IERG 4130 – 2026 Fall

香港中文大學
CUHK

Introduction to the Web

Yajin ZHOU

https://yajin.org

Credits: the slides are from CS161 of UC Berkeley

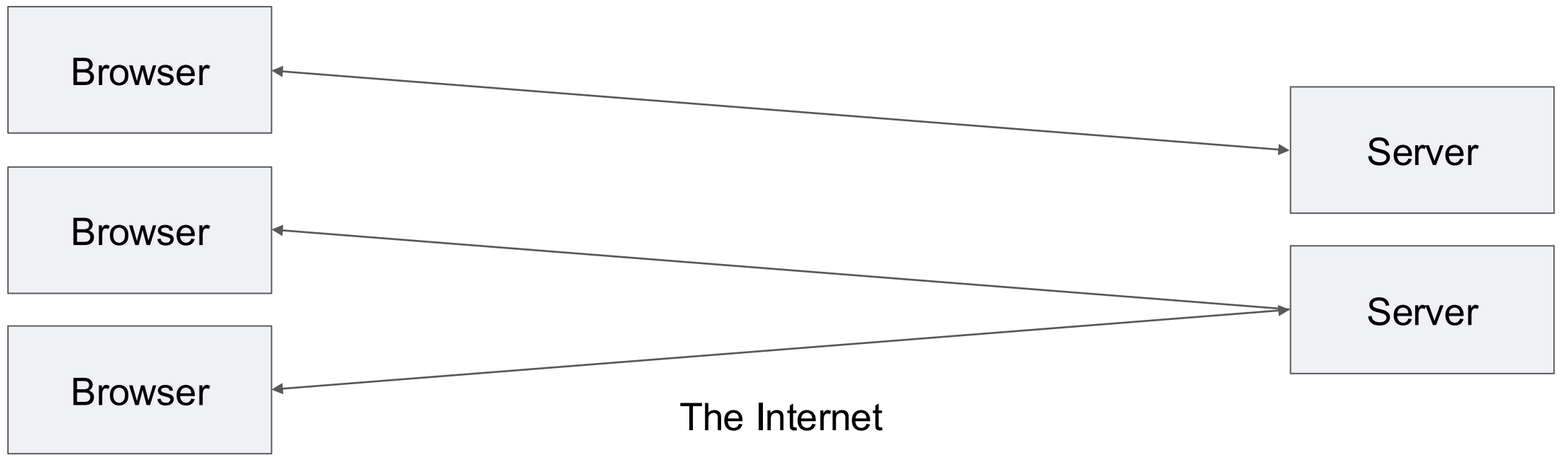
Summary



- **What's the web?**
- URLs
- HTTP
- Parts of a Webpage
  - HTML
  - CSS
  - JavaScript
- Security on the Web
- Same-Origin Policy

What's the Web?

- **World Wide Web ("Web"): A collection of data and services**
  - Data and services are provided by **web servers**
  - Data and services are accessed using **web browsers** (e.g. Chrome, Firefox)
- **The web is not the internet**
  - The Internet describes *how* data is transported between servers and browsers
  - We will study the internet later in the networking unit.



Today: Elements of the Web



- **URLs: Uniquely identify a piece of data on the web**
- **HTTP: The standard for how web browsers communicate with web servers**
- **Data on a web page can contain**
  - **HTML:** A markup language for creating web pages
  - **CSS:** A style sheet language for defining the appearance of web pages
  - **JavaScript:** A programming language for running code in the web browser

Summary



- What's the web?
- **URLs**
- HTTP
- Parts of a Webpage
  - HTML
  - CSS
  - JavaScript
- Security on the Web
- Same-Origin Policy

URLs

- **URL (Uniform Resource Locator):** A string that uniquely identifies one piece of data on the web
  - A type of URI (Uniform Resource Identifier)
- **Divided into parts**
  - Scheme/Protocol
  - Domain
  - Location (Username / Port)
  - Path
  - Query
  - Fragment / Anchor

Parts of a URL: Scheme / Protocol



- Located just before the double slashes
- Defines how to retrieve the data over the Internet (which protocol to use)
- Typically, http (unencrypted) or https (secure, encrypted)

https://yajin.org/teaching/IERG4130/2026/index.html

Parts of a URL: Domain



- **Located after the double slashes and before the next single slash**
- **Defines which web server to contact**
  - Recall: The web has many web servers. The domain specifies which one we're looking for.
- **Written as several phrases separated by dots**

https://yajin.org/teaching/IERG4130/2026/index.html

Parts of a URL: Location



- **The domain with some additional information**
  - **Username: evanbot@cs161.org**
    - Identifies a specific user on the web server
    - Rarely seen
  - **Port: toon.cs161.org:4000**
    - Identifies one specific application on the web server
    - If we don't include one, the default is usually based on the protocol
      - Example: http defaults to **port 80**
      - Example: https defaults to **port 443**
    - We will revisit ports in the networking unit
  - Domain is required, but additional information (username/port) is optional.

Parts of a URL: Path



- **Located after the first single slash**
- **Defines which file on the web server to fetch**
  - Think of the web server as having its own filesystem
  - The path represents a file on the web sever's filesystem
- **Examples:**
  - **https://yajin.org/teaching/IERG4130/2026/index.html**
    - Look in the **teaching/IERG4130/2026** folder for index.html
  - **https://yajin.org/teaching**
    - Return the teaching directory – not allowed (per my server's configuration)

Parts of a URL: Query



- **Providing a query is optional**
- **Located after a question mark**
- **Supplies arguments to the web server for processing**
  - Think of the web server as offering a function at a given path
  - To access this function, a user makes a request to the path, with some arguments in the query
  - The web server runs the function with the user's arguments and returns the result to the user
- **Arguments are supplied as name=value pairs**
- **Arguments are separated by ampersands (&)**

  - https://www.google.com/search?q=cuhk&sclient=gws-wiz-serp

Parts of a URL: Fragment / Anchor



- **Providing a fragment is optional**
- **Located after a hash sign (#)**
- **Not sent to the web server! Only used by the web browser**
  - Common usage: Tells the web browser to scroll to a part of a webpage
  - Usage: Supplies content to code in the web browser (JavaScript) without sending the content to the server

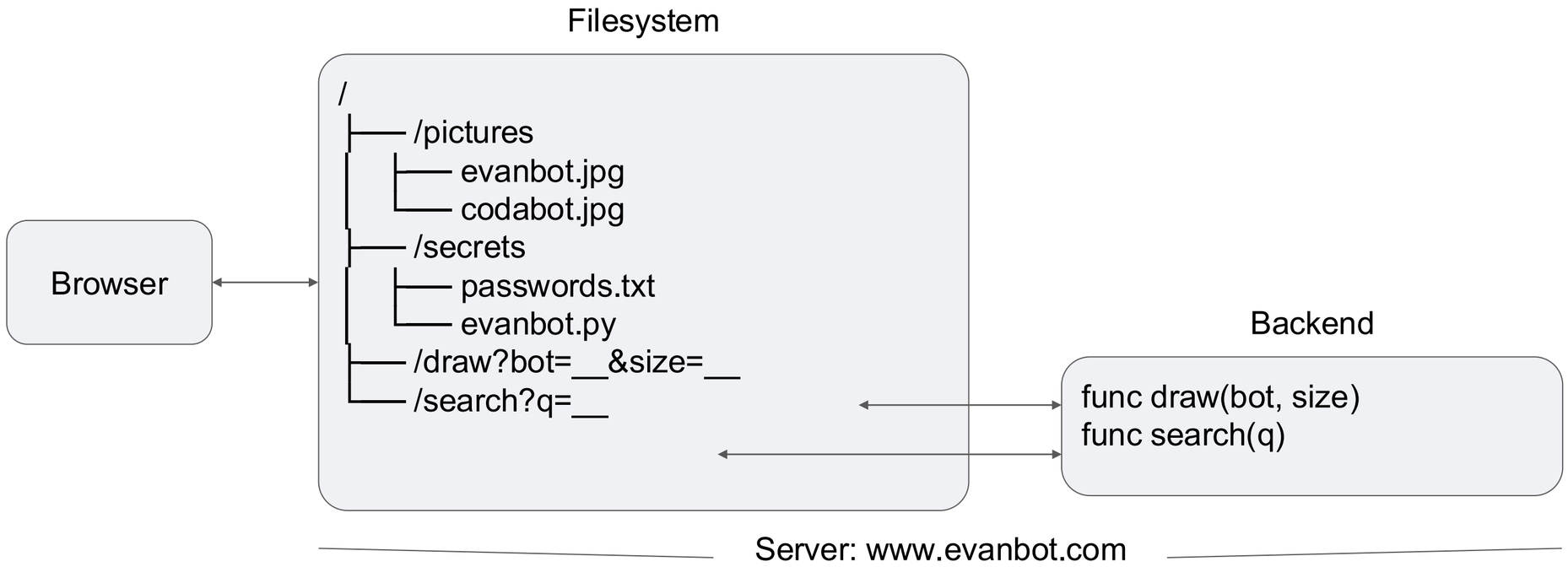
https://yajin.org/#publications

URL Escaping

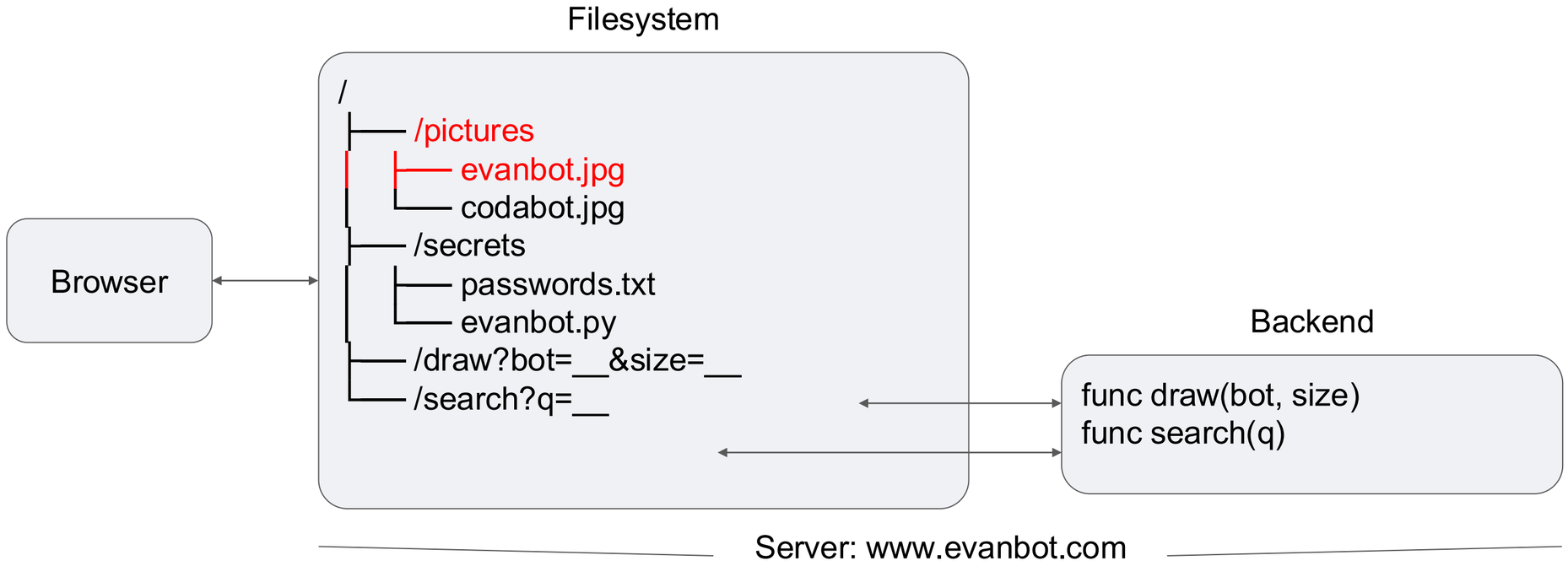


- **URLs contain printable, human-readable characters (ASCII)**
  - What if we want to include non-printable characters in the URL?
- **Recall: URLs have special characters (?, #, /)**
  - What if we want to use a special character in the URL?
- **Solution: URL encoding**
  - Notation: Percent sign (**%**) followed by the hexadecimal value of the character
  - Example: %20 = **' '** (spacebar)
  - Example: %35 = **'#'** (hash sign)
  - Example: %50 = **'2'** (printable characters can also be encoded!)
- **Security issues: Makes scanning for malicious URLs harder**
  - Suppose you want to block all requests to the path **/etc/passwd**
  - What if an attacker makes a request to **%2F%65%74%63%2F%70%61%73%73%77%64**
  - Known as **obfuscation**

A Simplified View of the Web



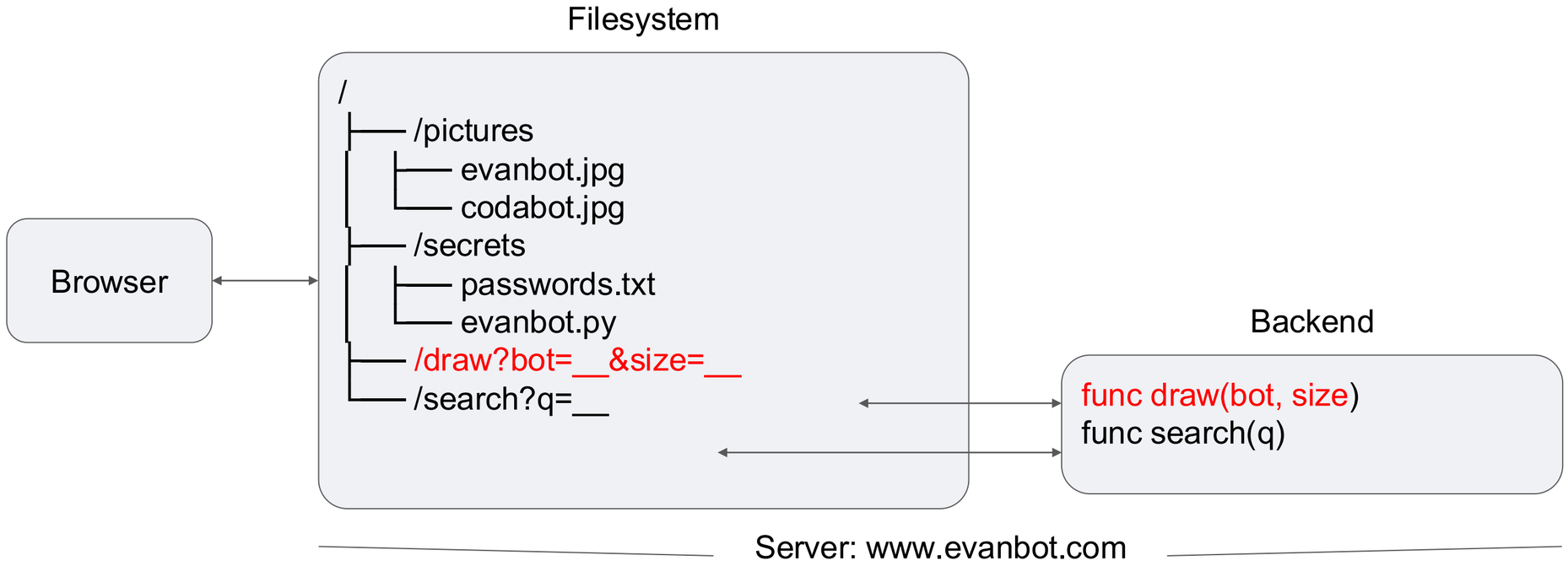
A Simplified View of the Web — File Query



The browser can request a file from the server with a URL:

https://evanbot.com/pictures/evanbot.jpg

A Simplified View of the Web — Compute Query



The browser can request some computation from the server:

https://evanbot.com/draw?bot=evan&size=large

Summary



- What's the web?
- URLs
- **HTTP**
- Parts of a Webpage
  - HTML
  - CSS
  - JavaScript
- Security on the Web
- Same-Origin Policy

HTTP



- **HTTP (Hypertext Transfer Protocol)**: A protocol used to request and retrieve data from a web server
- **HTTPS: A secure version of HTTP**
  - Uses cryptography to secure data
  - We'll see HTTPS later in the networking unit
- **HTTP is a request-response model**
  - The web browser sends a **request** to the web server
  - The web server processes the request and sends a **response** to the web browser

Parts of an HTTP Request



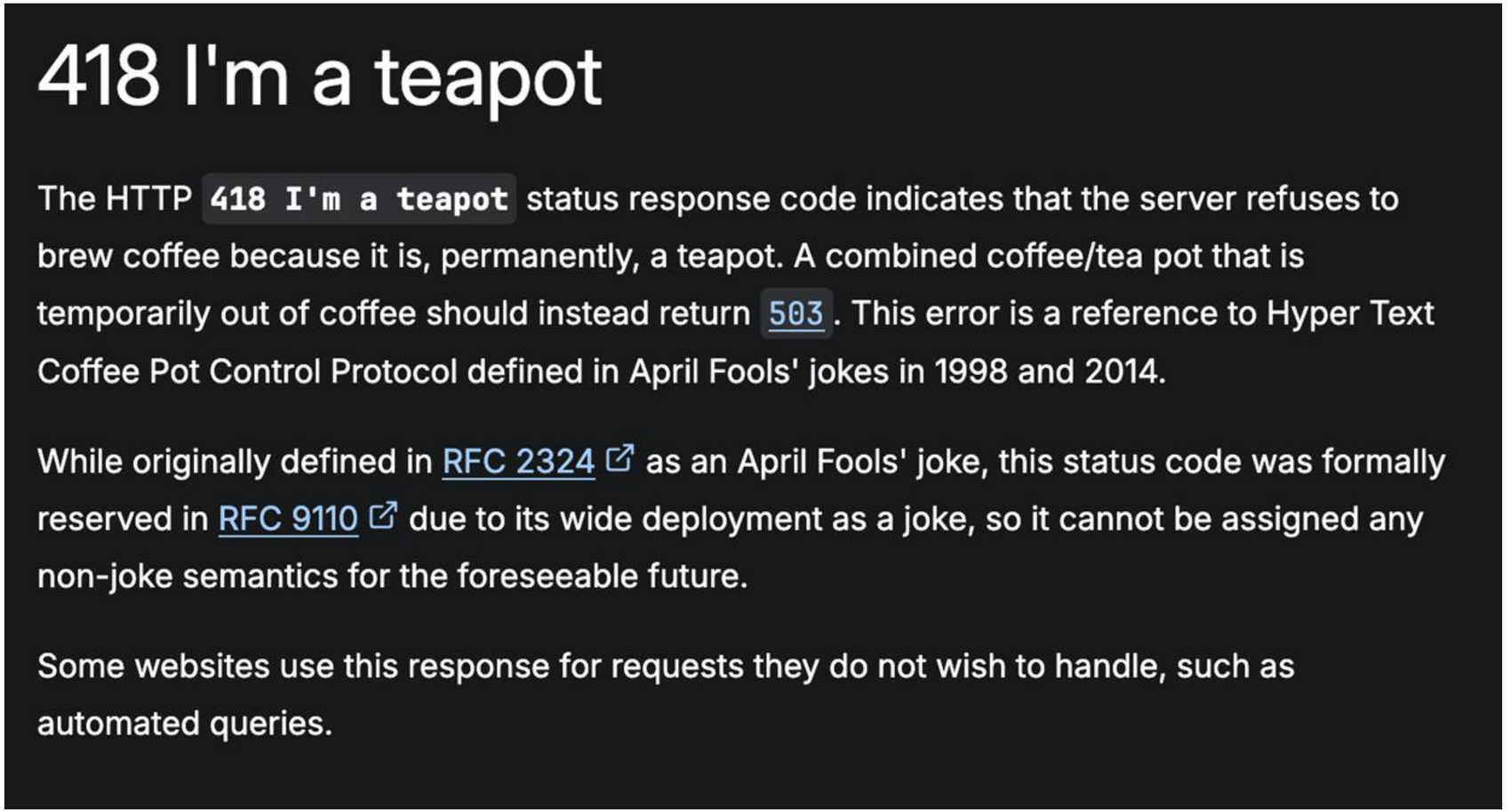
- **URL path (possibly with query parameters)**
- **Method**
  - **GET**: Requests that don't change server-side state ("*get*" information from the server)
  - **POST**: Request that update server-side state ("*post*" information to the server)
- **Data**
  - GET requests do not contain any data
  - POST requests can contain data

Parts of an HTTP Response



- **Status code: A number indicating what happened with the request**
  - Example: **200 OK**
  - Example: **403 Access forbidden**
  - Example: **404 Page not found**
- **Data**
  - Can be a webpage, image, audio, PDF, executable, etc.

Parts of an HTTP Response



418 I'm a teapot

The HTTP `418 I'm a teapot` status response code indicates that the server refuses to brew coffee because it is, permanently, a teapot. A combined coffee/tea pot that is temporarily out of coffee should instead return `503`. This error is a reference to Hyper Text Coffee Pot Control Protocol defined in April Fools' jokes in 1998 and 2014.

While originally defined in RFC 2324 as an April Fools' joke, this status code was formally reserved in RFC 9110 due to its wide deployment as a joke, so it cannot be assigned any non-joke semantics for the foreseeable future.

Some websites use this response for requests they do not wish to handle, such as automated queries.

Parts of an HTTP Request



```
POST / HTTP/1.1
Content-Length: 52

{
“data”: “ABC123”
}
```

Summary



- What's the web?
- URLs
- HTTP
- **Parts of a Webpage**
  - HTML
  - CSS
  - JavaScript
- Security on the Web
- Same-Origin Policy

HTML

- **HTML (Hypertext Markup Language)**: A markup language to create structured documents
- **Defines elements on a webpage with *tags***
  - Tags are defined with angle brackets **< >**
  - Each tag has a **name** and optional properties inside the brackets.
  - Some tags stand alone
    - Example: **
** for a line break
  - Most tags are opened and closed with *content* in between
    - Example: **** tag creates bold text: **Bla Blah blah** → **Bla Blah blah**
  - Tags can be nested to accomplish an effect.
    - Example: ** Bla Blah blah ** → ***Bla Blah blah***

<name property=value**>***content***</name>**

Features of HTML: Links

```
<a href="https://toon.cs161.org">
    Check out these comics!
</a>
```



HTML

Webpage

Features of HTML: Forms

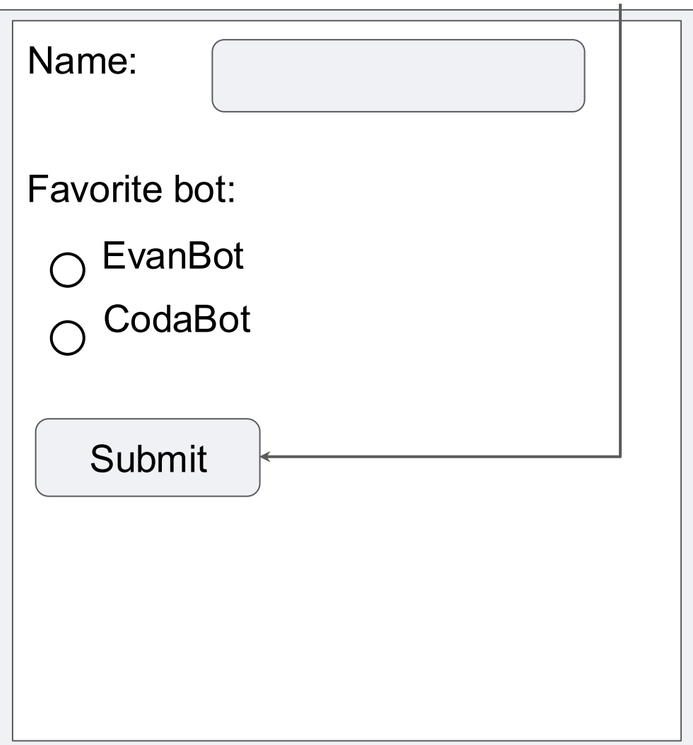
Clicking on the submit button makes a POST request to https://toon.cs161.org/feedback with the contents of the form



```
<form action="/feedback" method="POST">
        <label for="name">Name:</label>
        <input type="text" id="name">
        <br>
        <label for="bot">Favorite bot:</label><br>
        <input type="radio" id="evan">
        <label for="html">EvanBot</label><br>
        <input type="radio" id="coda">
        <label for="css">CodaBot</label><br>
        <br>
        <input type="submit" value="Submit">
</form>
```

HTML

The HTML between the <form> tags creates the form fields for the user to fill in.



Webpage

Features of HTML: Embed Images

```
<p>Look at my new desktop background!</p>
<img
src="https://toon.cs161.org/assets/desktop.png">
```

The browser will make a GET request to https://toon.cs161.org/assets/desktop.png and display the returned image on the page.

HTML

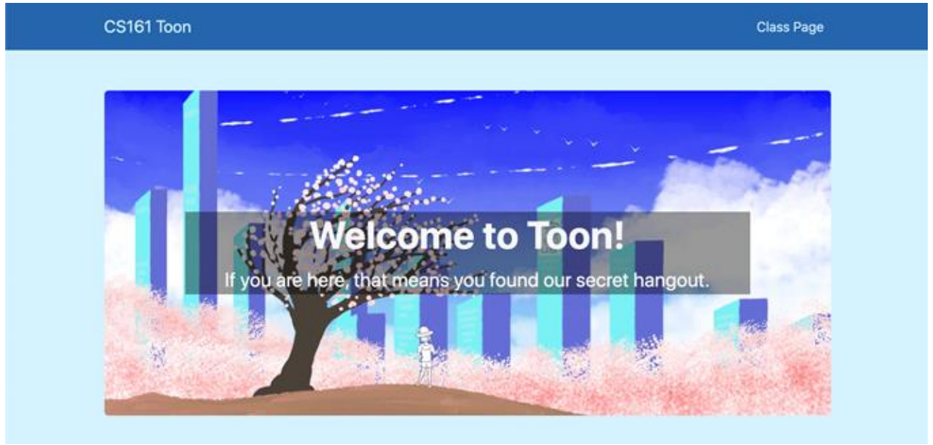
Look at my new desktop background!



Webpage

Features of HTML: Embed Websites

```
<iframe
        src="https://toon.cs161.org"
        height="500"
        width="1000"
></iframe>
<p>CS 161 toon website above.</p>
```



CS 161 toon website above.

The browser will make a GET request to https://toon.cs161.org/ and display the returned page in a 500 × 1000 pixel box.

HTML

Webpage

The outer frame embeds the inner frame (sometimes called an iframe or frame)

CSS



- **CSS (Cascading Style Sheets):** A style sheet language for defining the appearance of webpages
  - You don’t need to know the specifics of CSS

JavaScript

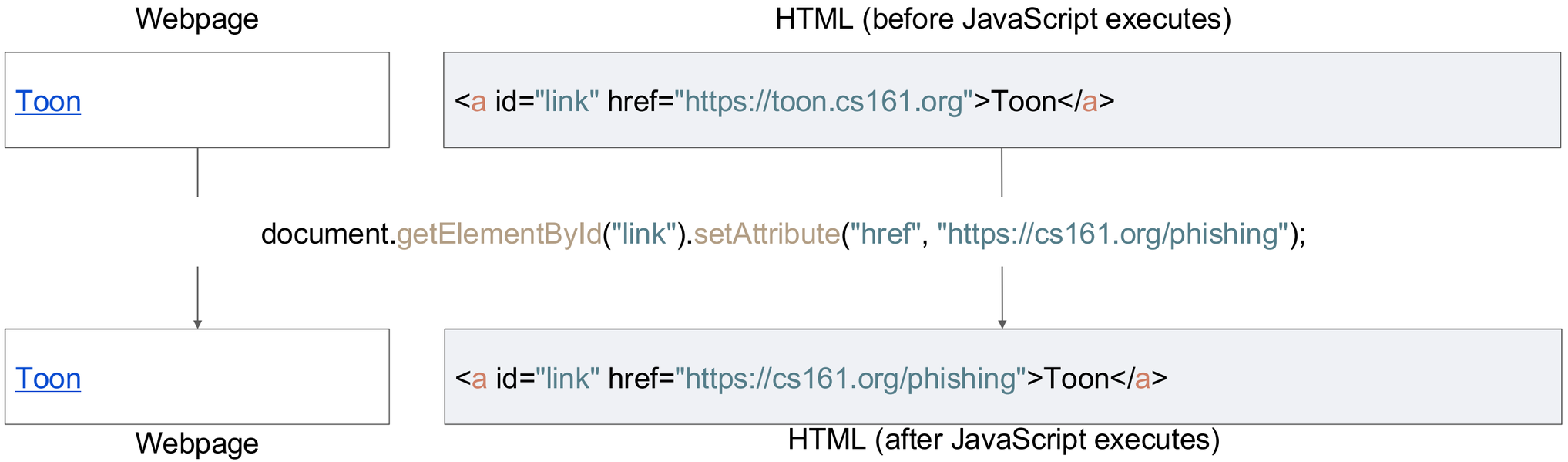


- **JavaScript: A programming language for running code in the web browser**
- **JavaScript is client-side**
  - Code sent by the server as part of the response
  - Runs in the browser, not the web server!
- **Used to manipulate web pages (HTML and CSS)**
  - Makes modern websites interactive
  - JavaScript can be directly embedded in HTML with **<script>** tags
- **Most modern webpages involve JavaScript**
  - JavaScript is supported by all modern web browsers
- **You don’t need to know JavaScript syntax**
  - However, knowing common attack functions helps

Features of JavaScript: Modify Content



- Modify any part of the webpage (e.g. HTML or CSS)

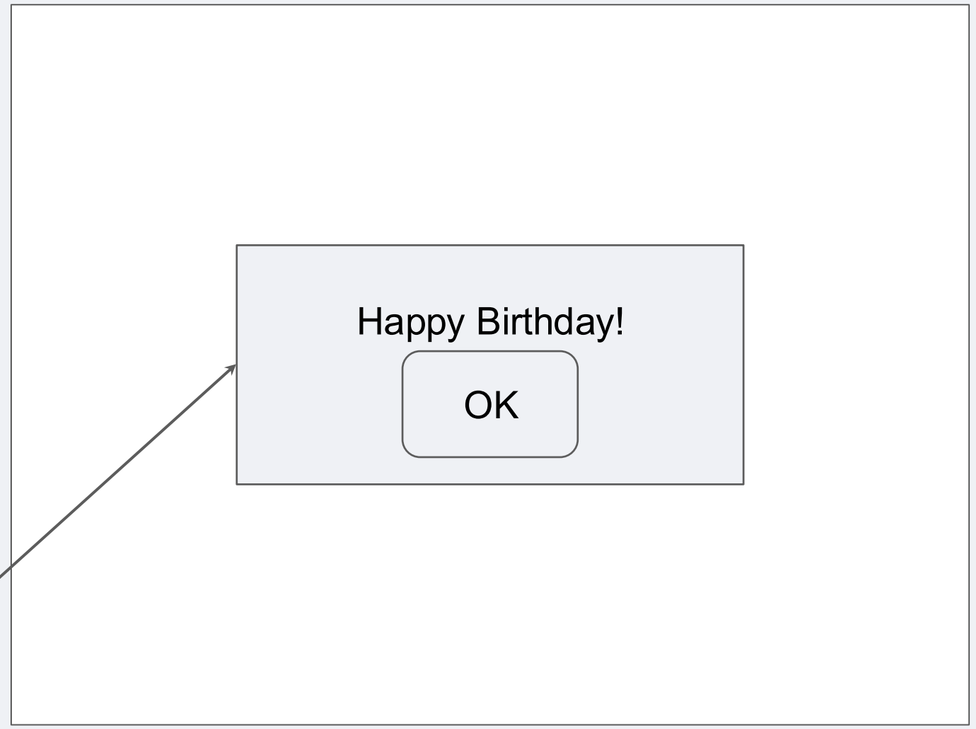


JavaScript changed the link! Now clicking it opens https://cs161.org/phishing

Features of JavaScript: Pop Ups

```
<script>
        alert("Happy Birthday")
</script>
```

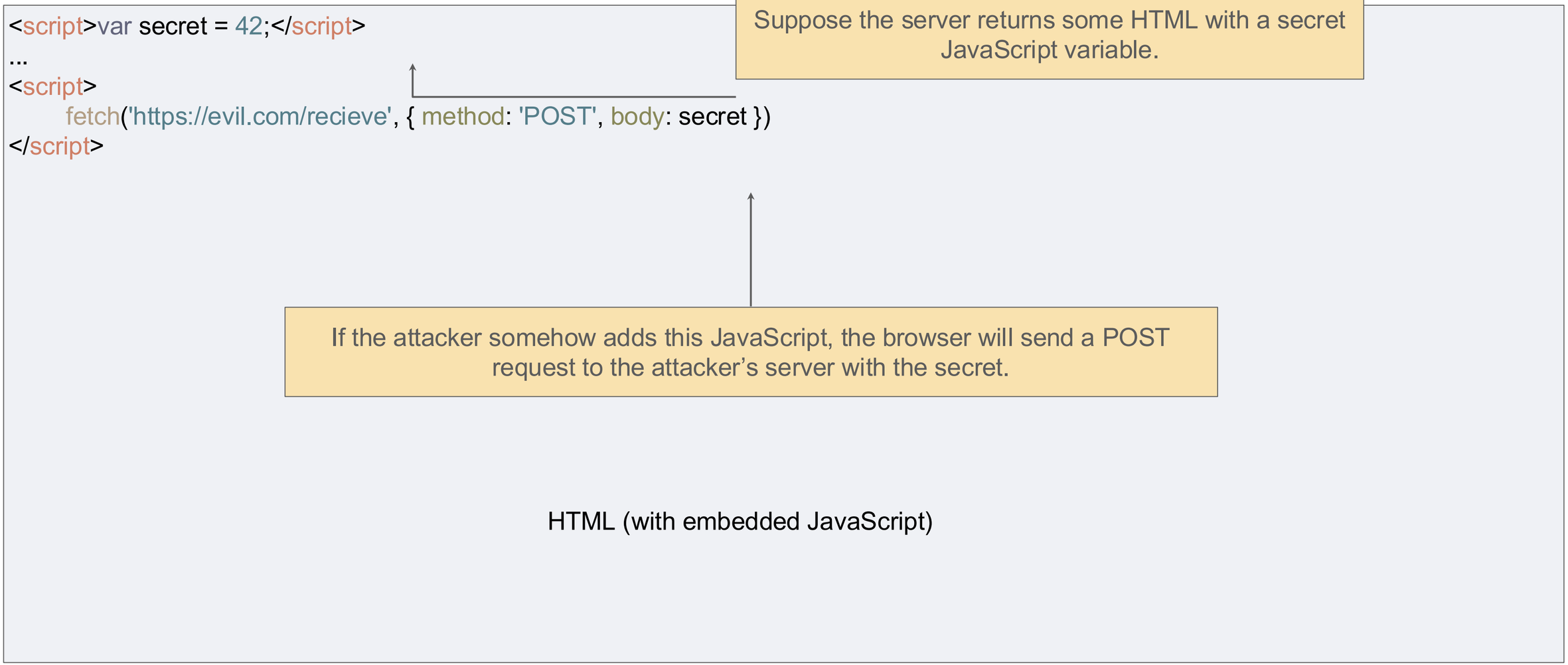
When the browser loads this HTML, it will run the embedded JavaScript and cause a pop-up to appear.



HTML (with embedded JavaScript)

Webpage

Features of JavaScript: HTTP Requests



Rendering a Webpage



- **Process of displaying (rendering) a webpage in a web browser:**
  - The browser receives HTML, CSS, and JavaScript from the server
  - HTML and CSS are parsed into a **DOM (Document Object Model)**
  - JavaScript is interpreted and executed, possibly modifying the DOM
  - The painter uses the DOM to draw the webpage
- **DOM (Document Object Model)**: Cross-platform model for representing and interacting with objects in HTML
  - A tree of nodes
  - Each node has a tag, attributes, and child nodes

```
document.getElementById("link").setAttribute("href", "https://cs161.org/phishing");
```

Summary



- What's the web?
- URLs
- HTTP
- Parts of a Webpage
  - HTML
  - CSS
  - JavaScript
- **Security on the Web**
- Same-Origin Policy

Risks on the Web



- **Risk #1: Web servers should be protected from unauthorized access**
  - Example: An attacker should not be able to hack into **google.com** and provide malicious search results to users
- **Protection: Server-side security**
  - Example: Protect the server computer from buffer overflow attacks

Risks on the Web



- **Risk #2: A malicious website should not be able to damage our computer**
  - Example: Visiting **evil.com** should not infect our computer with malware
  - Example: If we visit **evil.com**, the attacker who owns **evil.com** should not be able to read/write files on our computer
- **Protection: Sandboxing**
  - JavaScript is not allowed to access files on our computer
  - Privilege separation, least privilege
  - Browsers are carefully written to avoid vulnerabilities (e.g., write the browser in a memory-safe language)

Risks on the Web



- Risk #3: A malicious website should not be able to tamper with our interactions with other websites
  - Example: If we visit **evil.com**, the attacker who owns **evil.com** should not be able to read our emails in Gmail or buy things with our Amazon account
- **Protection: Same-origin policy**
  - The web browser prevents a website from accessing other unrelated websites

Risks on the Web



- Risk #4: A malicious website should not be able to compromise our web browser
  - Example: If we visit **evil.com**, the attacker who owns **evil.com** should not be able to exploit our web browser
- **Protection: Sandboxing**
  - The web browser prevents a website from accessing other websites and underlying software components
  - The rendering engine has a least privilege

Summary



- What's the web?
- URLs
- HTTP
- Parts of a Webpage
  - HTML
  - CSS
  - JavaScript
- Security on the Web
- **Same-Origin Policy**

Same-Origin Policy: Definition



- **Same-origin policy**: A rule that prevents one website from tampering with other unrelated websites
  - Enforced by the web browser
  - Prevents a malicious website from tampering with behavior on other websites

Same-Origin Policy



- **Every webpage has an origin defined by its URL with three parts:**
  - **Protocol**: The protocol in the URL
  - **Domain**: The domain in the URL's location
  - ***Port***: The port in the URL's location
    - If no port is specified, the default is 80 for HTTP and 443 for HTTPS

https://toon.cs161.org:443/assets/lock.PNG

http://cs161.org/assets/images/404.png

80 (default port)

Same-Origin Policy



- Two webpages have the same origin *if and only if* the protocol, domain, and port of the URL all match exactly
  - Think string matching: The protocol, domain, and port strings must be equal

First domain	Second domain	Same origin?
http://toon.cs161.org	https://toon.cs161.org	No. Protocol mismatch http ≠ https
http://toon.cs161.org	http://cs161.org	No. Domain mismatch toon.cs161.org ≠ cs161.org
http://toon.cs161.org	http://toon.cs161.org:8000	No. Port mismatch 80 ≠ 8000

Same-Origin Policy



- **Webpages from different origins cannot interact with each other**
  - Example: If **cs161.org** embeds **google.com**, the inner frame cannot interact with the outer frame, and the outer frame cannot interact with the inner frame
- **Exception: JavaScript runs with the origin of the page that loads it**
  - Example: If **cs161.org** fetches JavaScript from **google.com**, the JavaScript has the origin of **cs161.org**
  - Intuition: **cs161.org** has "copy-pasted" JavaScript onto its webpage
- **Exception: webpages can fetch and display images from other origins**
  - However, Javascript in the webpage only knows about the image's size and dimensions (cannot actually manipulate the image)
- **Exception: Websites can agree to allow some limited sharing**
  - Cross-origin resource sharing (CORS)
  - The **postMessage** function in JavaScript

Summary



- **What's the web?**
- **URLs**
- **HTTP**
- **Parts of a Webpage**
  - HTML
  - CSS
  - JavaScript
- **Security on the Web**
- **Same-Origin Policy**