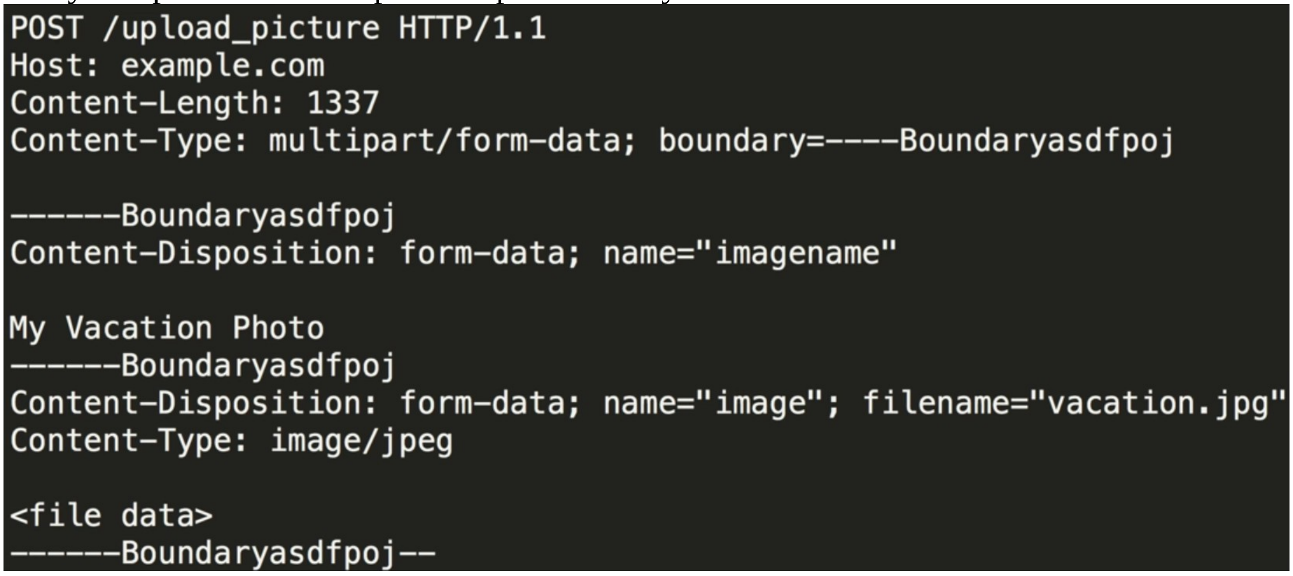
All this text wasn’t written by me (Tiago a.k.a TigaxMT).

All of it was transcribed by me from HackerOne video lessons.

All credits and thanks go to them.

File Upload

Study this print about of a picture upload and try to understand it



```
POST /upload_picture HTTP/1.1
Host: example.com
Content-Length: 1337
Content-Type: multipart/form-data; boundary=----Boundaryasdfpoj

------Boundaryasdfpoj
Content-Disposition: form-data; name="imagename"

My Vacation Photo
------Boundaryasdfpoj
Content-Disposition: form-data; name="image"; filename="vacation.jpg"
Content-Type: image/jpeg

<file data>
------Boundaryasdfpoj--
```

FILENAMES

The most obvious way to break this is via the filename. If you see an uploaded file retain the original filename – or a derivative thereof – chances are good that you can manipulate it. For instance, directory traversal works well here. What if you upload a file called ”../../test.php”?

MIME TYPES

Often, the MIME type sent by the browser on file uploads is stored in a database along with the uploaded filename and associated metadata.
If this is sent down with the file when it’s accessed, this can allow things like XSS. For instance, you could upload an HTML file disguised with an image filename, and send text/html as the MIME type.
Upon access, the browser will parse it like normal.

SNIFFING

When uploading images in particular, many sites will attempt to validate that the file is an actual image, then allow it to pass through with no modification.
The problem is that even if the file is a valid image, it can still contain malicious content, e.g. HTML

PNG files, for instance, can contain arbitrary ‘chunks’ that are completely valid and ignored by the file loader. This allows you to embed any HTML into a PNG while maintaining a completely proper file.

Depending on filename and MIME type sent by the server, the browser could decide to parse it purely as HTML, rather than an image, leading to XSS.

SEPARATE DOMAINS

In the vast majority of cases, files uploaded by users should be hosted on a separate domain.
The reason for this is that if you don't do that, same-origin policy comes into play, and it's possible for JavaScript running in the context of the domain to manipulate the site, get cookies, etc.
This does not solve any of the issues directly, but it does give you insurance if your other mitigations fail.

GENERATED FILENAMES

Filenames should never come from a user directly. When you receive a file upload, a filename should be generated randomly (or via hash of **file contents** + **timestamp**, for instance) with an extension based on MIME type or detected file type.
Note that the extensions should be whitelisted. You don't want people uploading HTML or other malicious content!

TYPE AND DISPOSITION

If the uploaded files are not being served directly from the filesystem, then the issue becomes: what MIME type and content disposition do we use?
The general rule is that MIME types should be whitelisted. If you want to allow types that are not on the list, the content disposition should be set to 'attachment'. That will force the browser to download the file, rather than display it.

IMAGE STRIPPING

If you're handling only images, then removing EXIF data from JPEGs, ancillary chunks from PNG files, and all other extraneous data is always a good idea.
The reason for this is that those are great hiding spots for HTML, and while there may not be a way to get the browser to load your image as HTML today, that can easily change tomorrow.
Removing this data is the only safe route.

GENERALIZED

The most important thing to remember when dealing with file uploads is: the user cannot be trusted.
Whenever possible, get rid of any non-essential data that the user provide you.

If you’re handling images, nothing but the actual image data matters. If you’re handling archive files, unpack them and repack them to ensure that nothing malicious is in the archive structure itself.