.com/blog/2008/12/17/sun-fixes-gifars/>

ⁱⁱⁱ PDF Reference and Adobe Extensions to the PDF Specification:

http://www.adobe.com/devnet/pdf/pdf_reference.html

^{iv} Julia Wolf – That PDF thing: <http://blog.fireeye.com/research/2010/06/that-pdf-thing.html>

^v Kirk Jackson – There’s something shiny in that Word doc! <http://2010.kiwicon.org/the-con/talks/#e39>

^{vi} Microsoft Office Isolated Conversion Environment: <http://support.microsoft.com/kb/935865>

^{vii} Wikipedia – Magic number (programming): [http://en.wikipedia.org/wiki/Magic_number_\(programming\)](http://en.wikipedia.org/wiki/Magic_number_(programming))

^{viii} MIME Type Detection in Internet Explorer:

[http://msdn.microsoft.com/en-us/library/ms775147\(v=vs.85\).aspx#MIME_sniff_feature_control](http://msdn.microsoft.com/en-us/library/ms775147(v=vs.85).aspx#MIME_sniff_feature_control)