Multipart Parsing and File Uploads

HTML Forms - POST

- Last time we saw forms:
  - Submit the content of the form in the body of the POST request

```
<form action="/form-path" method="post">
    <label>Enter your name: <br/>
        <input type="text" name="commenter"><br/>
    </label>

    <label>Comment: <br/>
        <input type="text" name="comment"><br/>
    </label>

    <input type="submit" value="Submit">
</form>
```

```
POST /form-path HTTP/1.1
Content-Length: 27
Content-Type: application/x-www-form-urlencoded

commenter=Jesse&comment=Good+morning%21
```

HTML Forms - POST

- You can specify the encoding type for form submissions
- Default is url encoded (Special characters are % encoded)

```
POST /form-path HTTP/1.1
Content-Length: 27
Content-Type: application/x-www-form-urlencoded

commenter=Jesse&comment=Good+morning%21
```

```
<form action="/form-path" method="post" enctype="application/x-www-form-urlencoded">
    <label>Enter your name: <br/>
        <input type="text" name="commenter"><br/>
    </label>

    <label>Comment: <br/>
        <input type="text" name="comment"><br/>
    </label>

    <input type="submit" value="Submit">
</form>
```

HTML Forms - POST

- url encoding cannot be used to upload files from a form!
  - The browser will only send the filename, not the contents of the file
- We need something else for file uploads

```
POST /form-path HTTP/1.1
Content-Length: 27
Content-Type: application/x-www-form-urlencoded

commenter=Jesse&comment=Good+morning%21
```

```
<form action="/form-path" method="post" enctype="application/x-www-form-urlencoded">
    <label>Enter your name: <br/>
        <input type="text" name="commenter"><br/>
    </label>

    <label>Comment: <br/>
        <input type="text" name="comment"><br/>
    </label>

    <input type="submit" value="Submit">
</form>
```

HTML Forms - POST

- Specify multipart encoding to receive each input separately in the body of the request
- With multipart encoding, the browser will send the contents of files

```
<form action="/form-path" method="post" enctype="multipart/form-data">
    <label>Enter your name: <br/>
        <input type="text" name="commenter"><br/>
    </label>

    <label>Comment: <br/>
        <input type="text" name="comment"><br/>
    </label>

    <input type="submit" value="Submit">
</form>
```

HTML Forms - POST

- Our server receives a multipart form request in this format

```
POST /form-path HTTP/1.1
Content-Length: 252
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryfkz9sCA6fR3CAHN4

------WebKitFormBoundaryfkz9sCA6fR3CAHN4
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundaryfkz9sCA6fR3CAHN4
Content-Disposition: form-data; name="comment"

Good morning!
------WebKitFormBoundaryfkz9sCA6fR3CAHN4--
```

HTML Forms - POST

- Content-Type specifies a string that separates each input
- Each input has its own headers
- Great for submitting different types of data in the same form
  - Required for file uploads

```
POST /form-path HTTP/1.1
Content-Length: 252
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryfkz9sCA6fR3CAHN4

------WebKitFormBoundaryfkz9sCA6fR3CAHN4
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundaryfkz9sCA6fR3CAHN4
Content-Disposition: form-data; name="comment"

Good morning!
------WebKitFormBoundaryfkz9sCA6fR3CAHN4--
```

File Uploads

- We must use multipart/form-data to upload files
  - If not, browser only sends the filename
- Add an input with type "file"
  - The browser does the rest
  - Users will be able to choose a file to send

```
<form action="/form-path" method="post" enctype="multipart/form-data">
    <label>Enter your name: <br/>
        <input type="text" name="commenter"><br/>
    </label>

    <label>File: <br/>
        <input type="file" name="upload"><br/>
    </label>

    <input type="submit" value="Submit">
</form>
```

File Uploads

- When our server receives the file, it will appear in one of the parts of the multi-part POST request
- The content type will tell us the type of file
- The body of the part will contain all the bytes of that file
  - Can write these bytes to a new file on our server to save that file

```
------WebKitFormBoundarygVWEOc5JIyJ1qthO
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarygVWEOc5JIyJ1qthO
Content-Disposition: form-data; name="upload"; filename="discord2.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarygVWEOc5JIyJ1qthO--
```

File Uploads

- When receiving the bytes of a binary file, do not apply any encodings
  - When we received bytes representing text, we decoded it by interpreting the bytes as UTF-8 encoded text
  - When receiving files, we are often interested in the raw bytes (Ex. Images, videos)
  - The files will be encoded with algorithms other than UTF-8 (Ex. png, jpeg, mp4)
- Attempting to treat a binary file as a UTF-8 String will corrupt the data!

Parsing Multipart

- Your goal is to parse HTTP POST requests in this format
  - Without corrupting the image
- Let's walk through the steps you'll need to take

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Identify the request using the method and path
- This is a post request request for form-path
- Based on our HTML, we know to treat this as a form submission

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Parse the headers and read the length of the content
- This will be the number of bytes that need to be read from the body
- Follows the same protocol as your HTTP responses

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Refer to your content length and read that many bytes from the body of the request
- Important: Do not attempt to parse anything until you've read this many bytes from the body (More details in the buffering lecture)

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Read the type of the content to get the boundary
- Boundary is specified in addition to the MIME type
  - Recall that we used the same format to specify charset=utf-8

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- The body will consist of multiple parts, each separated by the boundary
- The browser will guarantee that the boundary is not contained in any of the data being sent
- This example has 2 parts

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- The parts are separated by the boundary with two leading dash characters "--"
- Each boundary must be at the beginning of a line
- The full boundary is CRLF + "--" + <boundary>
  - Except the first one which is already preceded by a CRLF from the blank line separating the headers and body

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- The end of the last boundary is marked by the full boundary plus two trailing "-" characters
- The last boundary is CRLF + "--" + <boundary> + "--"
- Can have a trailing CRLF after the last boundary that should be ignored

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Each part of the request will follow a similar format to HTTP requests:
  - Any number of headers
  - One blank line
  - The content of the part

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- The headers should include a Content-Disposition
  - Provides the name of the part in quotes
  - Name matches the name from your form
  - For files, the original filename is provided in quotes

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Content-Type is optional for parts
- If excluded, the default MIME type is text/plain

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Content-Length is not included in the parts
- The purpose of Content-Length is to ensure we've received the full body before parsing
- Already read the full body using the HTTP Content-Length

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- When reading the bytes of a [non-text] file
  - Never, never, never encode the bytes as a string!
- But wait.. how do I parse all this stuff without making it a String?..

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Parse in bytes!
- When parsing bytes that contain non-text data:
  - We'll use byte-parsing, not String-parsing

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- You receive the request as an array of bytes
- Scan this array for the bytes you're looking for
- Create sub-arrays to extract data

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Example: To extract the headers you would find the first instance of the "\r\n\r\n" String and read everything before it
- **Encode** "\r\n\r\n" into bytes and search the byte array for this sequence of bytes, then create a new array with everything before that sequence

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- When you have a sub-array containing only the headers
  - Decode using ASCII/UTF-8 and parse the headers

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Use a similar approach with the boundary
- Encode the boundary into bytes
- Look for that sequence of bytes in the body

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Repeat for each part
- Encode "\r\n\r\n" into bytes, scan the part for this sequence of bytes, read the headers

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- If the headers for a part specify a non-text encoding
  - Handle the body of that part as raw bytes
  - The bytes are never decoded using a text encoding

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

Parsing Multipart

- Once you read the raw bytes of the file/image
  - Save these bytes in a file
  - Serve these files for users to enjoy

```
POST /form-path HTTP/1.1
Content-Length: 9937
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarycriD3u6M0UuPR1ia

------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="commenter"

Jesse
------WebKitFormBoundarycriD3u6M0UuPR1ia
Content-Disposition: form-data; name="upload"; filename="discord.png"
Content-Type: image/png

<bytes_of_the_file>
------WebKitFormBoundarycriD3u6M0UuPR1ia--
```

File Uploads

- To save a file on your server:

- Save the file to disk
  - Bad practice to store files in a database
  - Generate a naming convention for your saved files
    - Do not use the user supplied filename

- Store the filename in your database
  - When you pull a record from the database, it will tell you which filename to use and that file can be read from disk

Image Uploads

- To host user supplied images on your site:

- Pull the filename from the database and use this as the src of the img element in your HTML (Using HTML templates)
- The browser will read the src attribute and send a request for the file path
- Host the bytes of the file at that path

We Now Support File Sharing!

Hosting Files

- We can make all uploaded files available to our users
- We create a url scheme to accomplish this
  - Ex. GET /public/image/cool-picture.png
  - Ex. GET /public/image?filename=cool-picture.png
- Parse the request to find which image is being requested
  - Open that file and send the bytes
  - Can send a 404 if the file doesn't exist on the server

Hosting Files - Security

- You'll have code that effectively does this:
  - Receive request for /public/image/<filename>
  - Parse the path to extract <filename>
  - Read the bytes of the file
  - Send those bytes to the user in your HTTP response

Hosting Files - Security

- You'll have code that effectively does this:
  - Receive request for /public/image/<filename>
  - Parse the path to extract <filename>
  - Read the bytes of the file
  - Send those bytes to the user in your HTTP response

- ... and someone makes the following request:
  - GET /public/image/~/.ssh/id_rsa

Hosting Files - Security

- GET /image/~/.ssh/id_rsa
  - **This attacker requested your private encryption key!**
- **We cannot allow users to request arbitrary files from the server**
- Add logic to ensure users can't access files outside the public directories
  - Removing all "/" characters will suffice for the HW

Hosting Files - Security

- GET /image/~/.ssh/id_rsa
  - **This attacker requested your private encryption key!**

- Strongest defense:
  - Maintain a list of all valid files that can be requested
  - return a 400-level response if any other file is requested

Hosting Files - Security

- Testing for this vulnerability

```
curl --path-as-is http://localhost:8080/public/image/../../server.py
```

- Following the structure of the HW server, this request with ask the server for server.py
- Your server must not respect this request
  - Sending your server code alone is a security issue, but this is also a proof of concept that they can request arbitrary files
- Uses curl in the command line
- The browser and curl will simplify paths by default and chance the path to "/server.py"
- --path-as-is prevents the path simplification

Hosting Files - Security

- Don't forget to disable MIME type sniffing
- At this point an attacker can upload Javascript instead of an image
  - Don't let the browser sniff the JS MIME type and run the attack script