How does HTTP file upload work?

Asked 8 years, 8 months ago Active 3 months ago Viewed 580k times

549

226

When I submit a simple form like this with a file attached:

```
<form enctype="multipart/form-data" action="http://localhost:3000/upload?
upload_progress_id=12344" method="POST">
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
Choose a file to upload: <input name="uploadedfile" type="file" /><br />
<input type="submit" value="Upload File" />
</form>
```

How does it send the file internally? Is the file sent as part of the HTTP body as data? In the headers of this request, I don't see anything related to the name of the file.

I just would like the know the internal workings of the HTTP when sending a file.

http file-upload

edited Oct 13 '14 at 17:01
Mark Amery
97.4k ● 47 ● 331 ● 374

asked Dec 28 '11 at 18:34
0xSina
17.7k ● 27 ● 123 ● 241

I have not used a sniffer in a while but if you want to see what is being sent in your request (since it is to the server it is a request) sniff it. This question is too broad. SO is more for specific programming questions. – paparazzo Dec 28 '11 at 18:39

...as sniffers go, fiddler is my weapon of choice. You can even build up your own test requests to see how they post. – Phil Cooper Jan 31 '14 at 12:04

For those interested, also see "`MAX_FILE_SIZE` in PHP - what's the point" on stackoverflow.com/q/1381364/632951 – Pacerier Jul 24 '15 at 13:27

I find MAX_FILE_SIZE weird. as I can modify my html in chrome to 100000000 before posting it so it posts a better value. Either 1. have it in a cookie with a secure hash via salt so cookie if modified, server can validate and throw exception(like webpieces or playframework both do) or some sort of form validation that things haven't changed. @0xSina – Dean Hiller Apr 14 at 14:16

5 Answers

Active Oldest Votes

330

Let's take a look at what happens when you select a file and submit your form (I've truncated the headers for brevity):

```
POST /upload?upload_progress_id=12344 HTTP/1.1
Host: localhost:3000
Content-Length: 1325
```

```
Origin: http://localhost:3000
... other headers ...
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryePkpFF7tjBAqx29L

------WebKitFormBoundaryePkpFF7tjBAqx29L
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
------WebKitFormBoundaryePkpFF7tjBAqx29L
Content-Disposition: form-data; name="uploadedfile"; filename="hello.o"
Content-Type: application/x-object

... contents of file goes here ...
------WebKitFormBoundaryePkpFF7tjBAqx29L--
```

NOTE: each boundary string must be prefixed with an extra `--`, just like in the end of the last boundary string. The example above already includes this, but it can be easy to miss. See comment by @Andreas below.

Instead of URL encoding the form parameters, the form parameters (including the file data) are sent as sections in a multipart document in the body of the request.

In the example above, you can see the input `MAX_FILE_SIZE` with the value set in the form, as well as a section containing the file data. The file name is part of the `Content-Disposition` header.

The full details are here.

edited Jan 14 at 10:13
djc
10.6k ●4 ●35 ●47

answered Dec 28 '11 at 20:11
toddsundsted
5,421 ●2 ●18 ●12

Does this mean that port 80 (or the port serving http requests) is unusable during the time of the file transfer?. For e.g. if a huge file (about a GB) is being uploaded will the web server not be able to respond to any other requests during this time? – source.rar Apr 23 '14 at 16:39

7 @source.rar: No. Webservers are (almost?) always threaded so that they can handle concurrent connections. Essentially, the daemon process that's listening on port 80 immediately hands off the task of serving to another thread/process in order that it can return to listening for another connection; even if two incoming connections arrive at exactly the same moment, they'll just sit in the network buffer until the daemon is ready to read them. – eggyal Apr 30 '14 at 8:56

11 The threading explanation is a bit incorrect since there are high performance servers that are designed as single threaded and use a state machine to quickly take turns downloading packets of data from connections. Rather, in TCP/IP, port 80 is a listening port, not the port the data is transferred on. – slebetman Oct 13 '14 at 17:08

9 When an IP listening socket (port 80) receives a connection another socket is created on another port, usually with a random number above 1000. This socket is then connected to the remote socket leaving port 80 free to listen for new connections. – slebetman Oct 13 '14 at 17:10

11 @slebetman First of all, this is about HTTP. FTP active mode doesn't apply here. Second, listening socket doesn't get blocked on every connection. You can have as many connections to one port, as the other sides has ports to bind their own end to. – Slotos Nov 12 '14 at 20:58

3 @Slotos: I'm talking about HTTP. Not FTP active mode. I'm not talking about ports either, I'm talking about

sockets. – slebetman Nov 12 '14 at 22:55

34 Note that the boundary string that is passed as part of the Content-Type header field is 2 characters shorter than the boundary strings for the individual parts below. I've just spent an hour of trying to figure out why my uploader doesn't work because it's quite hard to notice that there are actually only 4 dashes in the first boundary string but 6 dashes in the other boundary strings. In other words: When using the boundary string to separate the individual form data, it has to be prefixed by two dashes: -- It's described in RFC1867 of course but I think it should be pointed out here as well – Andreas Jan 4 '15 at 17:40

That's a very nice historic document. Nice to see where it was conceived. On the elementary level, it looks like the input[type=file] lets the form to hook into a file on the client system, and then sends it as binary array on form submit. – Paceman Aug 12 '15 at 12:51

2 @Andreas comment is very, very important. I took me few hours to figure out how to upload a file from FoxPro :P until I figured out that this is the issue. Maybe this answer could be updated? – Deus777 Feb 21 '18 at 15:16

thanks, what i probably missed is how should create the `boundary` value for cases im sending a file in `formData` not using a browser? – Blue Bot Feb 18 at 11:52

288

> How does it send the file internally?

The format is called `multipart/form-data`, as asked at: What does enctype='multipart/form-data' mean?

I'm going to:

- add some more HTML5 references
- explain **why** he is right with a form submit example

HTML5 references

There are three possibilities for `enctype`:

- `x-www-urlencoded`
- `multipart/form-data` (spec points to RFC2388)
- `text-plain`. This is "not reliably interpretable by computer", so it should never be used in production, and we will not look further into it.

How to generate the examples

Once you see an example of each method, it becomes obvious how they work, and when you should use each one.

You can produce examples using:

- `nc -l` or an ECHO server: HTTP test server accepting GET/POST requests

- an user agent like a browser or cURL

Save the form to a minimal `.html` file:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8"/>
  <title>upload</title>
</head>
<body>
  <form action="http://localhost:8000" method="post" enctype="multipart/form-data">
  <p><input type="text" name="text1" value="text default">
  <p><input type="text" name="text2" value="a&#x03C9;b">
  <p><input type="file" name="file1">
  <p><input type="file" name="file2">
  <p><input type="file" name="file3">
  <p><button type="submit">Submit</button>
</form>
</body>
</html>
```

We set the default text value to `a&#x03C9;b`, which means `aωb` because `ω` is `U+03C9`, which are the bytes `61 CF 89 62` in UTF-8.

Create files to upload:

```
echo 'Content of a.txt.' > a.txt

echo '<!DOCTYPE html><title>Content of a.html.</title>' > a.html

# Binary file containing 4 bytes: 'a', 1, 2 and 'b'.
printf 'a\xCF\x89b' > binary
```

Run our little echo server:

```
while true; do printf '' | nc -l 8000 localhost; done
```

Open the HTML on your browser, select the files and click on submit and check the terminal.

`nc` prints the request received.

Tested on: Ubuntu 14.04.3, `nc` BSD 1.105, Firefox 40.

multipart/form-data

Firefox sent:

```
POST / HTTP/1.1
[[ Less interesting headers ... ]]
Content-Type: multipart/form-data; boundary=---------------------------
-735323031399963166993862150
Content-Length: 834

-----------------------------735323031399963166993862150
Content-Disposition: form-data; name="text1"

text default
-----------------------------735323031399963166993862150
Content-Disposition: form-data; name="text2"

aωb
-----------------------------735323031399963166993862150
Content-Disposition: form-data; name="file1"; filename="a.txt"
Content-Type: text/plain

Content of a.txt.

-----------------------------735323031399963166993862150
Content-Disposition: form-data; name="file2"; filename="a.html"
Content-Type: text/html

<!DOCTYPE html><title>Content of a.html.</title>

-----------------------------735323031399963166993862150
Content-Disposition: form-data; name="file3"; filename="binary"
Content-Type: application/octet-stream

aωb
-----------------------------735323031399963166993862150--
```

For the binary file and text field, the bytes `61 CF 89 62` (`aωb` in UTF-8) are sent literally. You could verify that with `nc -l localhost 8000 | hd`, which says that the bytes:

```
61 CF 89 62
```

were sent (`61` == 'a' and `62` == 'b').

Therefore it is clear that:

- `Content-Type: multipart/form-data; boundary=---------------------------735323031399963166993862150` sets the content type to `multipart/form-data` and says that the fields are separated by the given `boundary` string.

  But note that the:

  ```
  boundary=---------------------------735323031399963166993862150
  ```

  has two less dadhes `--` than the actual barrier

  ```
  -----------------------------735323031399963166993862150
  ```

This is because the standard requires the boundary to start with two dashes `--`. The other dashes appear to be just how Firefox chose to implement the arbitrary boundary. RFC 7578 clearly mentions that those two leading dashes `--` are required:

> 4.1. "Boundary" Parameter of multipart/form-data
>
> As with other multipart types, the parts are delimited with a boundary delimiter, constructed using CRLF, "--", and the value of the "boundary" parameter.

- every field gets some sub headers before its data: `Content-Disposition: form-data;`, the field `name`, the `filename`, followed by the data.

  The server reads the data until the next boundary string. The browser must choose a boundary that will not appear in any of the fields, so this is why the boundary may vary between requests.

  Because we have the unique boundary, no encoding of the data is necessary: binary data is sent as is.

  TODO: what is the optimal boundary size (`log(N)` I bet), and name / running time of the algorithm that finds it? Asked at: https://cs.stackexchange.com/questions/39687/find-the-shortest-sequence-that-is-not-a-sub-sequence-of-a-set-of-sequences

- `Content-Type` is automatically determined by the browser.

  How it is determined exactly was asked at: How is mime type of an uploaded file determined by browser?

application/x-www-form-urlencoded

Now change the `enctype` to `application/x-www-form-urlencoded`, reload the browser, and resubmit.

Firefox sent:

```
POST / HTTP/1.1
[[ Less interesting headers ... ]]
Content-Type: application/x-www-form-urlencoded
Content-Length: 51

text1=text+default&text2=a%CF%89b&file1=a.txt&file2=a.html&file3=binary
```

Clearly the file data was not sent, only the basenames. So this cannot be used for files.

As for the text field, we see that usual printable characters like `a` and `b` were sent in one byte, while non-printable ones like `0xCF` and `0x89` took up **3 bytes** each: `%CF%89`!

Comparison

File uploads often contain lots of non-printable characters (e.g. images), while text forms almost never do.

From the examples we have seen that:

- `multipart/form-data`: adds a few bytes of boundary overhead to the message, and must spend some time calculating it, but sends each byte in one byte.
- `application/x-www-form-urlencoded`: has a single byte boundary per field (`&`), but adds a *linear* overhead factor of **3x** for every non-printable character.

Therefore, even if we could send files with `application/x-www-form-urlencoded`, we wouldn't want to, because it is so inefficient.

But for printable characters found in text fields, it does not matter and generates less overhead, so we just use it.

edited May 7 at 10:40

answered Nov 6 '14 at 23:11
Ciro Santilli 郝海东冠状病六四事件法轮功
219k ●52 ●841 ●664

1 How would you add a binary attachment? (i.e. a small image) - I can see changing the values for the `Content-Disposition` and `Content-Type` attributes but how to handle the 'content'? – blurfus Feb 5 '15 at 20:54

3 @ianbeks The browser does it automatically before sending the request. I don't know which heuristics it uses, but most likely the file extension is amongst them. This may answer the question: stackoverflow.com/questions/1201945/… – Ciro Santilli 郝海东冠状病六四事件法轮功 Feb 17 '15 at 13:34

3 @CiroSantilli六四事件法轮功纳米比亚威视 I think this answer is much better than the chosen one. But please remove the irrelevant content from your profile. It is against the spirit of SO. – smwikipedia Jul 6 '15 at 7:07

2 @smwikipedia thanks for the rfc quote and for liking this answer! About username: to me, the spirit of SO is that everyone should have the best information at all times. ~~ Let's keep this discussion to twitter or meta. Peace. – Ciro Santilli 郝海东冠状病六四事件法轮功 Jul 6 '15 at 8:02

1 @KumarHarsh not enough detail to answer I think. Please open a new super detailed questions. – Ciro Santilli 郝海东冠状病六四事件法轮功 Apr 19 '17 at 12:18

64

Send file as binary content (upload without form or FormData)

In the given answers/examples the file is (most likely) uploaded with a HTML form or using the FormData API. The file is only a part of the data sent in the request, hence the `multipart/form-data` `Content-Type` header.

If you want to send the file as the only content then you can directly add it as the request body and you set the `Content-Type` header to the MIME type of the file you are sending. The file name can be added in the `Content-Disposition` header. You can upload like this:

```
var xmlHttpRequest = new XMLHttpRequest();

var file = ...file handle...
var fileName = ...file name...
var target = ...target...
var mimeType = ...mime type...

xmlHttpRequest.open('POST', target, true);
xmlHttpRequest.setRequestHeader('Content-Type', mimeType);
xmlHttpRequest.setRequestHeader('Content-Disposition', 'attachment; filename="' + fileName +
'"');
xmlHttpRequest.send(file);
```

If you don't (want to) use forms and you are only interested in uploading one single file this is the easiest way to include your file in the request.

edited Feb 11 '16 at 8:00

answered Jan 28 '15 at 13:04
Wilt
29.5k ● 10 ● 112 ● 160

How do you configure a server side service for this with Asp.Net 4.0? Will it handle multiple input parameters as well, such as userId, path, captionText etc? – Asle G Jan 30 '15 at 9:46

1 @AsleG Nope, it is only for sending a single file as the content of your request. I am not an Asp.Net expert, but you should simply pull the content (a blob) out of the request and save it to a file using the `Content-Type` from the header. – Wilt Jan 30 '15 at 9:55

@AsleG Maybe this link can help – Wilt Jan 30 '15 at 11:57

@wilt If I don't use form, but I want to use formdata API, can I do it that way? – angry kiwi Sep 15 '15 at 8:18

1 @AnkitKhettry Sounds like it is uploaded with a form or by using the form API. These 'weird strings' you refer to are the form boundaries normally used for separating the form data into parts on the server. – Wilt Jul 1 '16 at 15:16

9

I have this sample Java Code:

```
import java.io.*;
import java.net.*;
import java.nio.charset.StandardCharsets;

public class TestClass {
    public static void main(String[] args) throws IOException {
        ServerSocket socket = new ServerSocket(8081);
        Socket accept = socket.accept();
        InputStream inputStream = accept.getInputStream();

        InputStreamReader inputStreamReader = new InputStreamReader(inputStream,
StandardCharsets.UTF_8);
        char readChar;
        while ((readChar = (char) inputStreamReader.read()) != -1) {
            System.out.print(readChar);
        }

        inputStream.close();
        accept.close();
        System.exit(1);
    }
}
```

and I have this test.html file:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>File Upload!</title>
</head>
<body>
<form method="post" action="http://localhost:8081" enctype="multipart/form-data">
    <input type="file" name="file" id="file">
    <input type="submit">
</form>
</body>
</html>
```

and finally the file I will be using for testing purposes, named **a.dat** has the following content:

```
0x39 0x69 0x65
```

if you interpret the bytes above as ASCII or UTF-8 characters, they will actually will be representing:

```
9ie
```

So let 's run our Java Code, open up **test.html** in our favorite browser, upload `a.dat` and submit the form and see what our server receives:

```
POST / HTTP/1.1
Host: localhost:8081
Connection: keep-alive
Content-Length: 196
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: null
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_5) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/48.0.2564.97 Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary06f6g54NVbSieT6y
DNT: 1
Accept-Encoding: gzip, deflate
Accept-Language: en,en-US;q=0.8,tr;q=0.6
Cookie: JSESSIONID=27D0A0637A0449CF65B3CB20F40048AF
```

```
------WebKitFormBoundary06f6g54NVbSieT6y
Content-Disposition: form-data; name="file"; filename="a.dat"
Content-Type: application/octet-stream

9ie
------WebKitFormBoundary06f6g54NVbSieT6y--
```

Well I am not surprised to see the characters **9ie** because we told Java to print them treating them as UTF-8 characters. You may as well choose to read them as raw bytes..

```
Cookie: JSESSIONID=27D0A0637A0449CF65B3CB20F40048AF
```

is actually the last HTTP Header here. After that comes the HTTP Body, where meta and contents of the file we uploaded actually can be seen.

edited May 14 at 12:48

answered Jan 29 '16 at 18:44
Koray Tugay
19.2k ● 34 ● 144 ● 260

6

> An HTTP message may have a body of data sent after the header lines. In a response, this is where the requested resource is returned to the client (the most common use of the message body), or perhaps explanatory text if there's an error. In a request, this is where user-entered data or uploaded files are sent to the server.

http://www.tutorialspoint.com/http/http_messages.htm

answered Dec 28 '11 at 18:42
flagg19
1,662 ● 2 ● 22 ● 27

Highly active question. Earn 10 reputation in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.