CSE551: Advanced Computer Security

11. Introduction & Web Programming

Seongil Wi

Department of Computer Science and Engineering

Introduction to Web Security

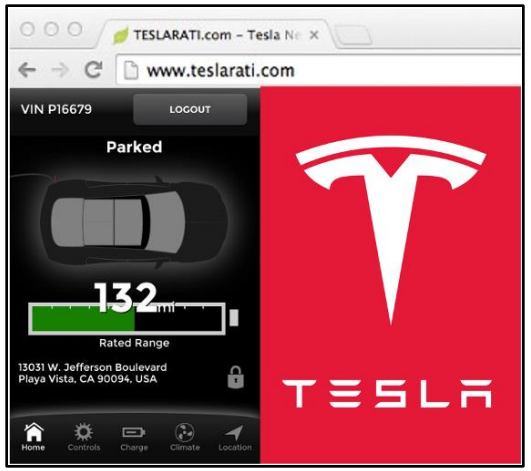
The Web has won

- Used by billions of people to store/retrieve information

2B users monthly

2.3M searches per second

- User interface for emerging systems



Self-driving car

Cryptocurrency

OpenAI ChatGPT

AI models

- Large coverage in desktop/mobile application

WebView

WebVR

… and the hackers with it

- Used by billions of people to store/retrieve information

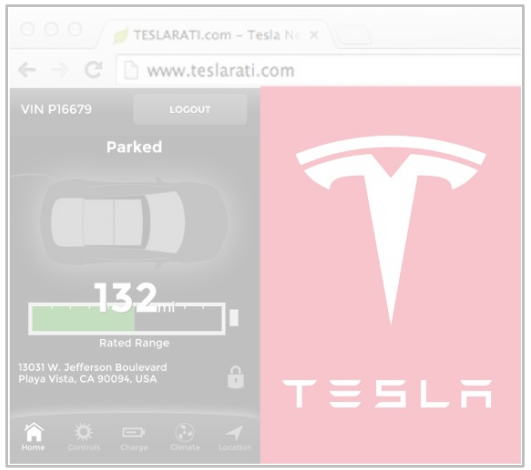
2B users monthly

2.3M searches per second

- Large coverage in desktop/mobile applications

WebView

- Interface for emerging systems



Self-driving car

Cryptocurrency

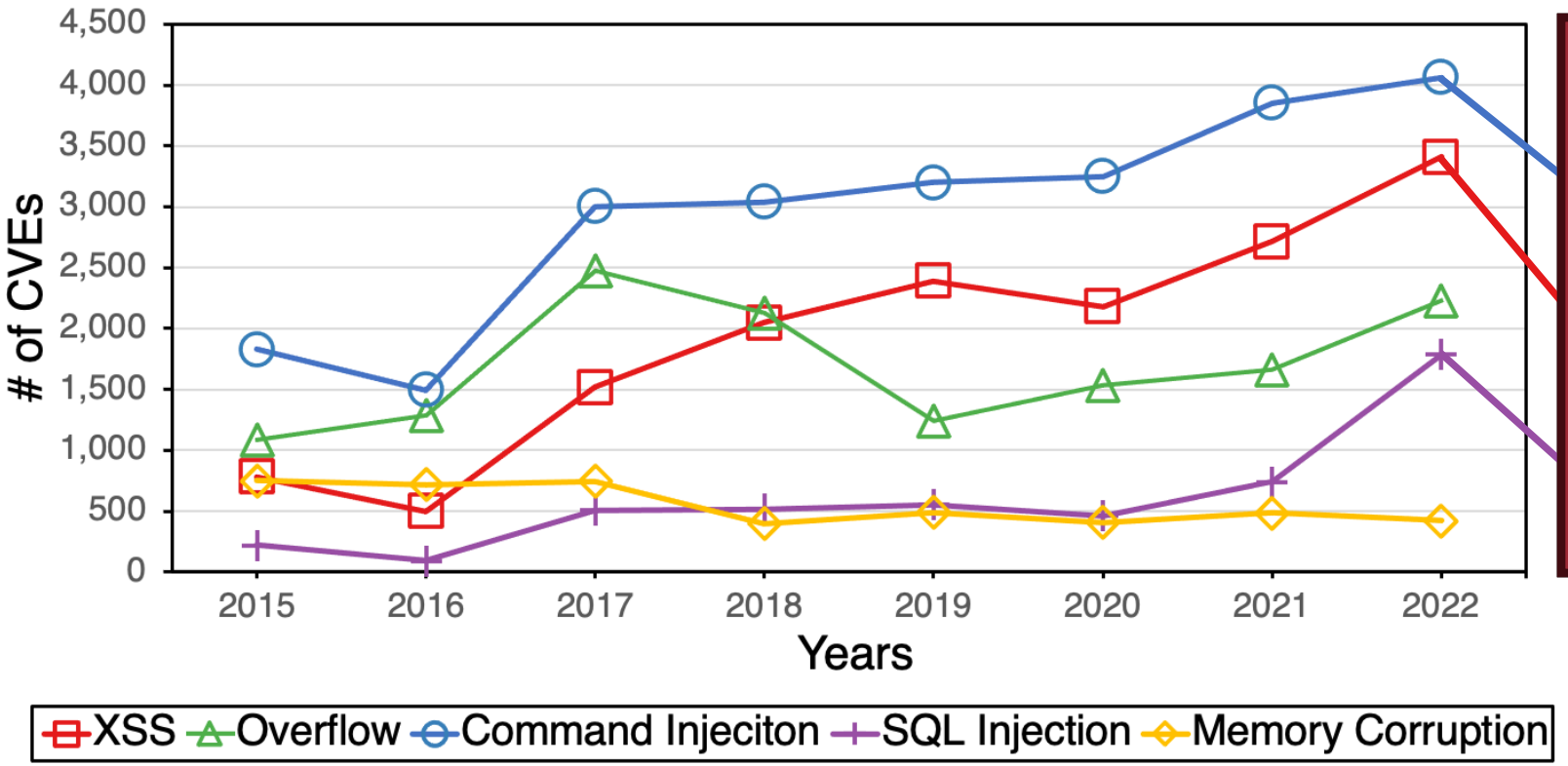
OpenAI ChatGPT

AI models

WebVR

Why Web Security?

- *Web attacks* accounted for **48.6%** of all reported threats



Web attacks

- Initiate denial-of-service (DoS) attacks
- Access to sensitive information
- Enable remote code execution

https://www.cvedetails.com/vulnerabilities-by-types.php

… and the hackers with it

Crypto

2FA compromise led to $34M Crypto.com hack

Anita Ramaswamy @anitaramaswamy / 3:13 AM GMT+9 • January 21, 2022 Comment

Companies paid $4.2M bug bounties

Jonathan Greig
January 17th, 2023

Briefs Cybercrime

Norton LifeLock says 925,000 accounts targeted by credential-stuffing attacks

Web skimming hackers infiltrate over 40 ecommerce websites - that we know of

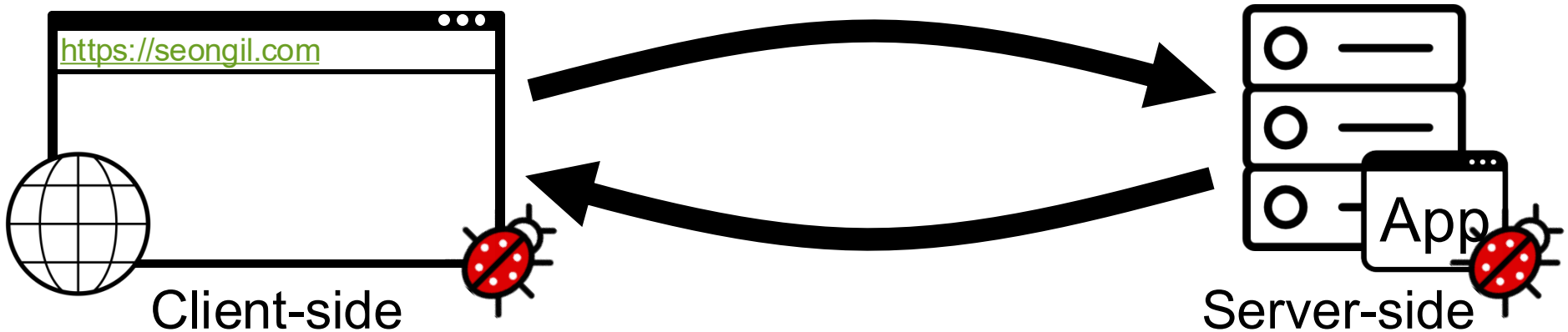
News By Abigail Opiah published December 08, 2022

Web threats are critical!

Introduction to Web Security

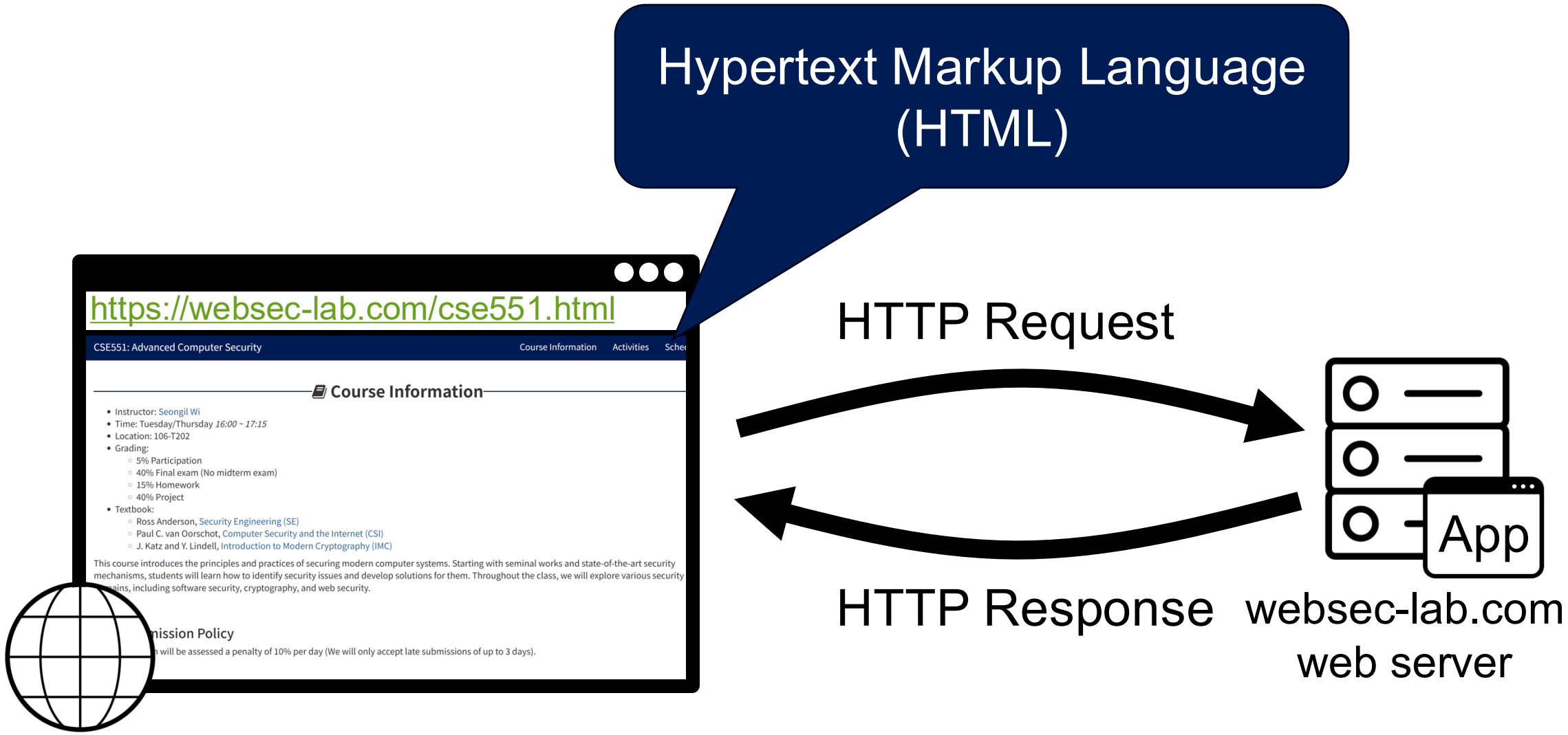
- We are going to study and discuss the web attacks and defenses.

- Web Programming Basic
- Server-side Web Attacks & Defenses
- Client-side Web Attacks & Defenses



Web Programming Basic

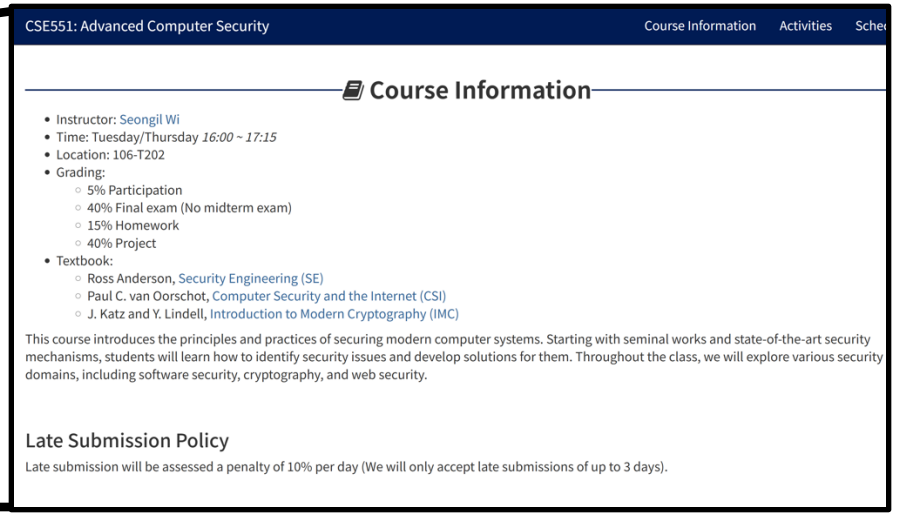
Web Infrastructure



Hypertext Markup Language (HTML)

- Markup language for web page layout
  - NOT programming language (i.e., for computation)!
- A web page (document) is written in HTML using markup tags
  - E.g., `<p>`, `<img>`

```
<html>
  <body>
    ...
  </body>
</html>
```



Hypertext Markup Language (HTML)

- Markup language for web page layout
  - NOT programming language (i.e., for computation)!
- A web page (document) is written in HTML using markup tags
  - E.g., `<p>, <img>`
- A **browser** interprets a web page when rendering the page
- Describes a hyper-text document
  - E.g., image, audio, video

What if we need computation?
⇒ JavaScript!

```
<html>
  <body>
    ...
  </body>
</html>
```

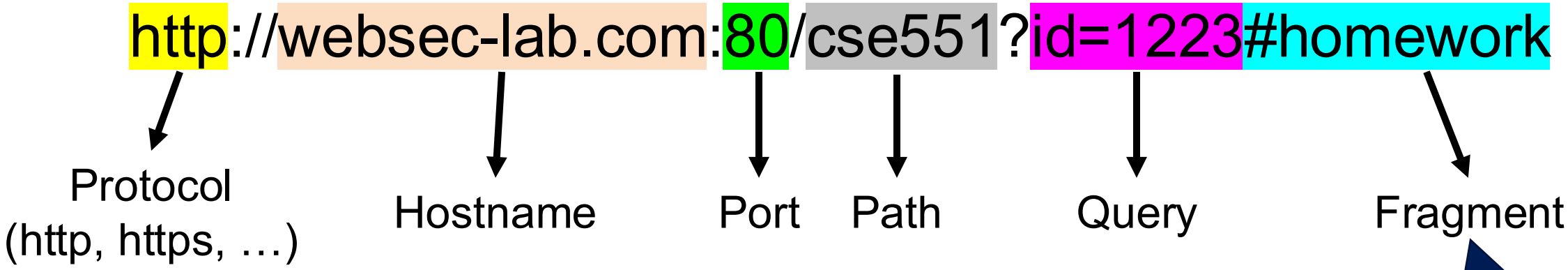
Uniform Resource Locators (URLs)

- Global identifiers of network-retrievable documents
- Example

http://websec-lab.com:80/cse551?id=1223#homework

Uniform Resource Locators (URLs)

- Global identifiers of network-retrievable documents
- Example

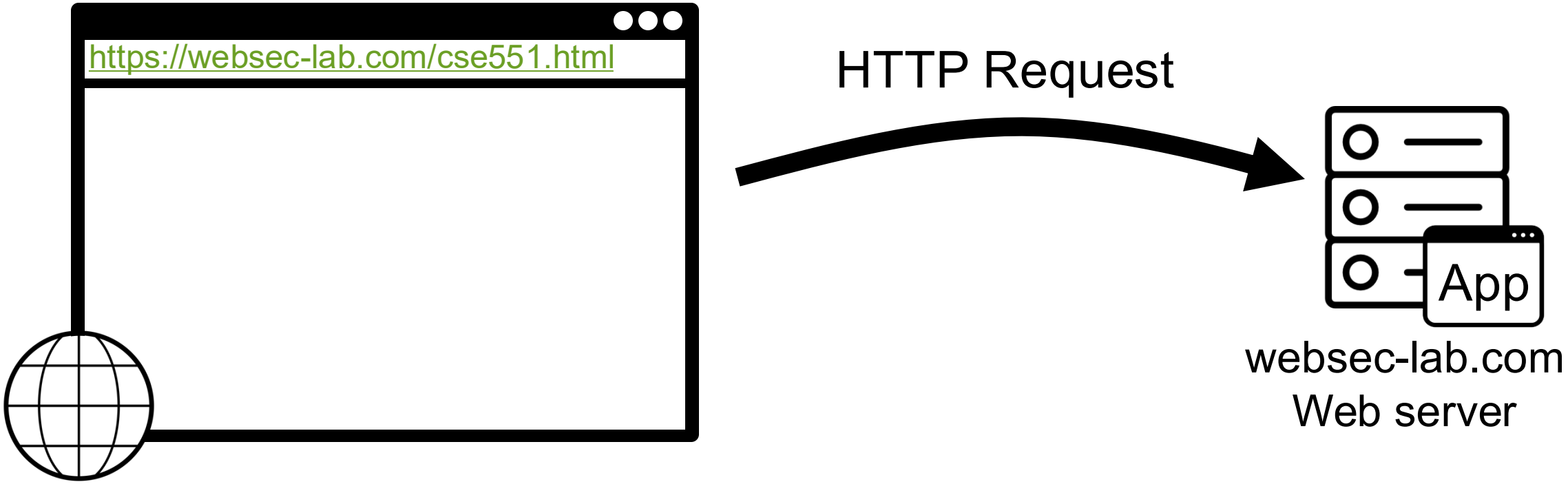


- Special characters are encoded as hex:
  - New line → %0A
  - Space → %20
  - + → %2B

Fragments are not sent to the server

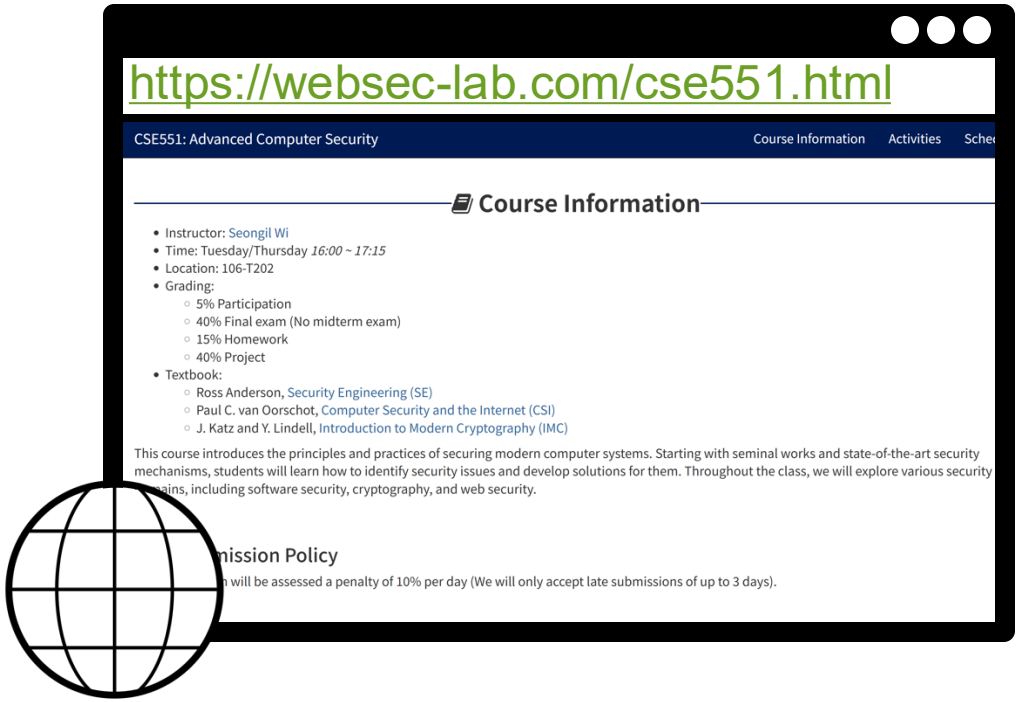
Hyper Text Transfer Protocol (HTTP)

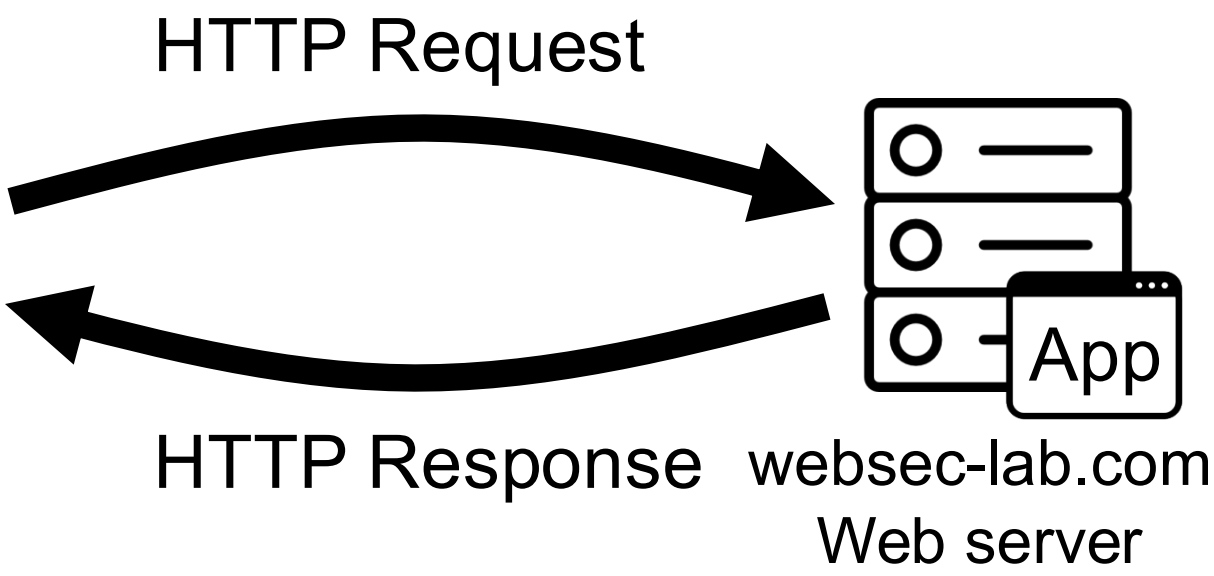
- The primary protocol for data transfer between web browsers and servers



Hyper Text Transfer Protocol (HTTP)

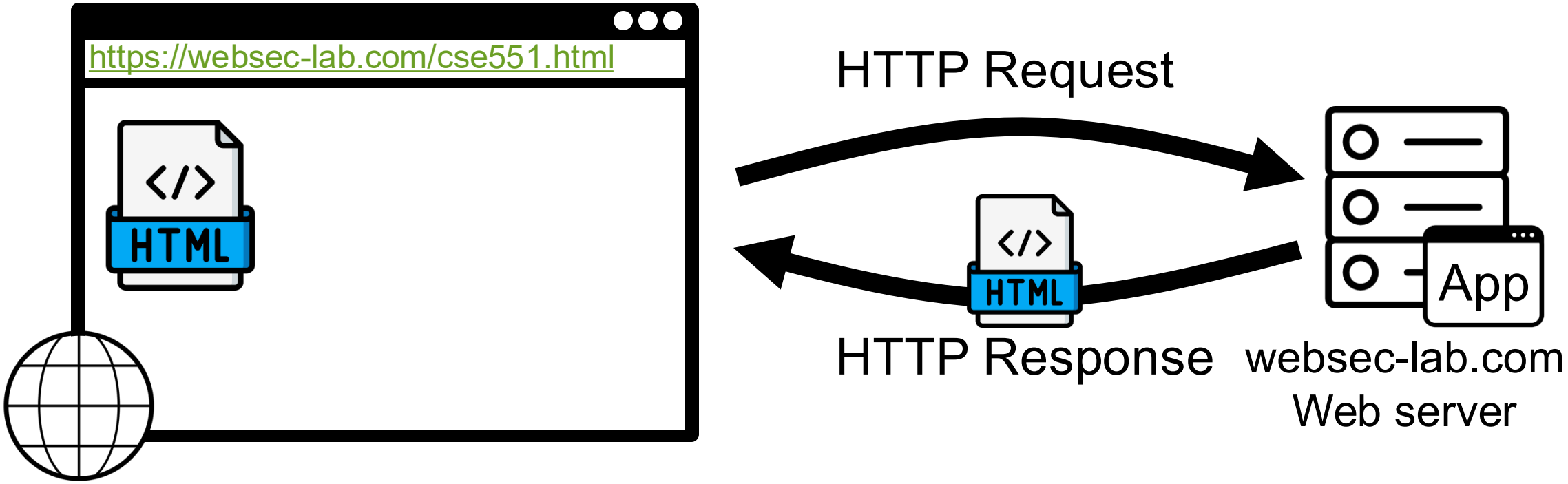
- The primary protocol for data transfer between web browsers and servers





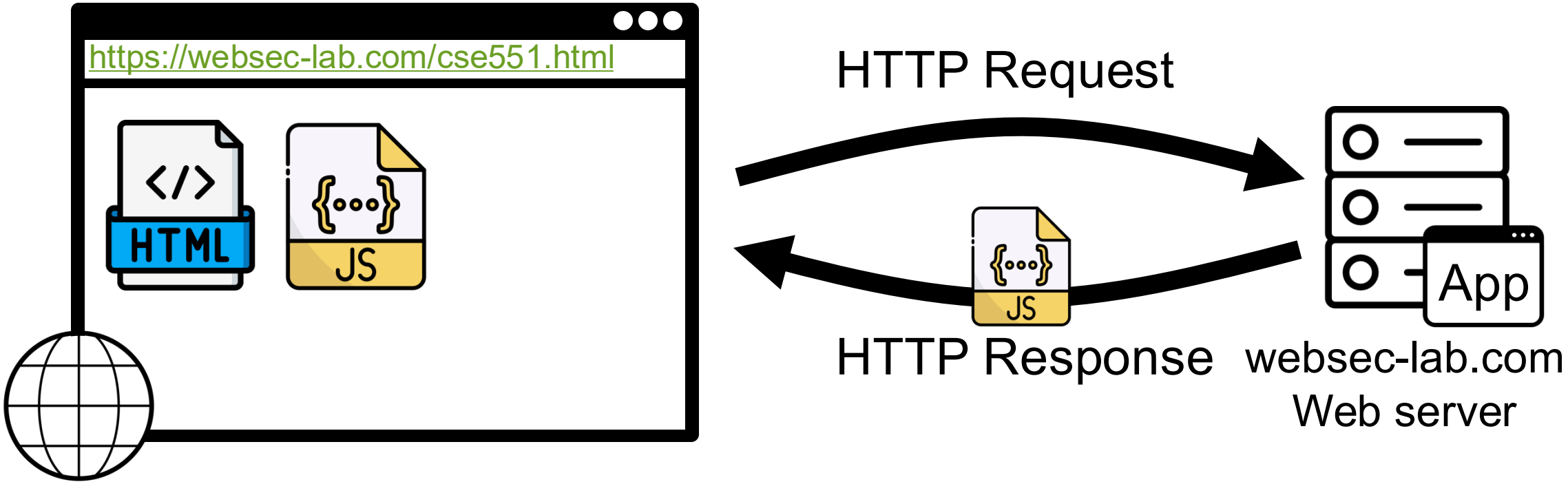
Hyper Text Transfer Protocol (HTTP)

- The primary protocol for data transfer between web browsers and servers



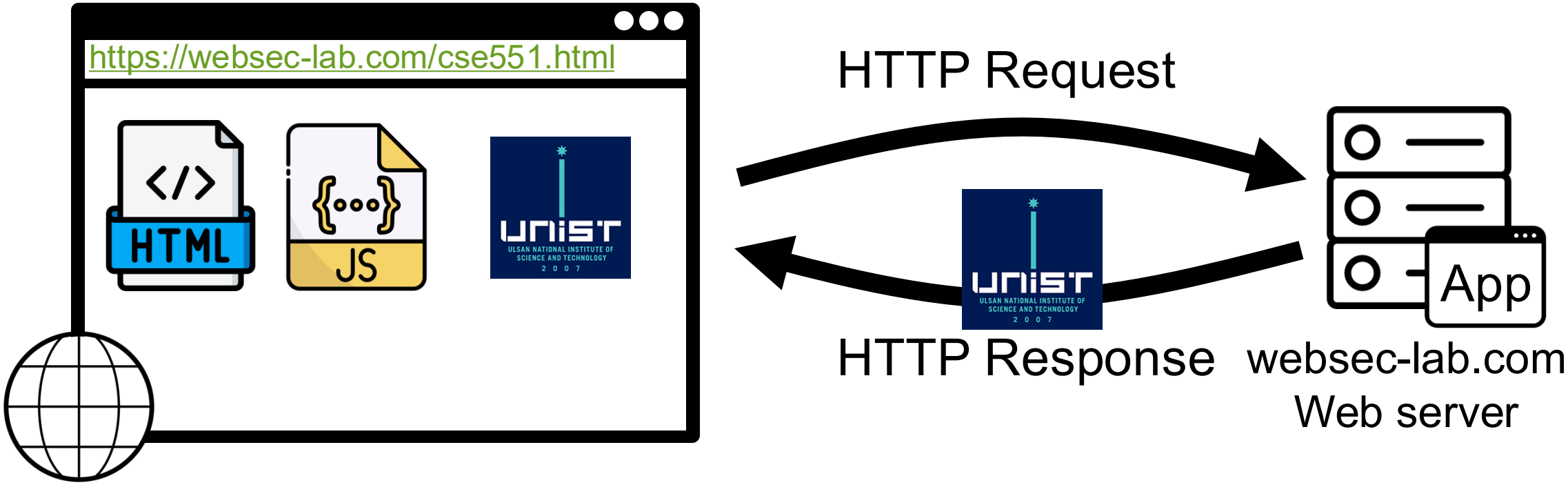
Hyper Text Transfer Protocol (HTTP)

- The primary protocol for data transfer between web browsers and servers



Hyper Text Transfer Protocol (HTTP)

- The primary protocol for data transfer between web browsers and servers



Hyper Text Transfer Protocol (HTTP)

- The primary protocol for data transfer between web browsers and servers

- Application layer protocol
  - A request is sent over a TCP connection on port:80

- Stateless request/response protocol
  - Each request is independent to previous requests

HTTP Request


```
GET  /cse551.html  HTTP/1.1
Host: websec-lab.com
Accept-Language: en
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;)
Referer: http://google.com?q=cse551
```

HTTP Request

Method | File path | Protocol

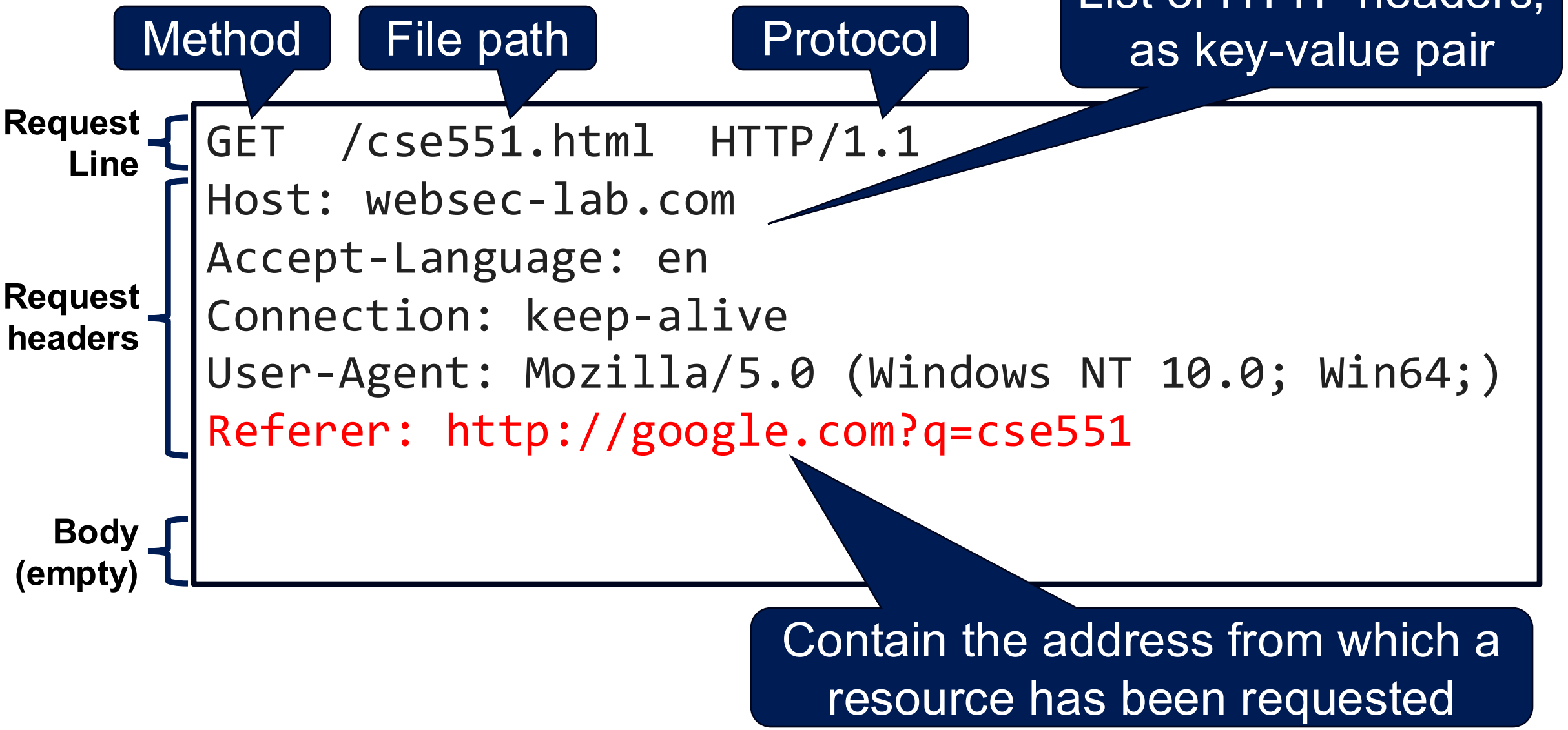
Request Line

```
GET  /cse551.html  HTTP/1.1
Host: websec-lab.com
Accept-Language: en
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;)
Referer: http://google.com?q=cse551
```

Many HTTP Methods

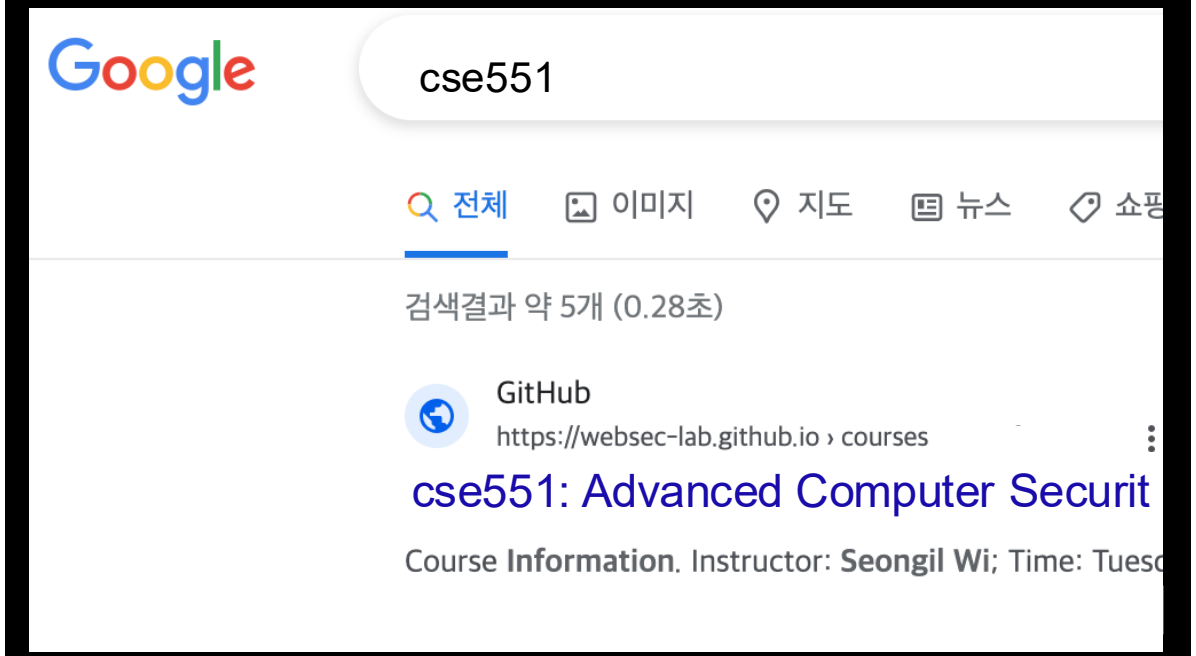
- **GET**: Get the resource at the specified URL
- **POST**: Create new resource at URL with payload
- **PUT**: Replace current representation of the target resource with request payload
- **PATCH**: Update part of the resource
- **DELETE**: Delete the specified URL

HTTP Request

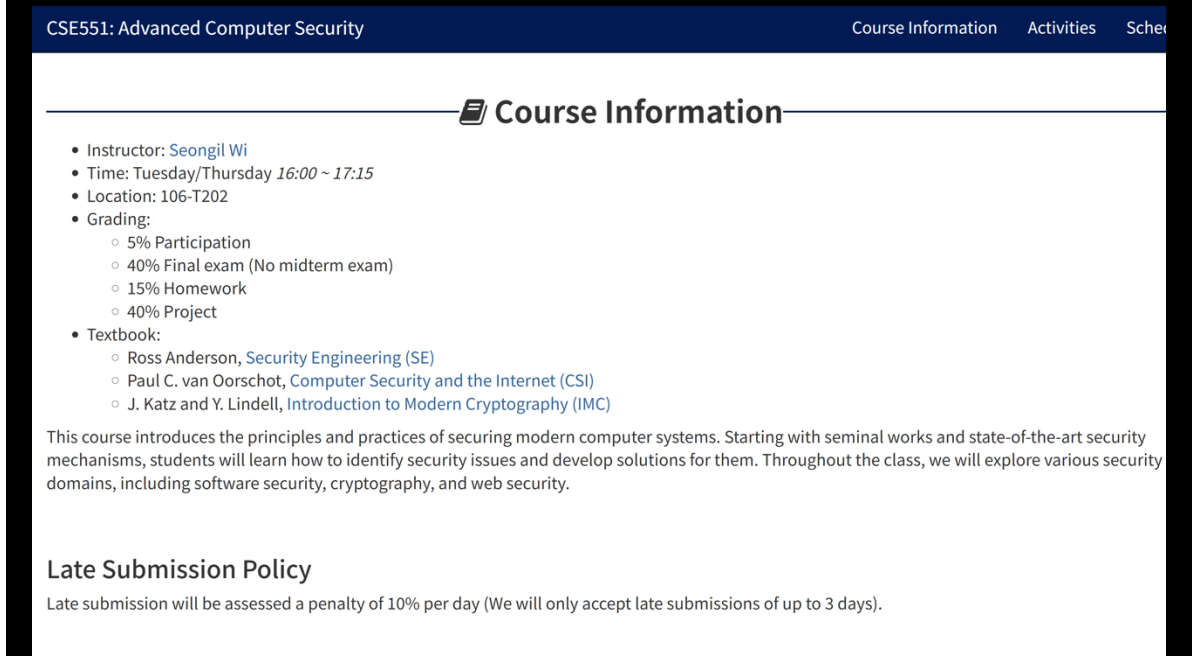


Referrer Header

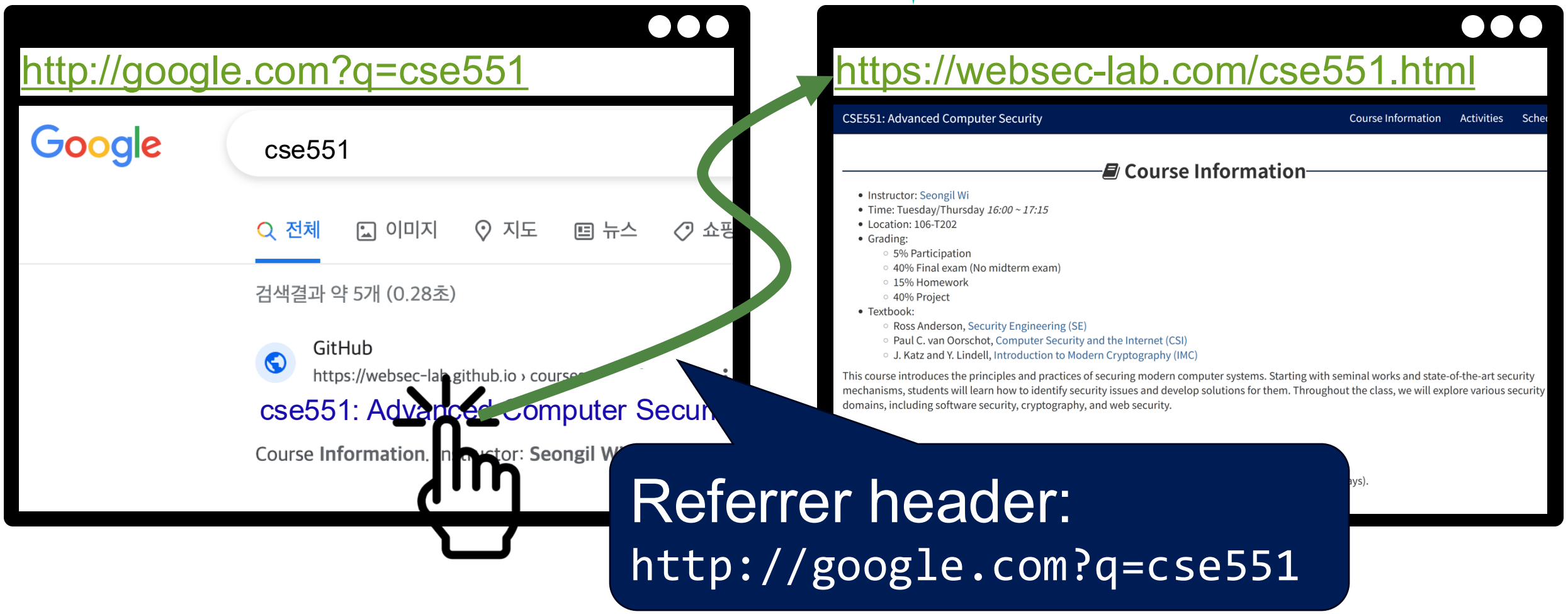
http://google.com?q=cse551



https://websec-lab.com/cse551.html

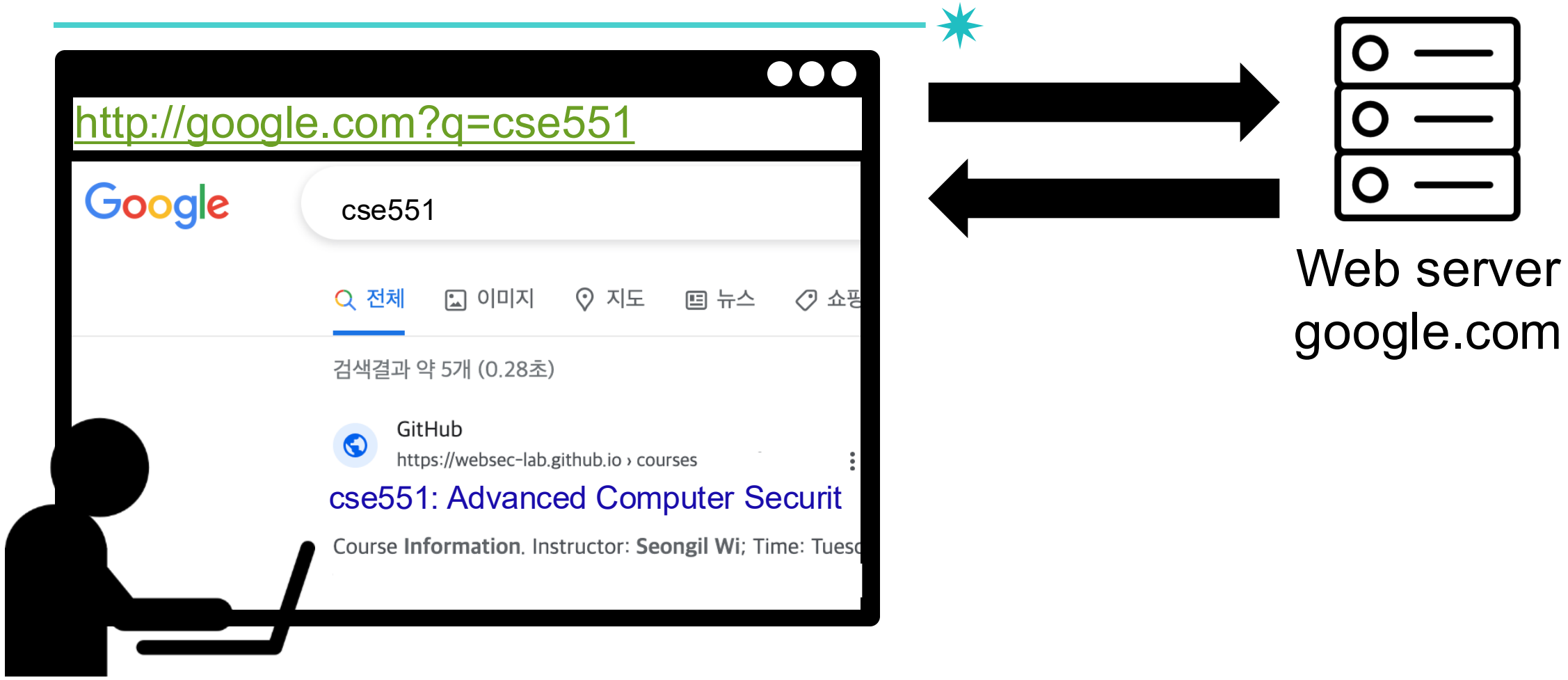


Referrer Header

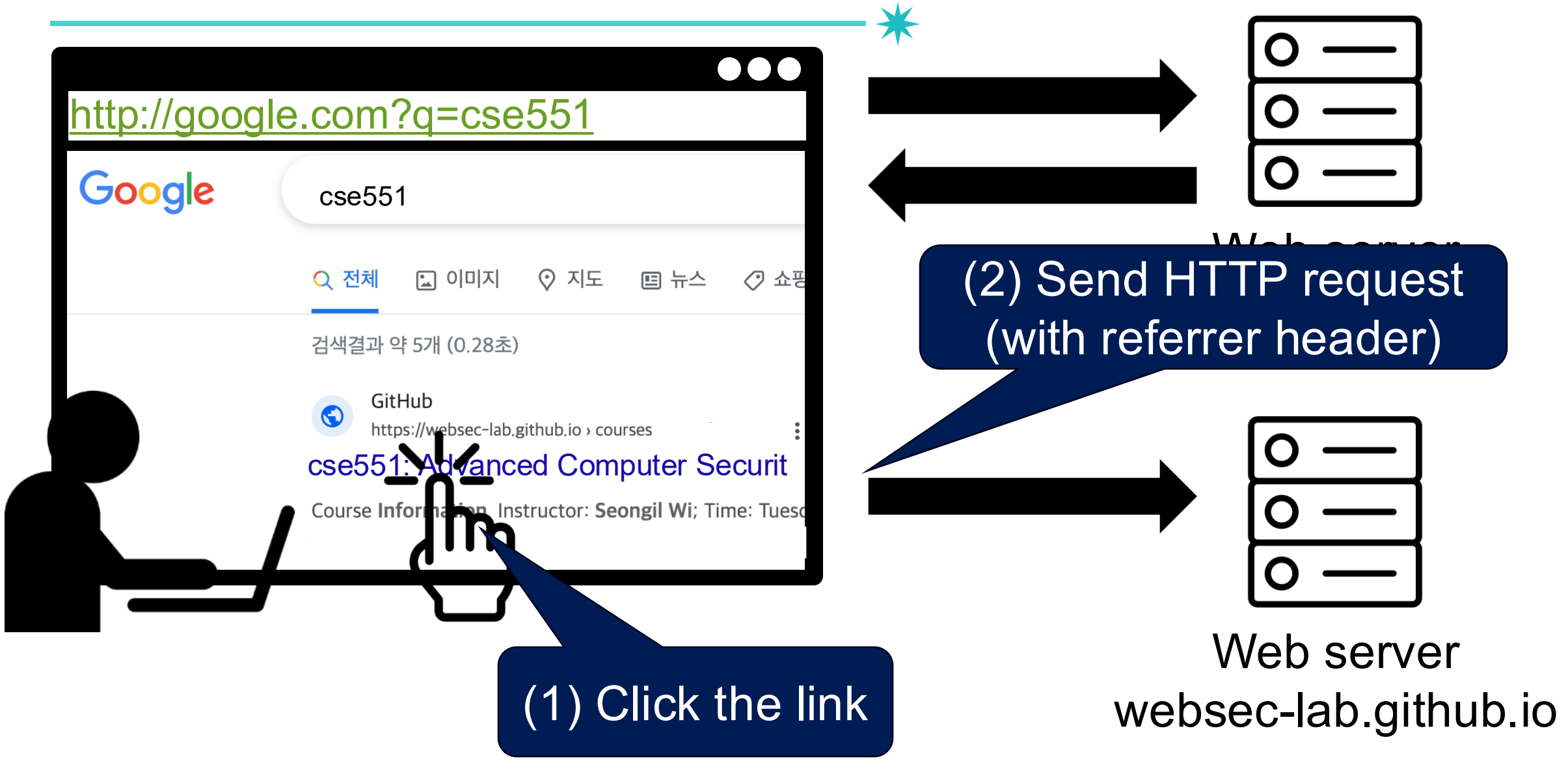


Referrer Header in Detail

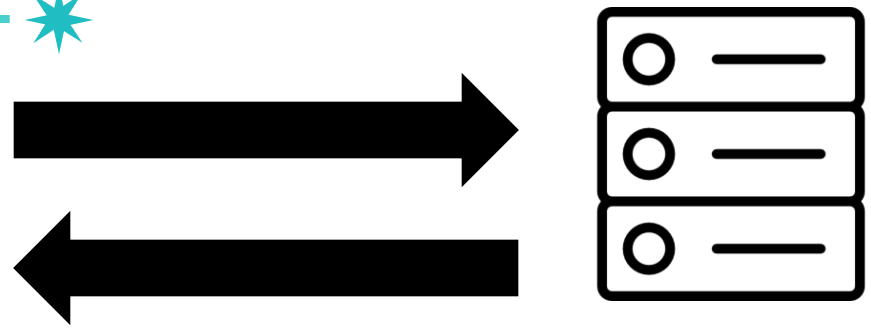
Referrer Header in Detail



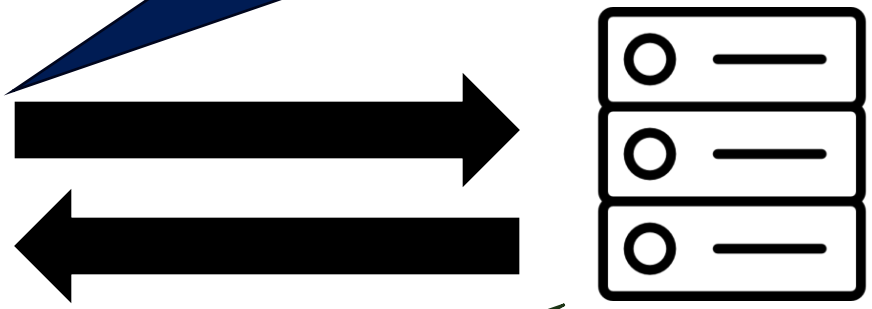
Referrer Header in Detail



Referrer Header in Detail



(2) Send HTTP request (with referrer header)



Web server
websec-lab.github.io

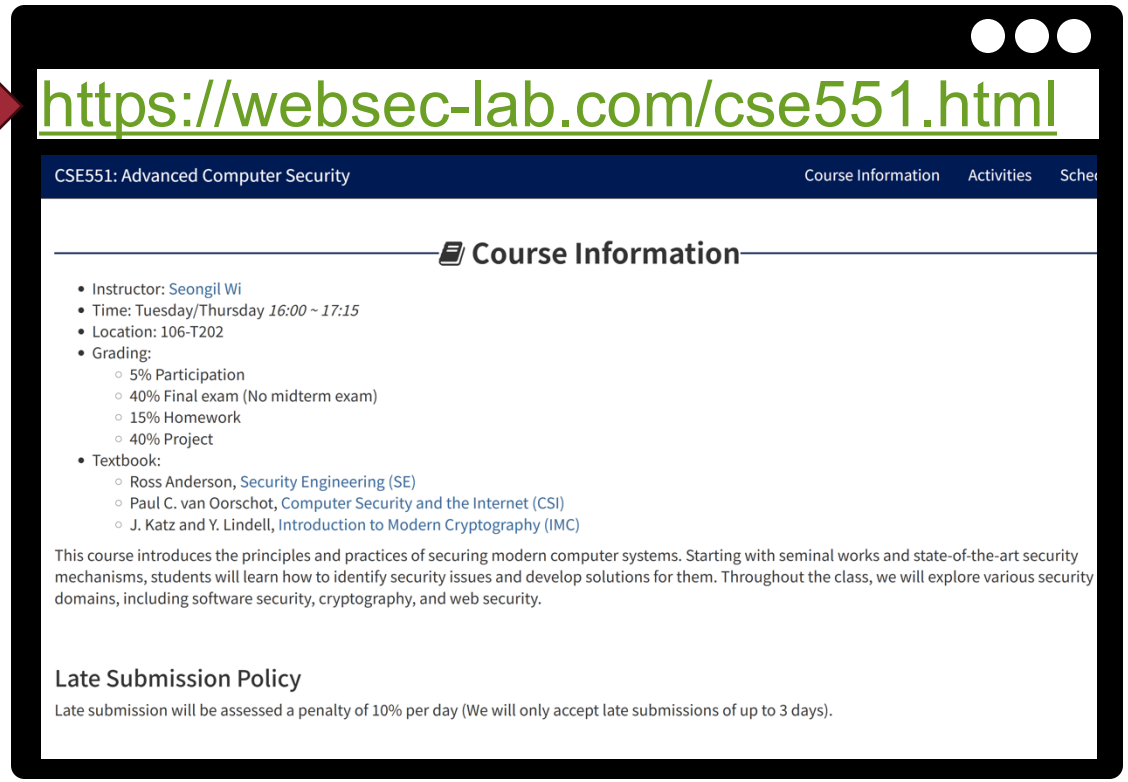
The server can analyze **where the request originated**

Question

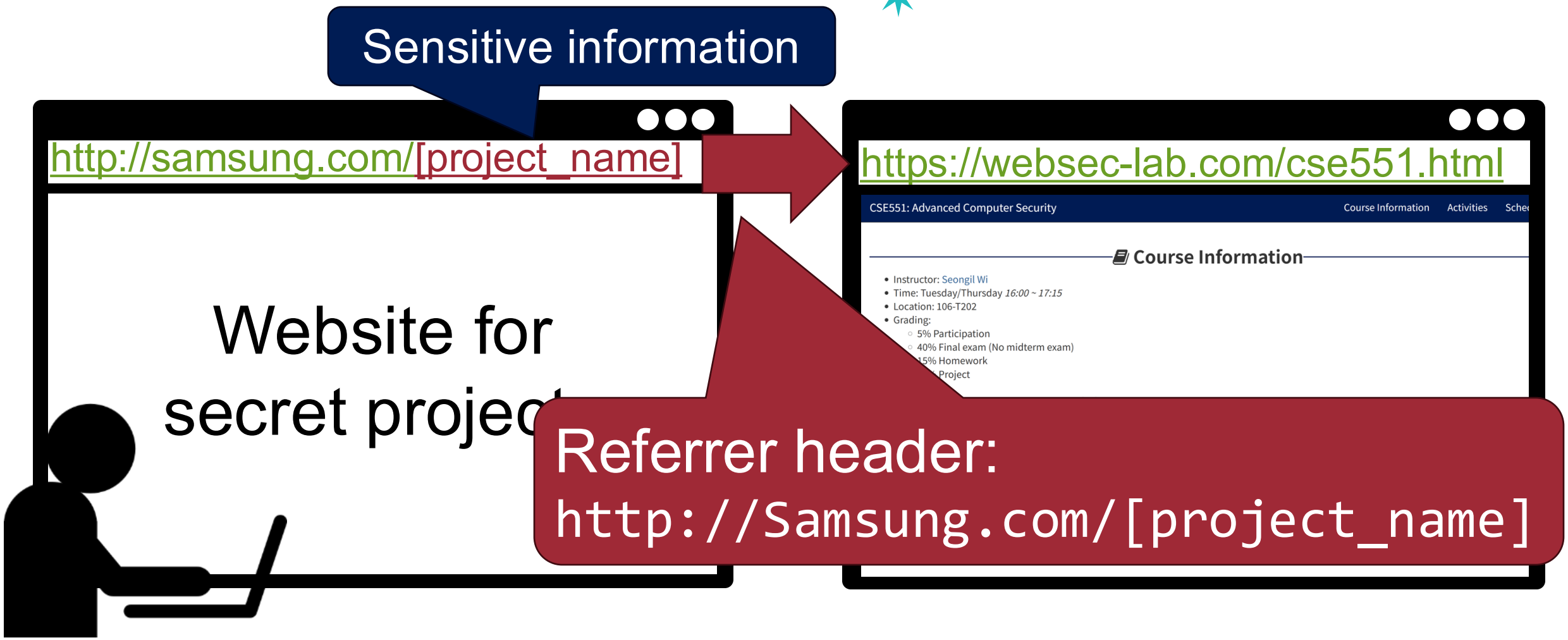
Are there any security issues with the referrer header?

Referrer Entails a Loss of Privacy

Sensitive information



Referrer Entails a Loss of Privacy



Referrer Entails a Loss of Privacy

Referrer Entails a Loss of Privacy – Example

private info

#1 https://music.example/superArtist123

#2 https://news.example/fr/search/?q=science&sort=latest

private + sensitive

#3 https://health-blog.example/path/sports-injuries/knee

private + identifying

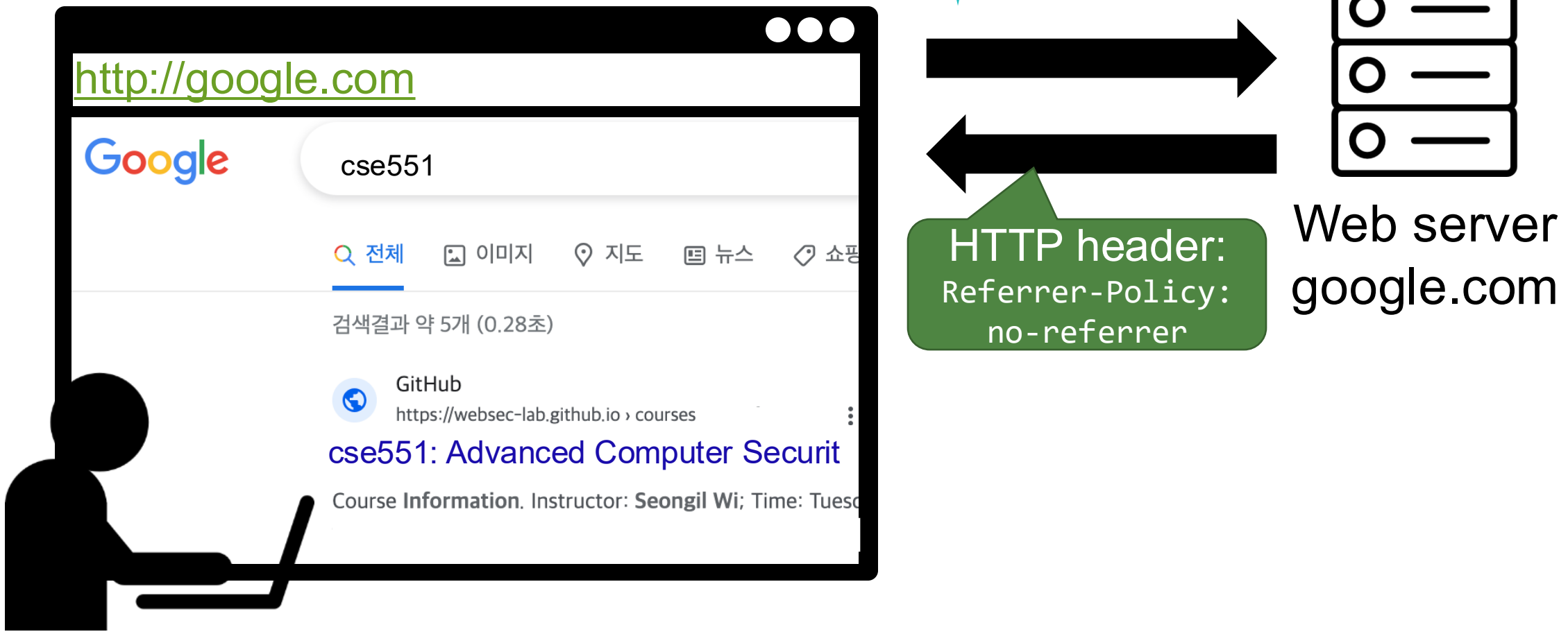
#4 https://social-network.example/my-account/johndoe86

#5 https://social-network.example/email_verified/?email=johndoe86@gmail.com

security-critical

#6 https://cloud-storage.example/625x1710s7Gtsr/password_reset

Defense: Referrer-Policy



Defense: Referrer-Policy

HTTP header:
`Referrer-Policy: no-referrer`

Web server
google.com

(2) Send HTTP request ***without*** referrer header

Web server
websec-lab.github.io

Defense: Referrer-Policy

Controls how much referrer information should be included with requests

- **`no-referrer`**: the referrer header will be omitted
- **`unsafe-url`**: Send the origin, path, and query string when performing any request
- **`strict-origin-when-cross-origin`**:
  - Send the origin, path, and query string when performing a same-origin request
  - For cross-origin requests send the origin (only), without path and query string

- Since Chrome 85, defaults to Referrer-Policy=strict-origin-when-cross-origin

HTTP Response

Status Line

Response headers

Response body

```
HTTP/1.1  200 OK
Date: Sat, 21 Oct 2023 07:58:24 GMT
Connection: Keep-alive
Content-Type: text/html
Content-Length: 2543

<html>
  <body>
    some data...
  </body>
</html>
```

HTTP Response

HTTP version | Status code | Status text

Status Line

Response headers

Response body

```
HTTP/1.1  200 OK
Date: Sat, 21 Oct 2023 07:58:24 G
Connection: Keep-alive
Content-Type: text/html
Content-Length: 2543

<html>
  <body>
    some data...
  </body>
</html>
```

HTTP STATUS CODES

2xx Success

200	Success / OK

3xx Redirection

301	Permanent Redirect
302	Temporary Redirect
304	Not Modified

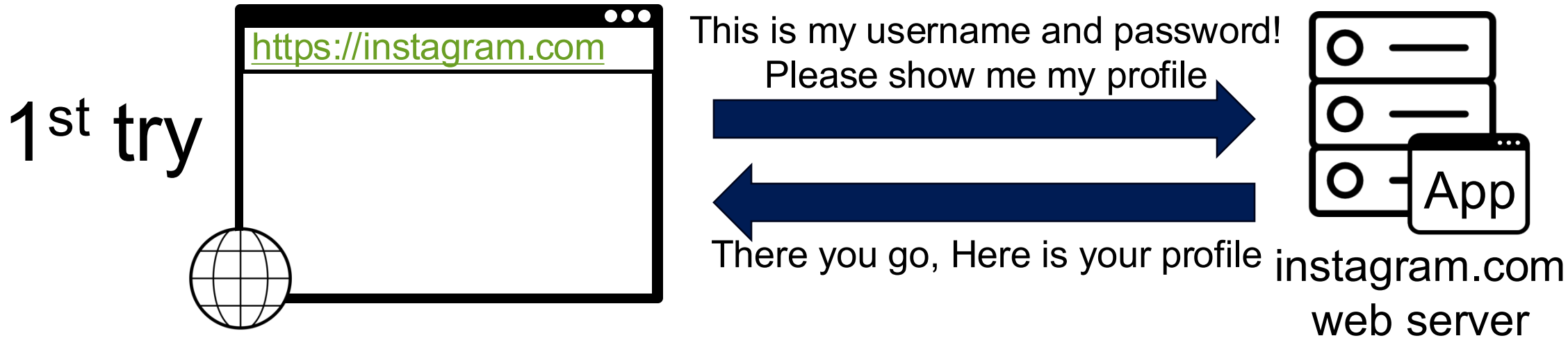
4xx Client Error

401	Unauthorized Error
403	Forbidden
404	Not Found
405	Method Not Allowed

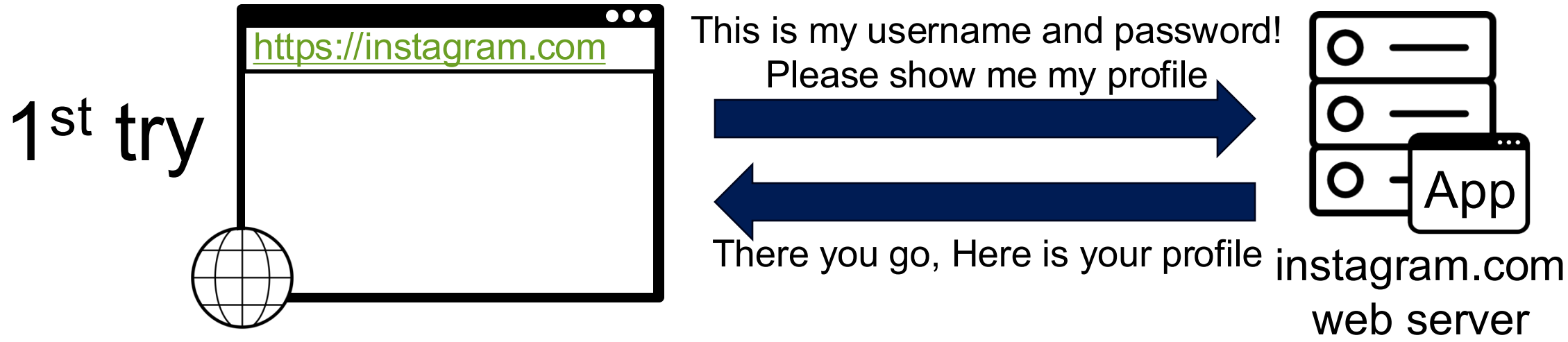
5xx Server Error

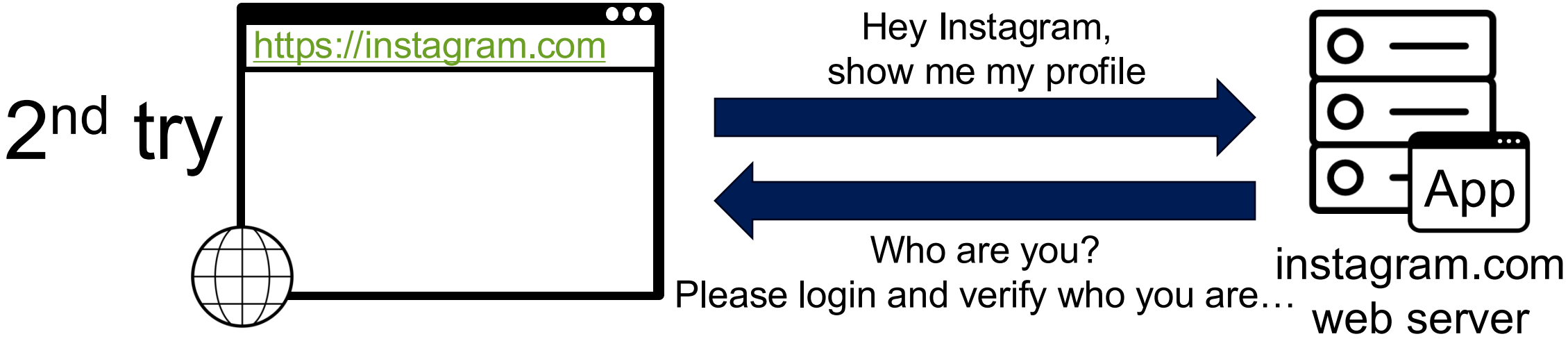
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout

HTTP is a Stateless Protocol



HTTP is a Stateless Protocol





HTTP is a Stateless Protocol

Stateless protocol:
Each request is independent to previous request

2nd try

is my username and password!
Please show me my profile

There you go, Here is your profile

App

instagram.com
web server

Hey Instagram,
show me my profile

Who are you?
Please login and verify who you are…

App

instagram.com
web server

Question

How to make HTTP “act” stateful?

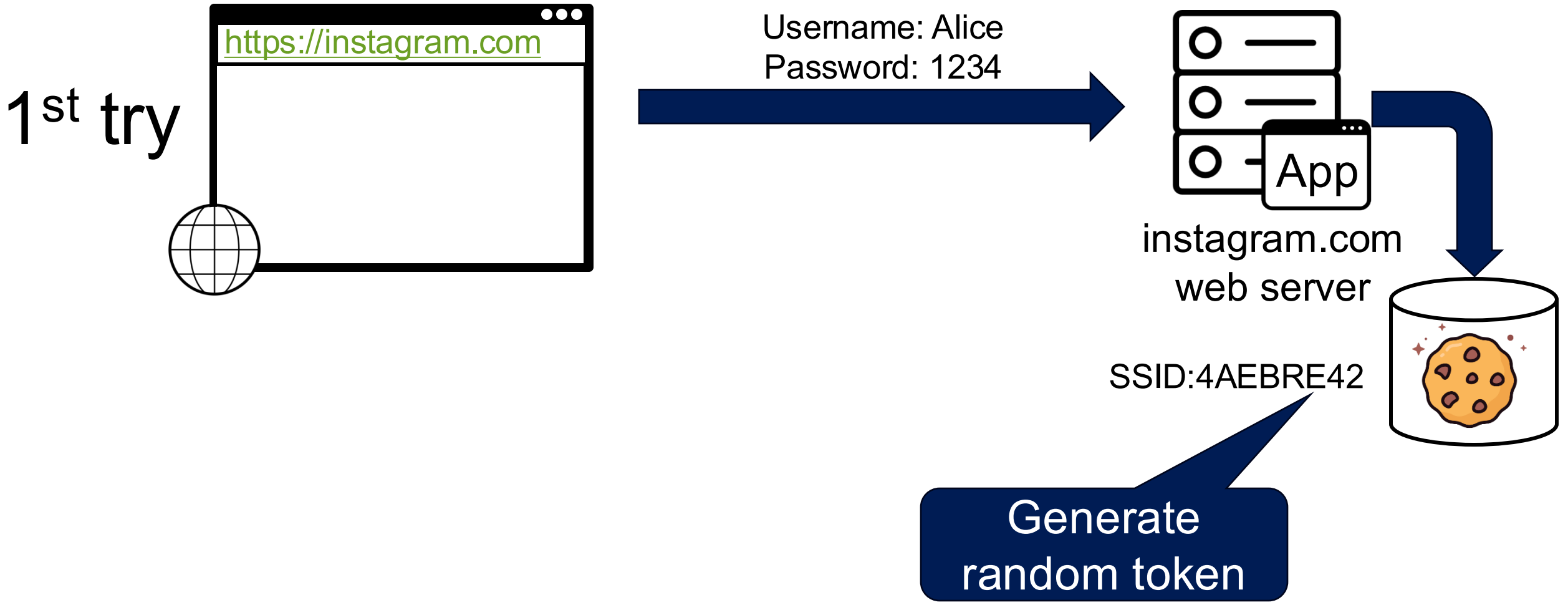
Adding State to HTTP

- Recall: no inherent state in HTTP
  - Server does not keep any state after the connection is closed
- For static content sites, no problem
  - Developing “applications” is impossible though
  - E.g., shopping cart on Amazon
- Need to introduce state in HTTP
  - in the form of “cookies”

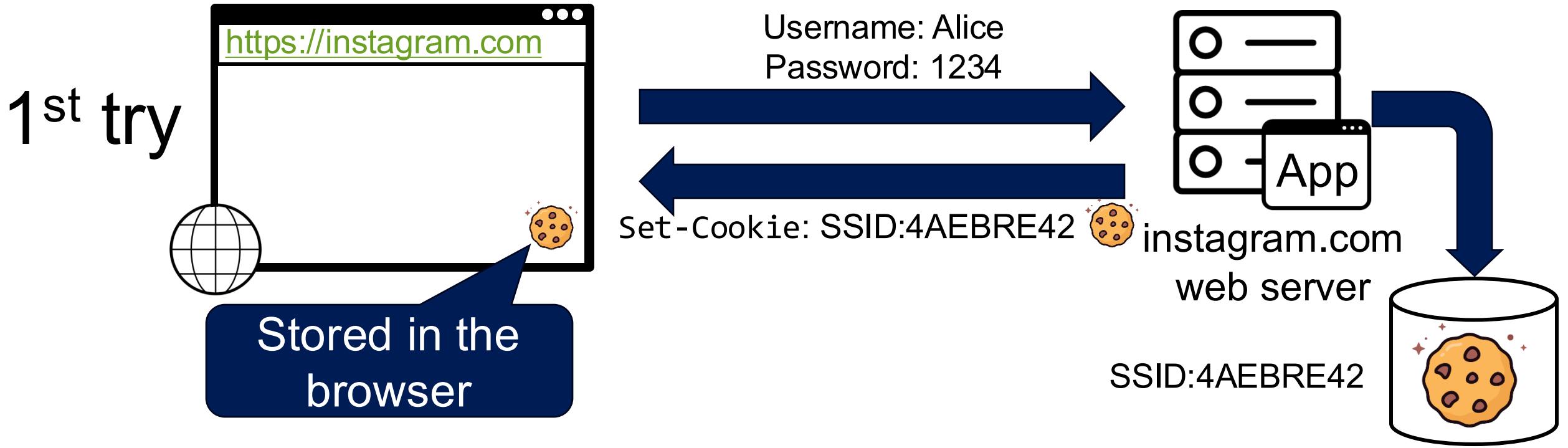
Cookie: Making HTTP Stateful

- HTTP cookie: small piece of data that a server sends to the browser, who stores it and sends it back with subsequent requests

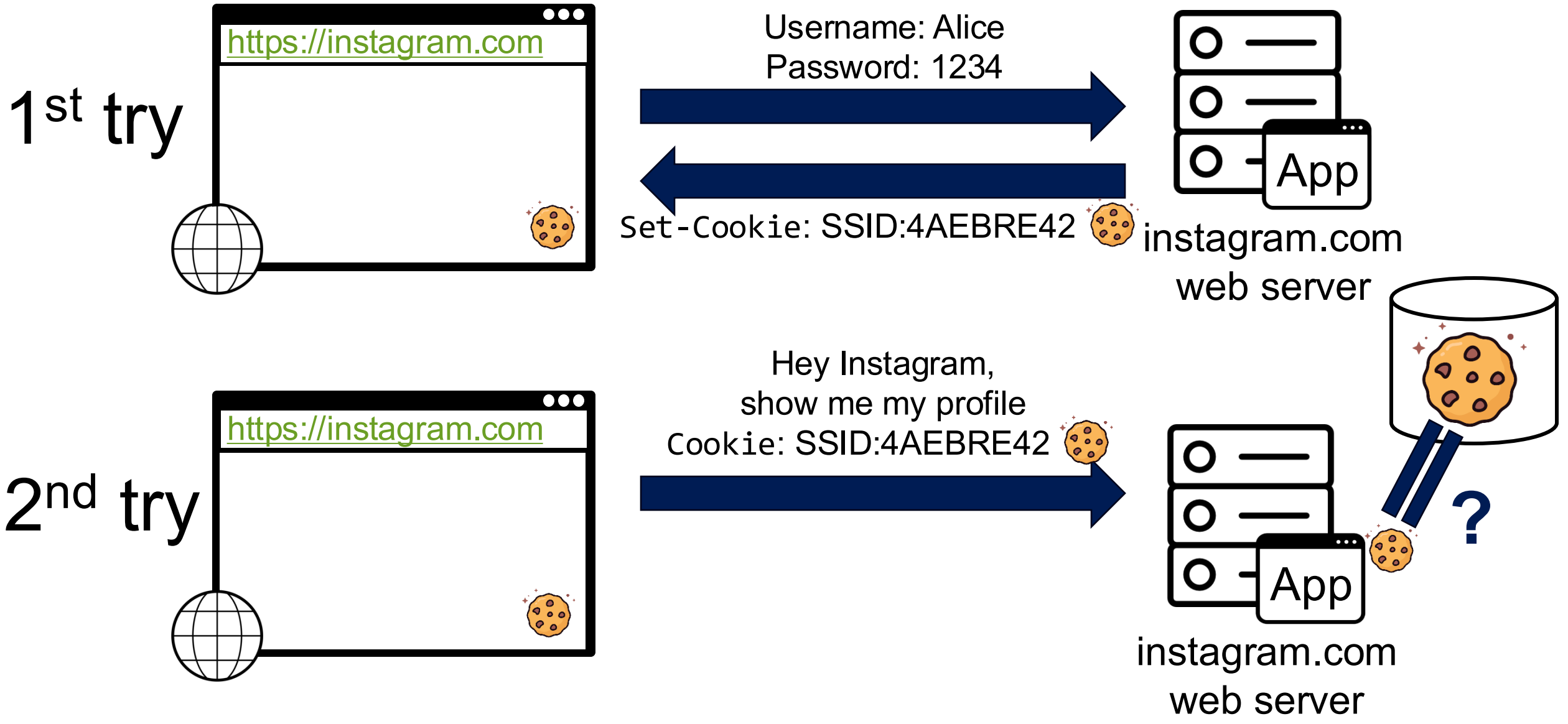
Cookie: Making HTTP Stateful



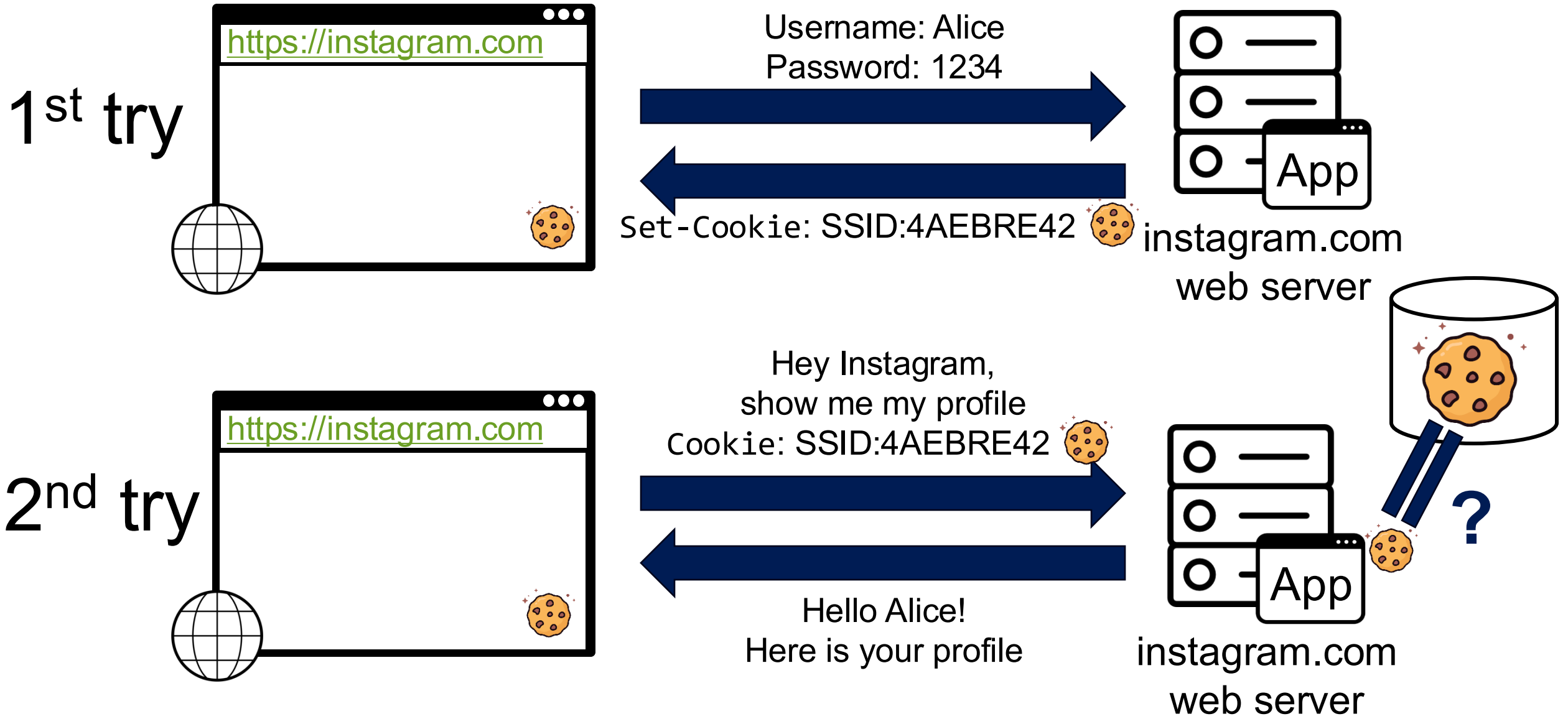
Cookie: Making HTTP Stateful



Cookie: Making HTTP Stateful



Cookie: Making HTTP Stateful



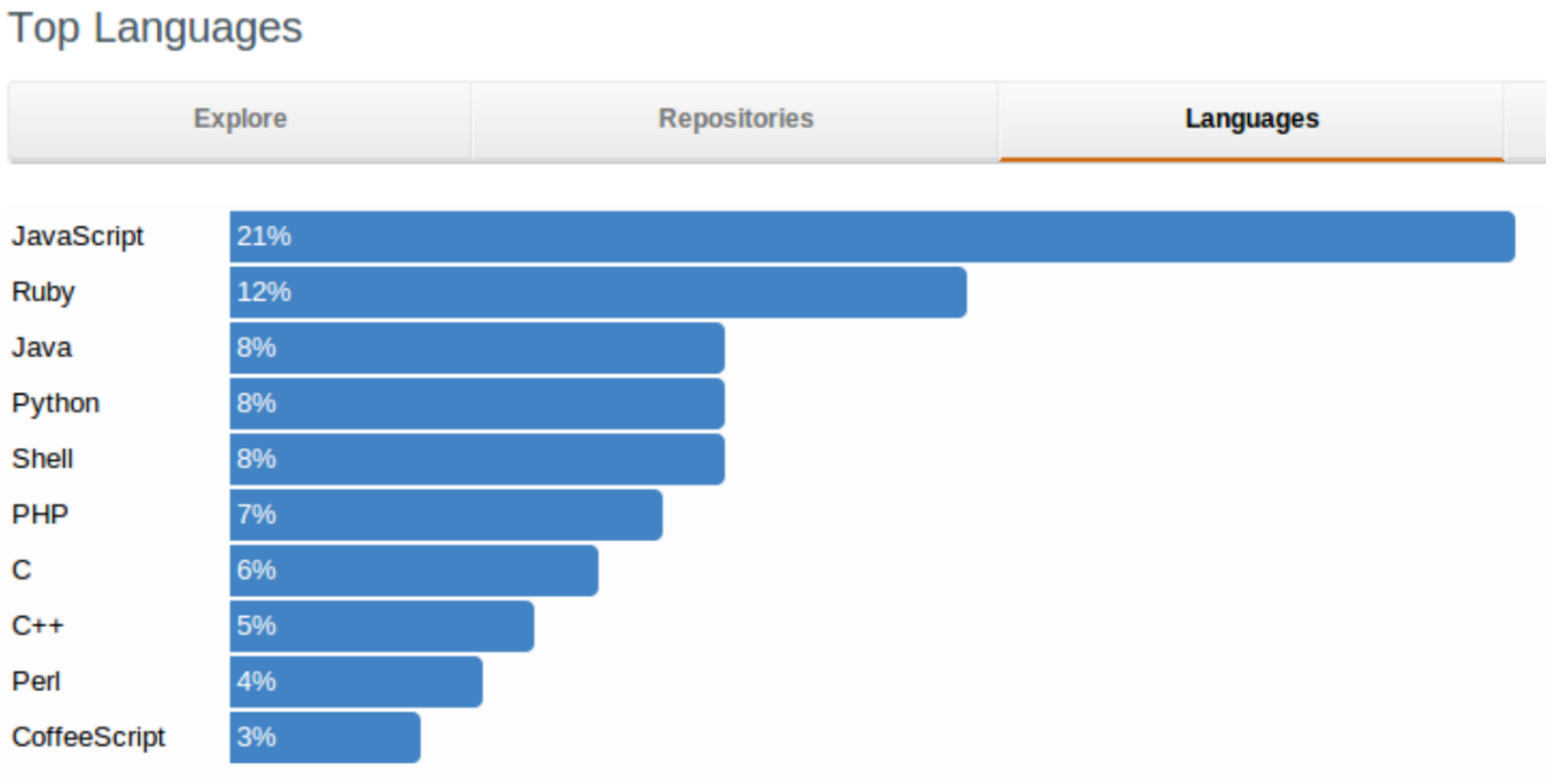
Cookie: Making HTTP Stateful

- Generate random token on first page visit
- Sent to client via `Set-Cookie` header
- Client always sends along cookies in every request to the server

- Cookies are persisted in the browser
  - Controllable by `Expires` option in cookie

JavaScript (JS)

- Most popular language in the world!



JavaScript (JS)

- Developed by Brendan Eich at Netscape
- Later standardized for browser compatibility
  - ECMAScript Edition 3 (a.k.a., JavaScript 1.5)

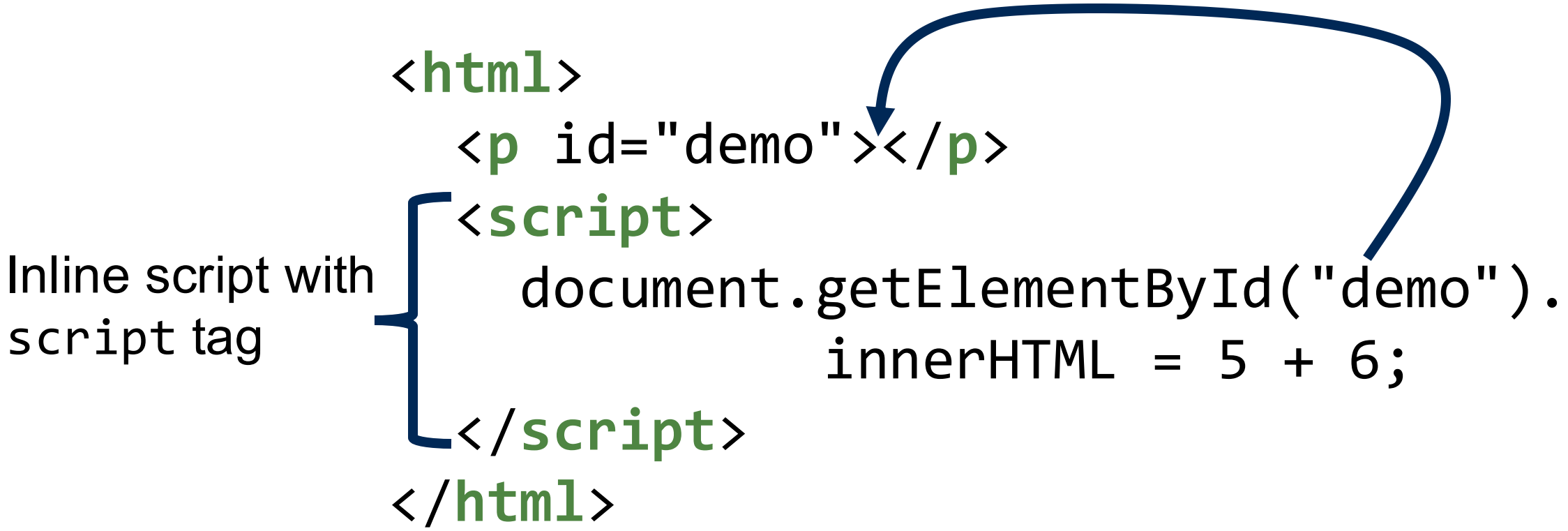
- HTML may contain JS program code to make web pages more dynamic

JS Example (1)


```
<html>
  <p id="demo"></p>
  <script>
    document.getElementById("demo").
                innerHTML = 5 + 6;
  </script>
</html>
```

https://websec-lab.github.io/courses/2025f-cse551/demo/demo1.html

JS Example (1)



https://websec-lab.github.io/courses/2025f-cse551/demo/demo1.html

JS Example (2)


```
<html>
  <button type="button" onclick="document.write(5 + 6)">
    Try it
  </button>
</html>
```

https://websec-lab.github.io/courses/2025f-cse551/demo/demo2.html

JS Example (2)

When the button is clicked, overwrite whole document with 11

```
<html>
  <button type="button" onclick="document.write(5 + 6)">
    Try it
  </button>
</html>
```

Inline script with `onclick` event handler

https://websec-lab.github.io/courses/2025f-cse551/demo/demo2.html

JS Example (3)

index.html

```
<html>
  <script src="write.js">
  </script>
</html>
```

write.js

```
document.write(5 + 6)
```

https://websec-lab.github.io/courses/2025f-cse551/demo/demo3.html

JS Example (3)

Overwrite whole document with 11

index.html

```
<html>
  <script src="write.js">
  </script>
</html>
```

write.js

```
document.write(5 + 6)
```

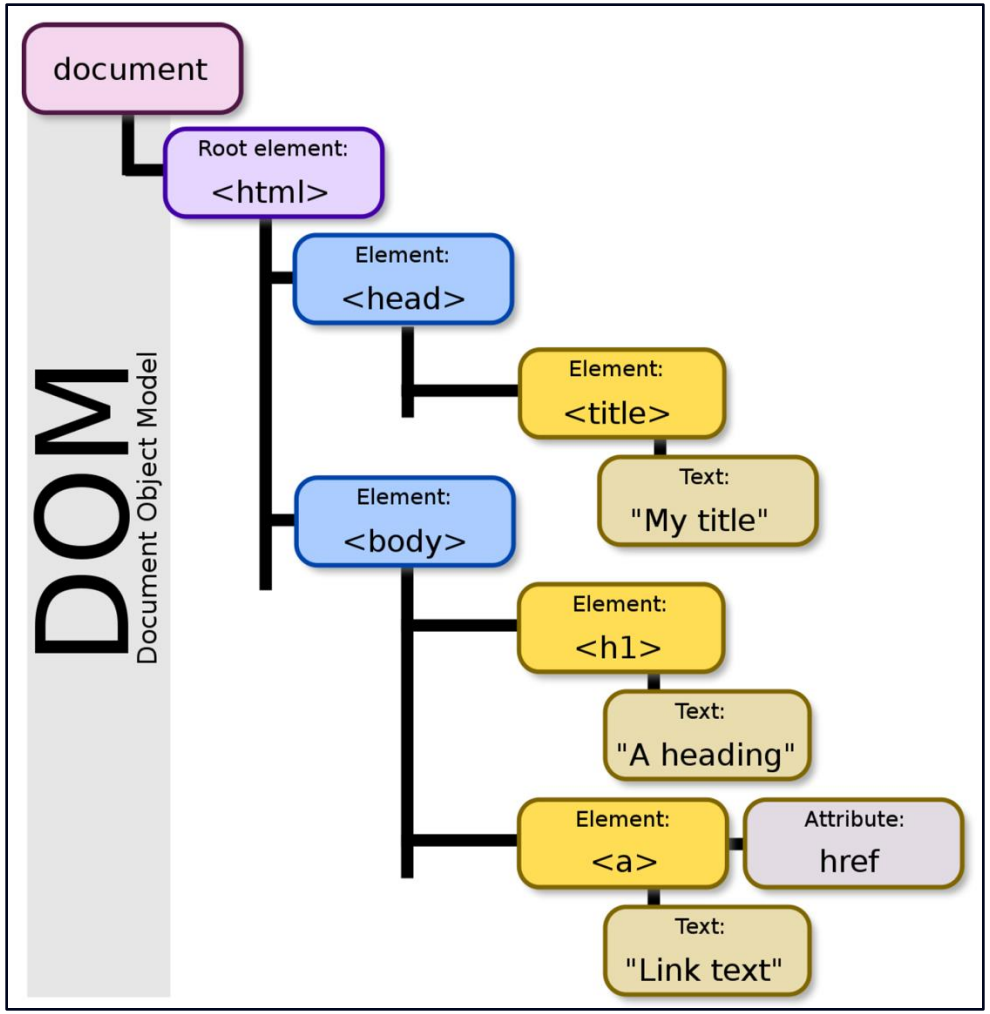
External script with `src` attribute

https://websec-lab.github.io/courses/2025f-cse551/demo/demo3.html

Document Object Model (DOM)

- An HTML document: structured data

```
<html>
  <head>
    <title>
      My title
    </title>
  </head>
  <body>
    <h1>A heading</h1>
    <a href="cse551.com">Link text</a>
  </body>
<body>
```



DOM and JS APIs

- Exposed to JavaScript through global objects
  - `document`: Access to the document (e.g., cookies, head/body)
  - `navigator`: Information about the browser (e.g., UA, plugins)
  - `screen`: Information about the screen (e.g., dimension, color depth)
  - `location`: Access to the URL (read and modify)
  - `history`: Navigation

Changing HTML DOM using JS

- JavaScript can change all the HTML DOM components in the page!

- using several APIs
  - `createElement(elementName)`
  - `createTextNode(text)`
  - `appendChild(newChild)`
  - `removeChild(node)`

Changing HTML DOM using JS (Example)


```
<html>
  <body>
    <ul id=“t1”>
      <li>Item 1</li>
    </ul>
…
  </body>
</html>
```

- Item 1

Changing HTML DOM using JS (Example)


```
<html>
  <body>
    <ul id="t1">
      <li>Item 1</li>
    </ul>
…
  </body>
</html>
```

```
<script>
  var list = document.getElementById('t1')
  var newitem = document.createElement('li')
  var newtext = document.createTextNode('Item 2')
  list.appendChild(newitem)
  newitem.appendChild(newtext)
</script>
```

- Item 1

Changing HTML DOM using JS (Example)

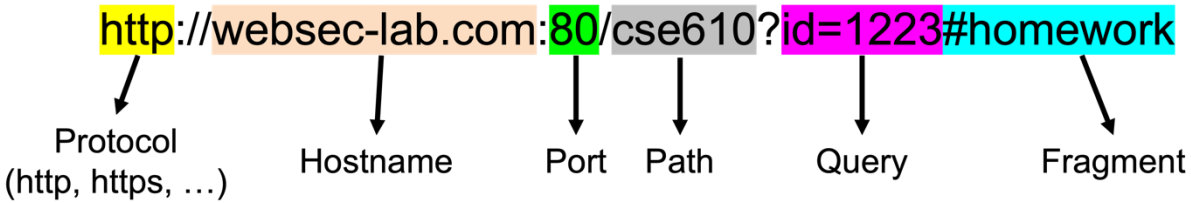

```
<html>
  <body>
    <ul id="t1">
      <li>Item 1</li>
    </ul>
…
  </body>
</html>
```

```
<script>
  var list = document.getElementById('t1')
  var newitem = document.createElement('li')
  var newtext = document.createTextNode('Item 2')
  list.appendChild(newitem)
  newitem.appendChild(newtext)
</script>
```

- Item 1
- Item 2

Accessing HTML DOM using JS (Example)

- `location.protocol`: protocol
- `location.hostname`: only HTTP host
- `location.port`: only the port
- `location.pathname`: path



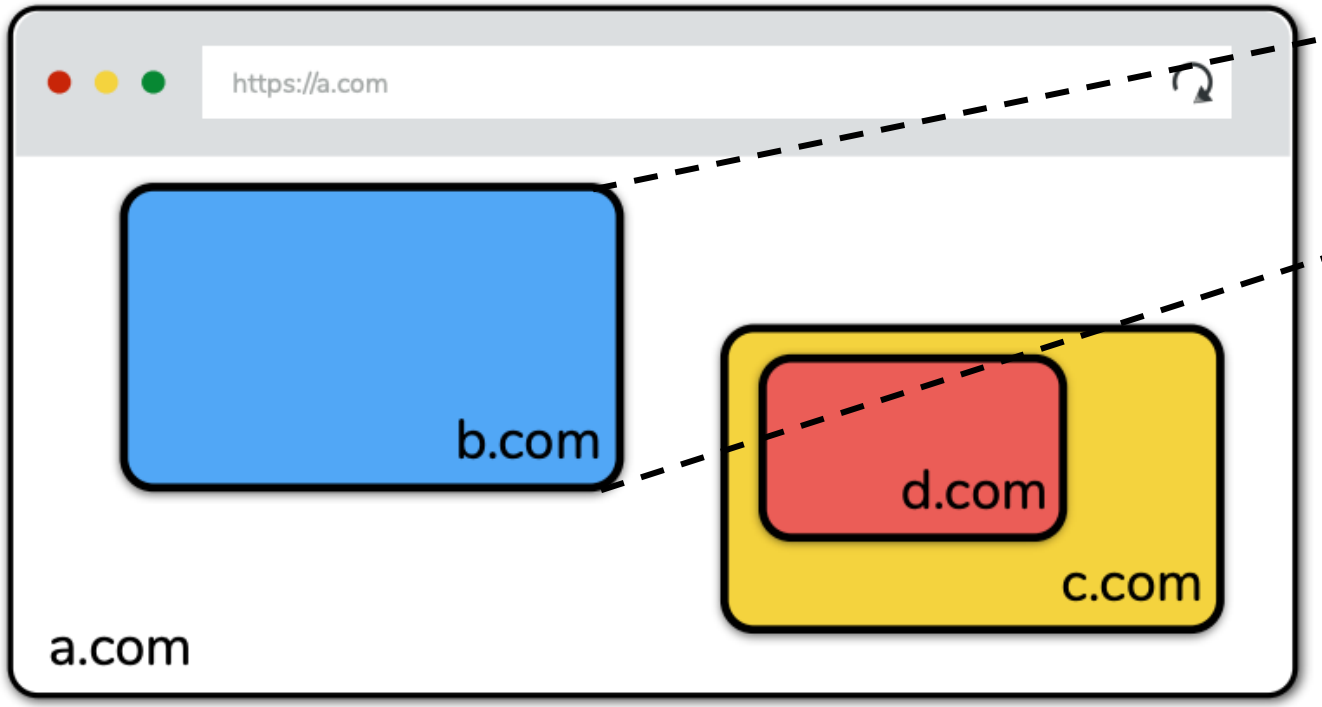
- We can display all cookies for current document by `alert(document.cookie)`

Basic Browser Execution Model

- Each browser window…
  - Loads content
  - Parses HTML and runs JavaScript
  - Fetches sub-resources (e.g., images, CSS, Javascript)
  - Respond to events like `onClick`, `onMouseover`, `onLoad`, `setTimeout`

Nested Execution Model

- Windows may contain frames from different sources
  - **Frame**: rigid visible division
  - **iFrame**: floating inline frame




```
<iframe src="b.com">
</iframe>
```

Nested Execution Model

- Windows may contain frames from different sources
  - **Frame**: rigid visible division
  - **iFrame**: floating inline frame

- Why use frames?
  - Delegate screen area to content from another source
  - Browser provides isolation based on frames
  - Parent may work even if frame is broken

Web Threat Models

- **Network attacker**
- **Remote attacker**
- **Web attacker**

Web Threat Models

- **Network attacker**: resides somewhere in the communication link between client and server
  - Passive: evasdropping
  - Active: modification of messages, replay…

- **Remote attacker**

- **Web attacker**

Web Threat Models

- **Network attacker**: resides somewhere in the communication link between client and server
  - Passive: evasdropping
  - Active: modification of messages, replay…

- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

- **Web attacker**

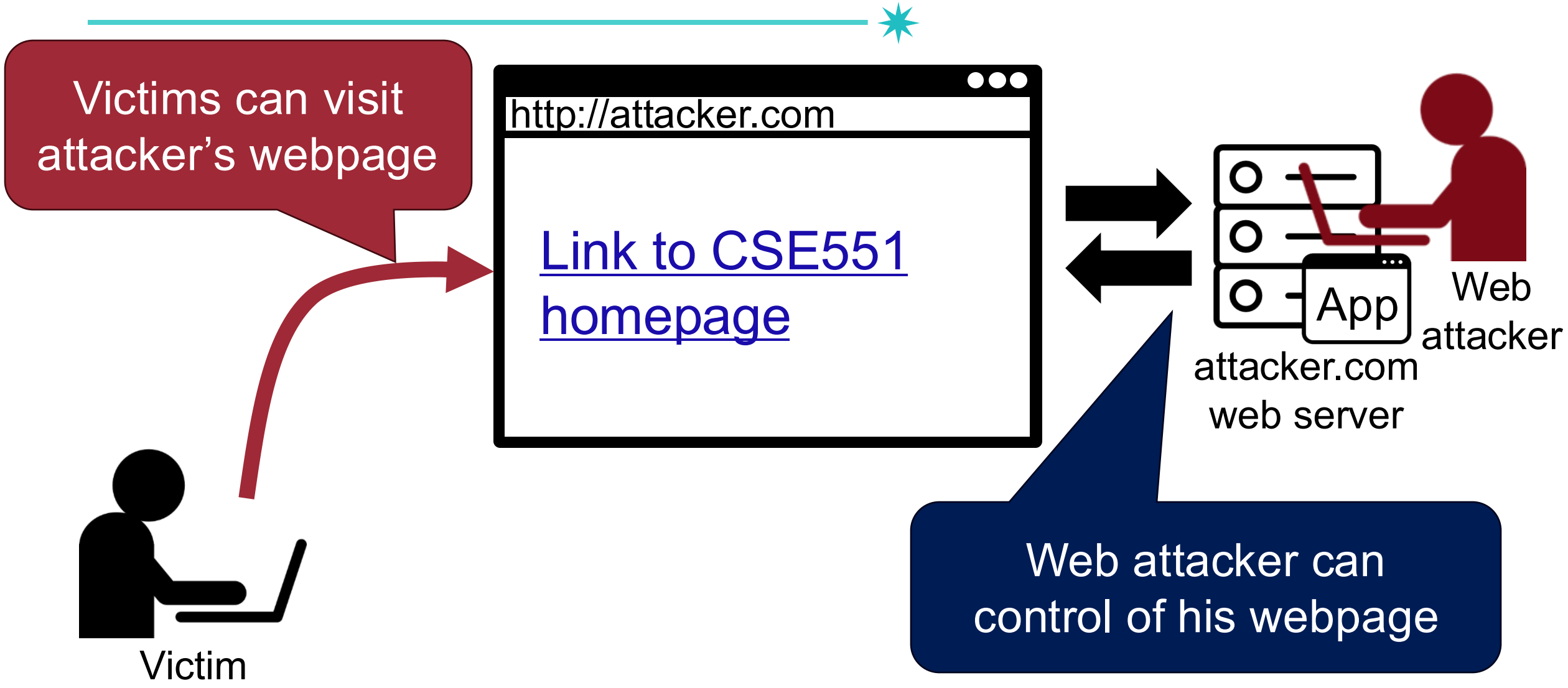
Web Threat Models

- **Network attacker**: resides somewhere in the communication link between client and server
  - Passive: evasdropping
  - Active: modification of messages, replay…

- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com

Web Attacker

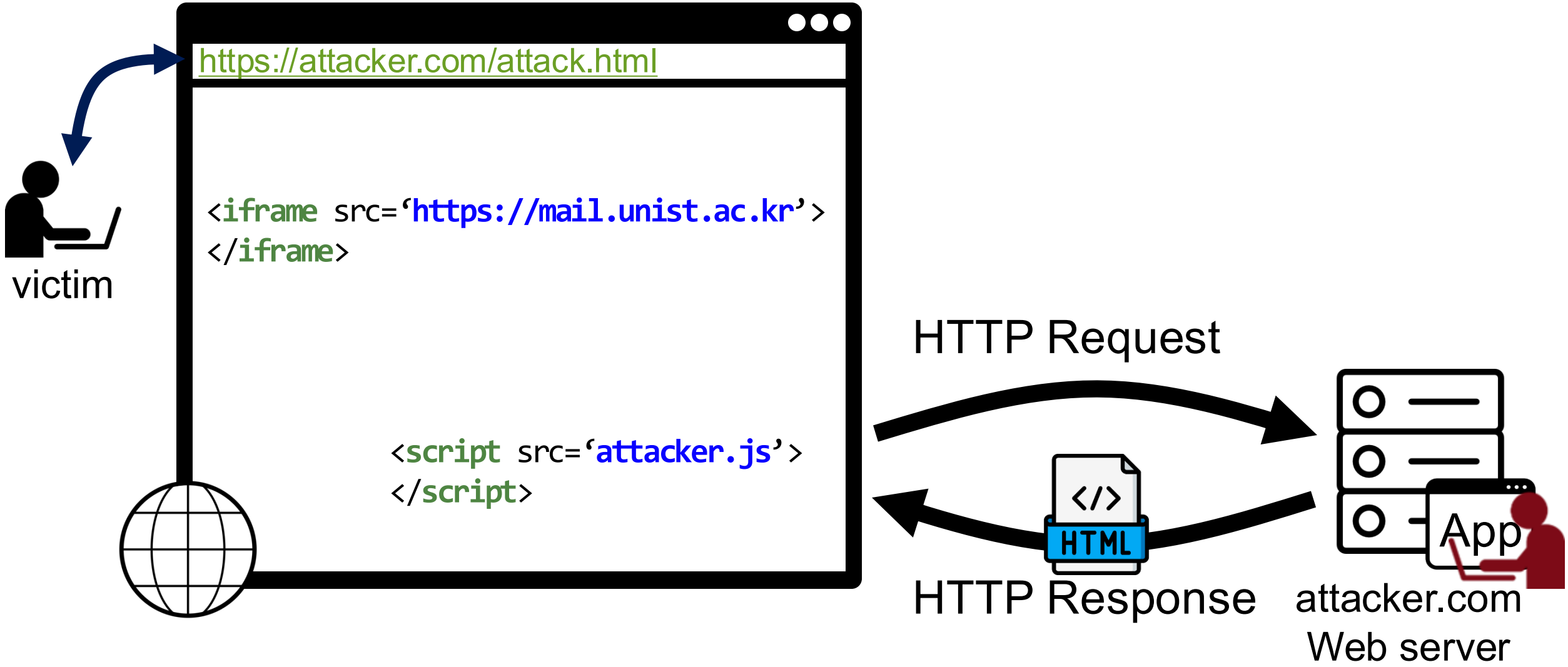


Question

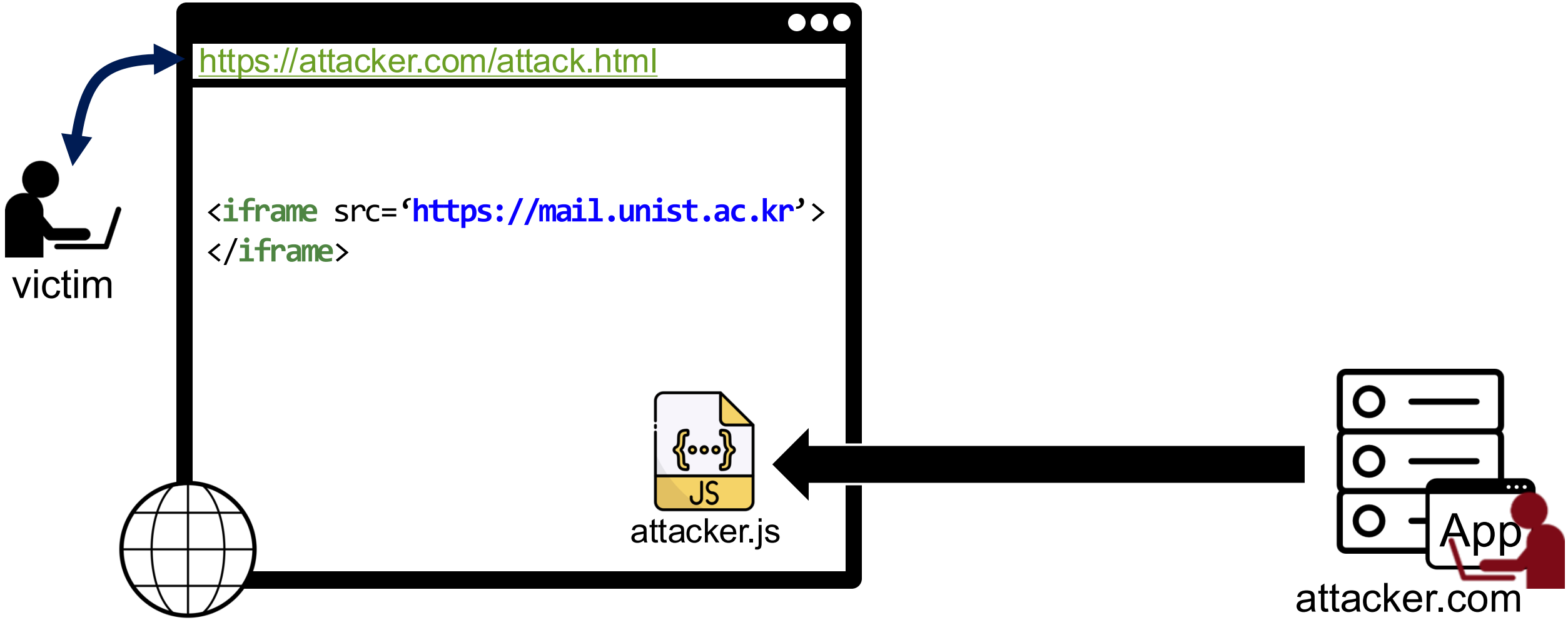
- Is the ***web attacker*** has a control on the victim's referrer header?

Client-side Security Basics

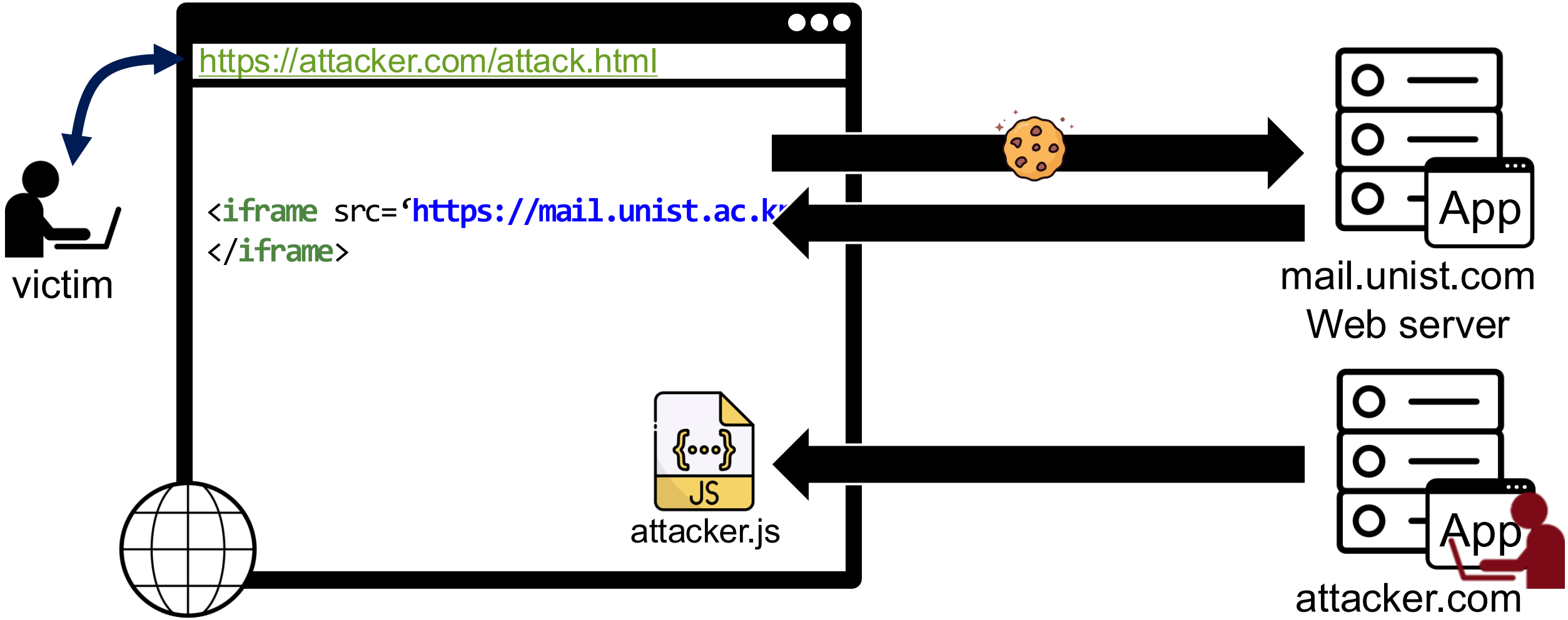
Motivation of the Client-side Security



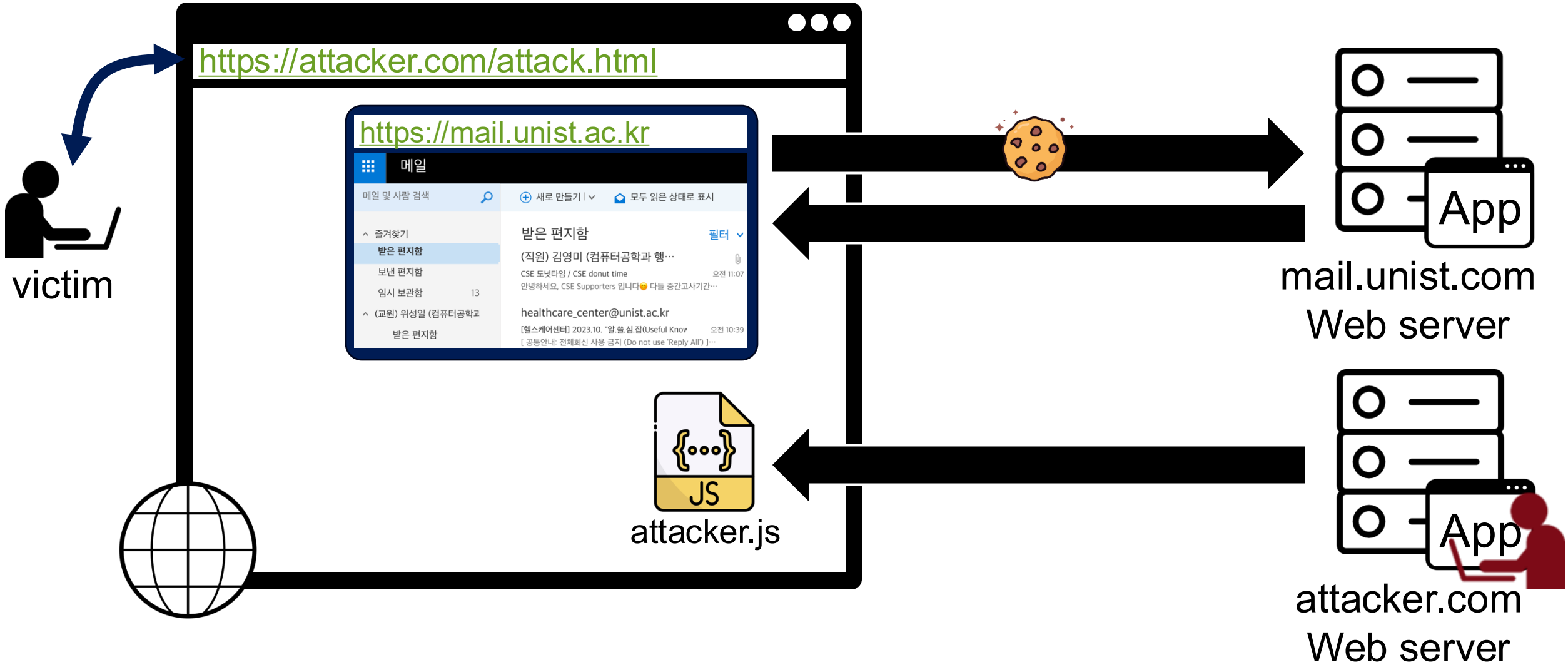
Motivation of the Client-side Security



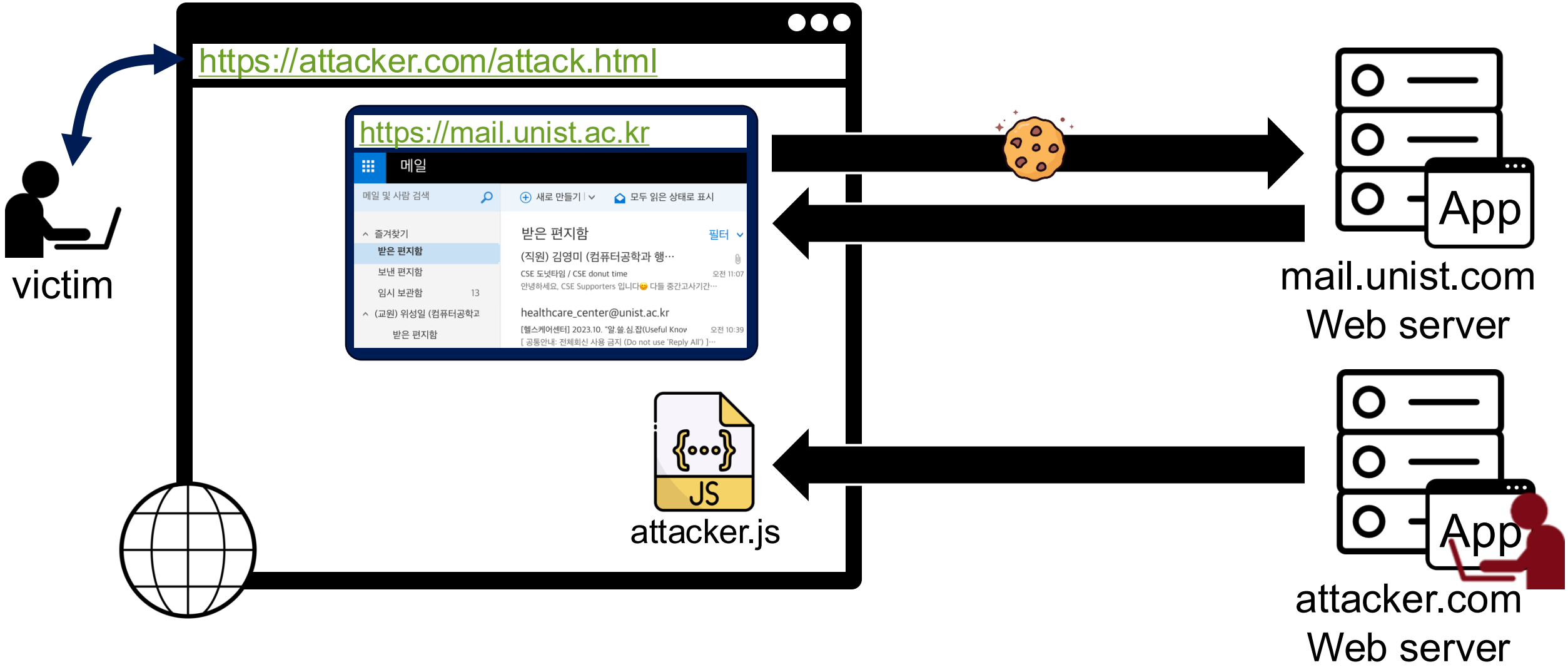
Motivation of the Client-side Security



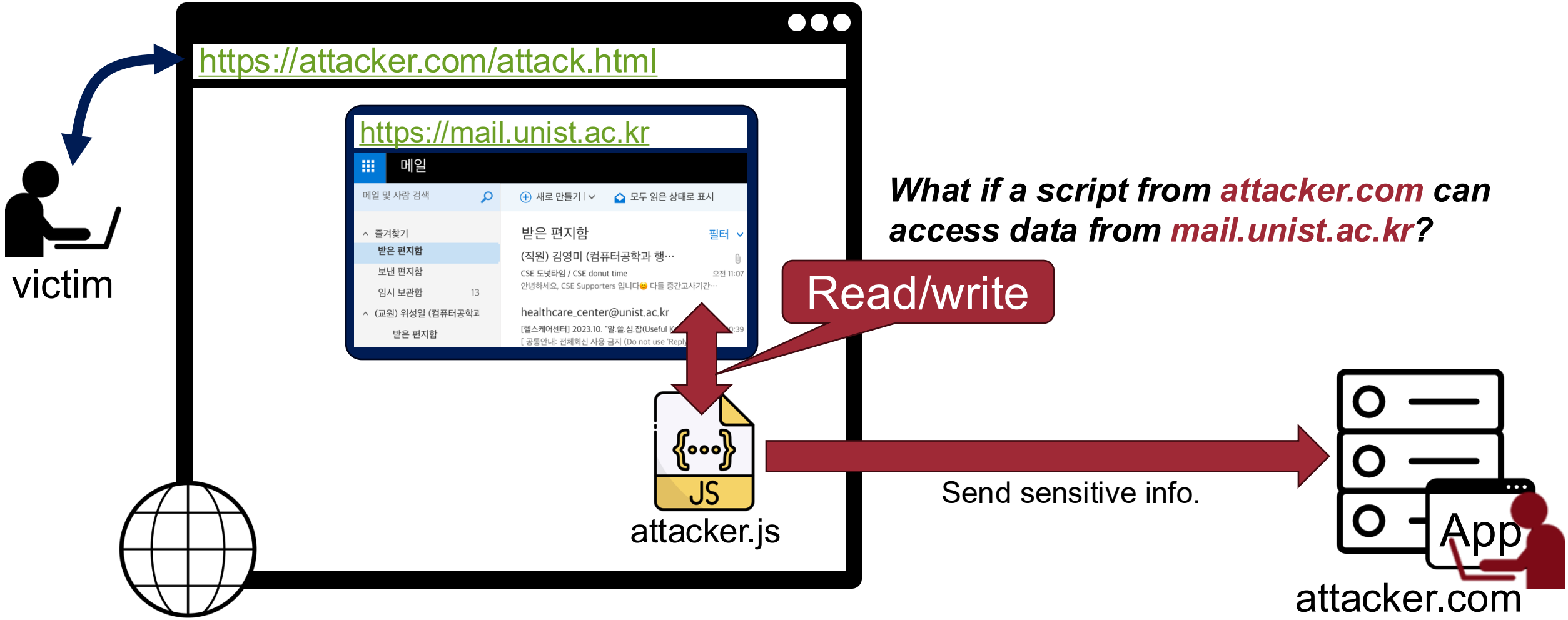
Motivation of the Client-side Security



Motivation of the Client-side Security



A World Without Separation between Sites

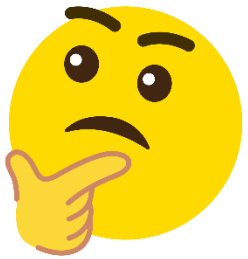


A World Without Separation between Sites

What if a script from attacker.com can access data from mail.unist.ac.kr?

It would be able to read your emails, private messages, authentication session cookies

Motivation of the Client-side Security

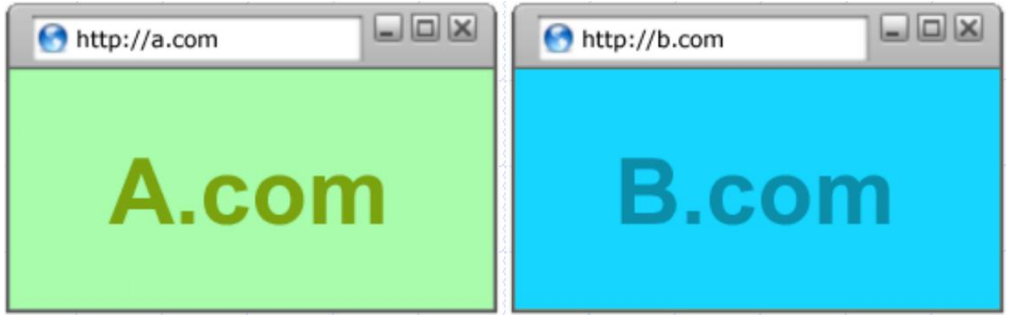


How can we prevent such malicious behaviors?

Policy Goals

- Safe to visit an evil website

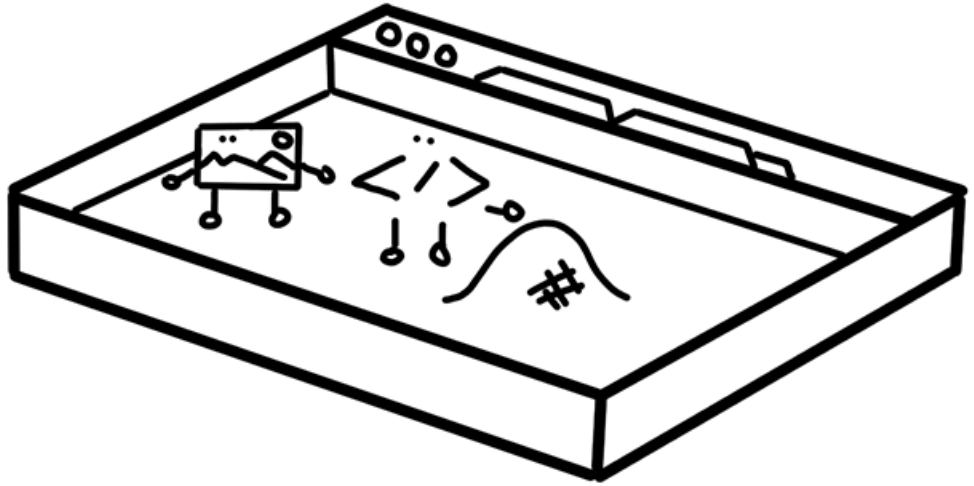
- Safe to visit two pages at the same time
  - Address bar distinguishes them



- Allow safe delegation

Browser Sandbox

- No direct file access, limited access to OS
- Goal: Safely execute JavaScript code provided by a remote website
  - Isolated process when HTML rendering and JavaScript execution



Browser Sandbox Escaping Vulnerabilities

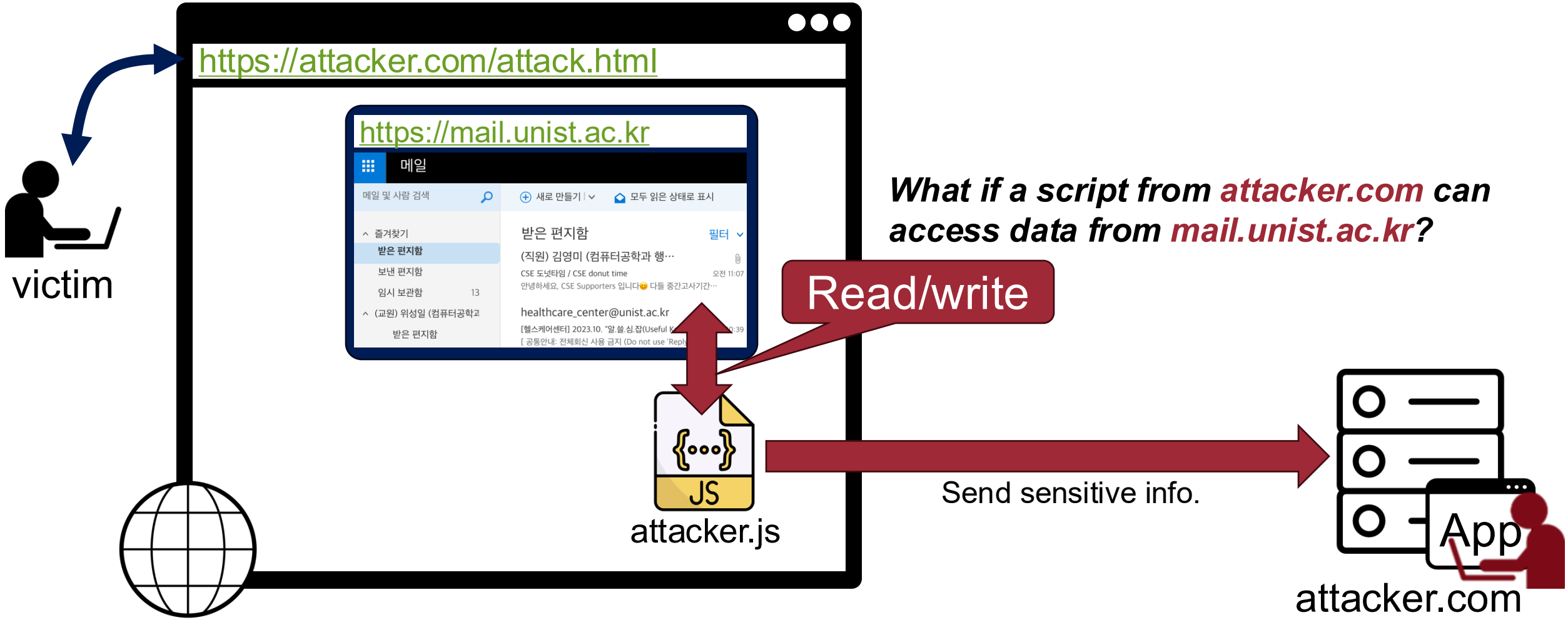
- Related to memory-level vulnerabilities, including Use-After-Free (UAF), heap overflow,…

- CVE-2013-6632
- CVE-2014-3188
- CVE-2015-6767
- CVE-2019-5850

Same Origin Policy (SOP)

- One of the browser sandboxing mechanism
- The basic security model enforced in the browser

A World Without Separation between Sites

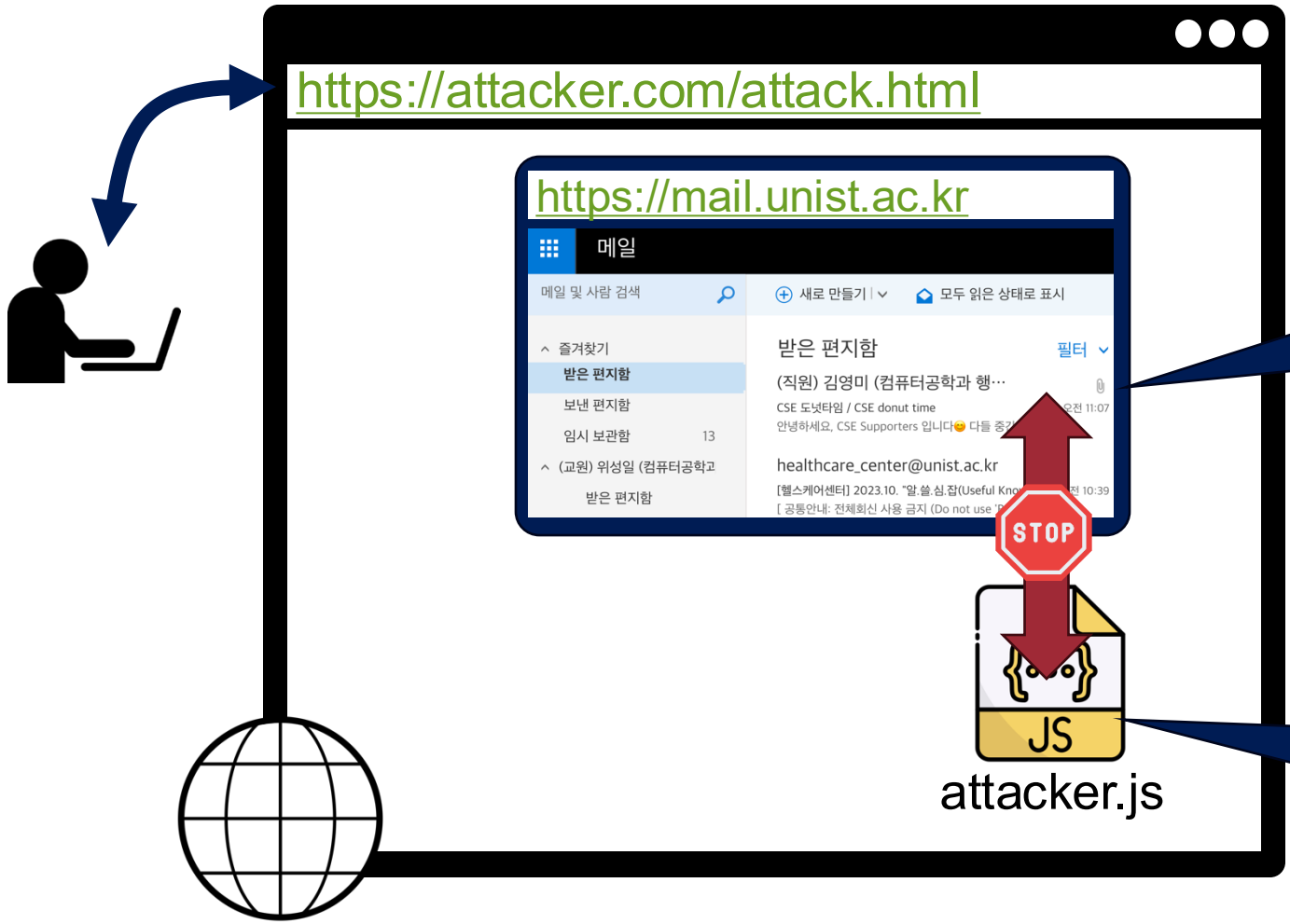


Same Origin Policy (SOP)

- Restricts scripts on one origin from accessing data from another origin

Same Origin Policy (SOP)

- Restricts scripts on one origin from accessing data from another origin



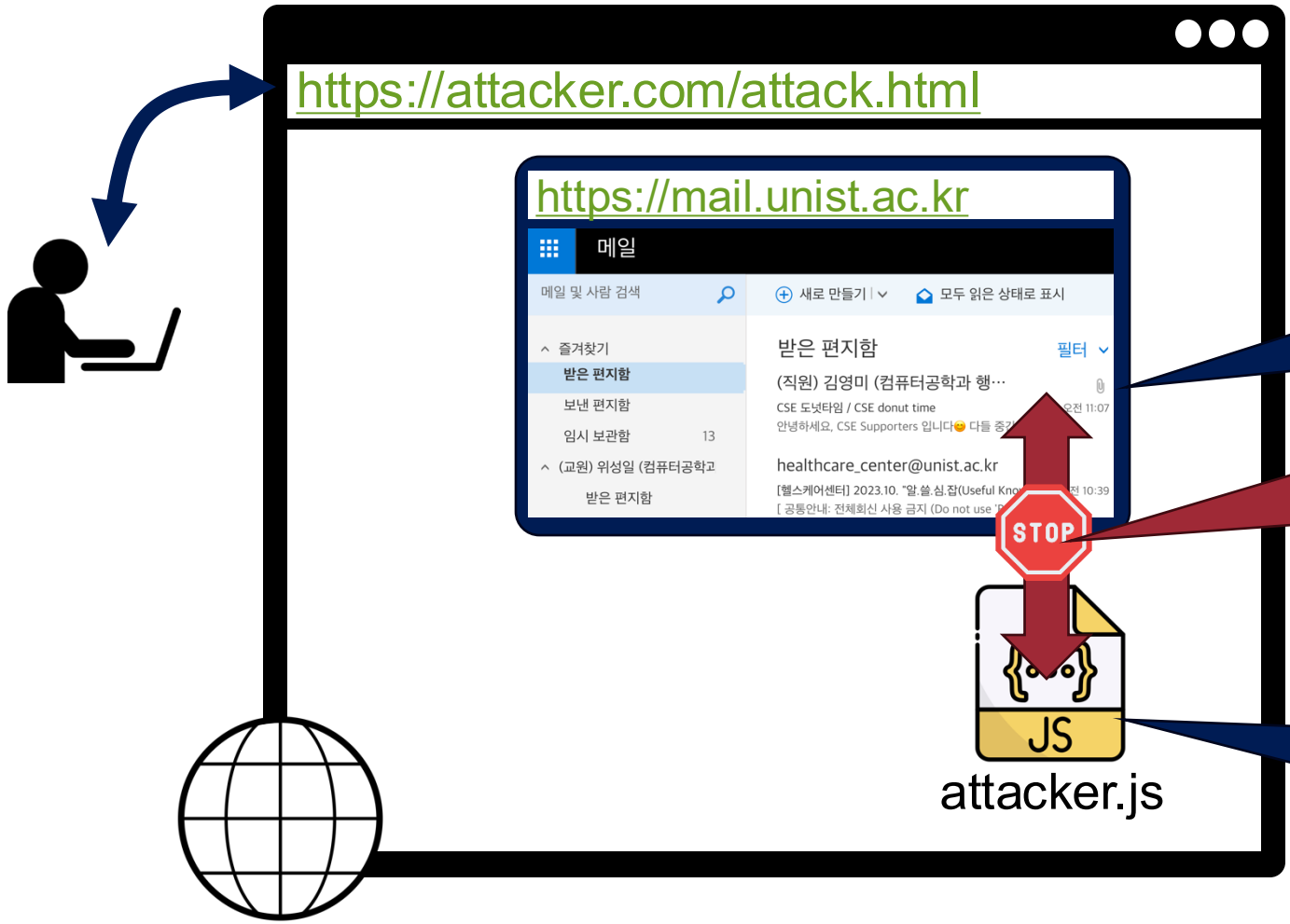
Any resource has its own origin

Resources located at the *https://mail.unist.ac.kr* origin

JavaScript runs with a *https://attacker.com* origin

Same Origin Policy (SOP)

- Restricts scripts on one origin from accessing data from another origin



Any resource has its own origin

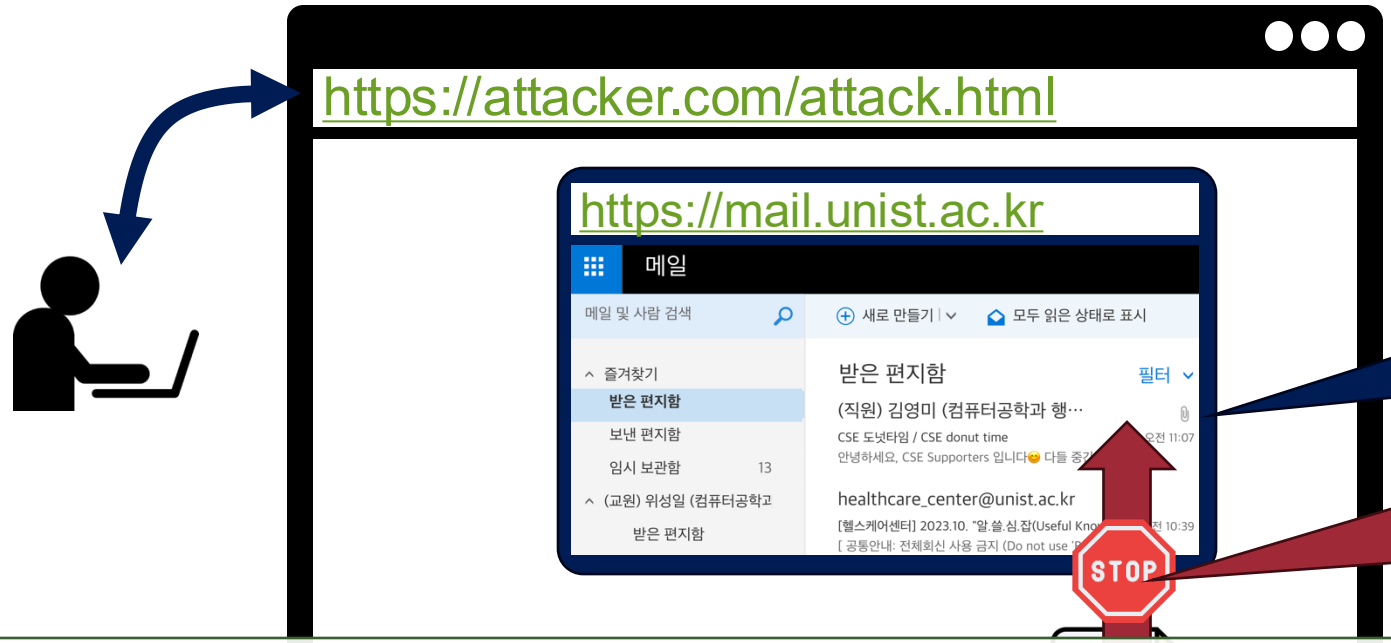
Resources located at the *https://mail.unist.ac.kr* origin

A JS runs with an origin cannot access other origin resources

JavaScript runs with a *https://attacker.com* origin

Same Origin Policy (SOP)

- Restricts scripts on one origin from accessing data from another origin



Any resource has its own origin

Resources located at the *https://mail.unist.ac.kr* origin

A JS runs with an origin cannot access other origin resources

Uncaught DOMException: Permission denied to access property "document" on cross-origin object

Same Origin Policy (SOP)

- Restricts scripts on one origin from accessing data from another origin

- The basic security model enforced in the browser

- Basic access control mechanism for web browsers
  - All resources such as DOM, cookies, JavaScript has their own origin
  - SOP allows a subject to access only the objects from the same origin

What is an Origin?

- **Origin = Protocol + Domain Name + Port**
  - `origin = protocol://domain:port`
- Any resource has its own origin (owner)

- Two URLs have the same origin if the **protocol**, **domain name** (not subdomains), **port** are the same for both URLs
  - All three must be equal origin to be considered the same

Quiz – Same Origin?

- Consider this URL:

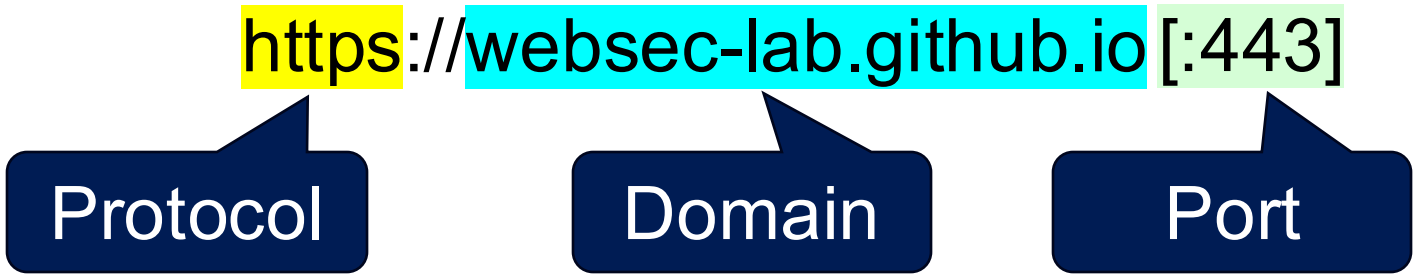
https://websec-lab.github.io

Origin = Protocol + Domain Name + Port

Idx	URL	Same Origin? ✓ (yes) **or** ✗ (No)
1	http://websec-lab.github.io	
2	https://www.websec-lab.github.io	
3	https://websec-lab.github.io:443	
4	https://websec-lab.github.io:8081	
5	https://websec-lab.github.io/cse610	

Quiz – Same Origin?

- Consider this URL:



Origin = Protocol + Domain Name + Port

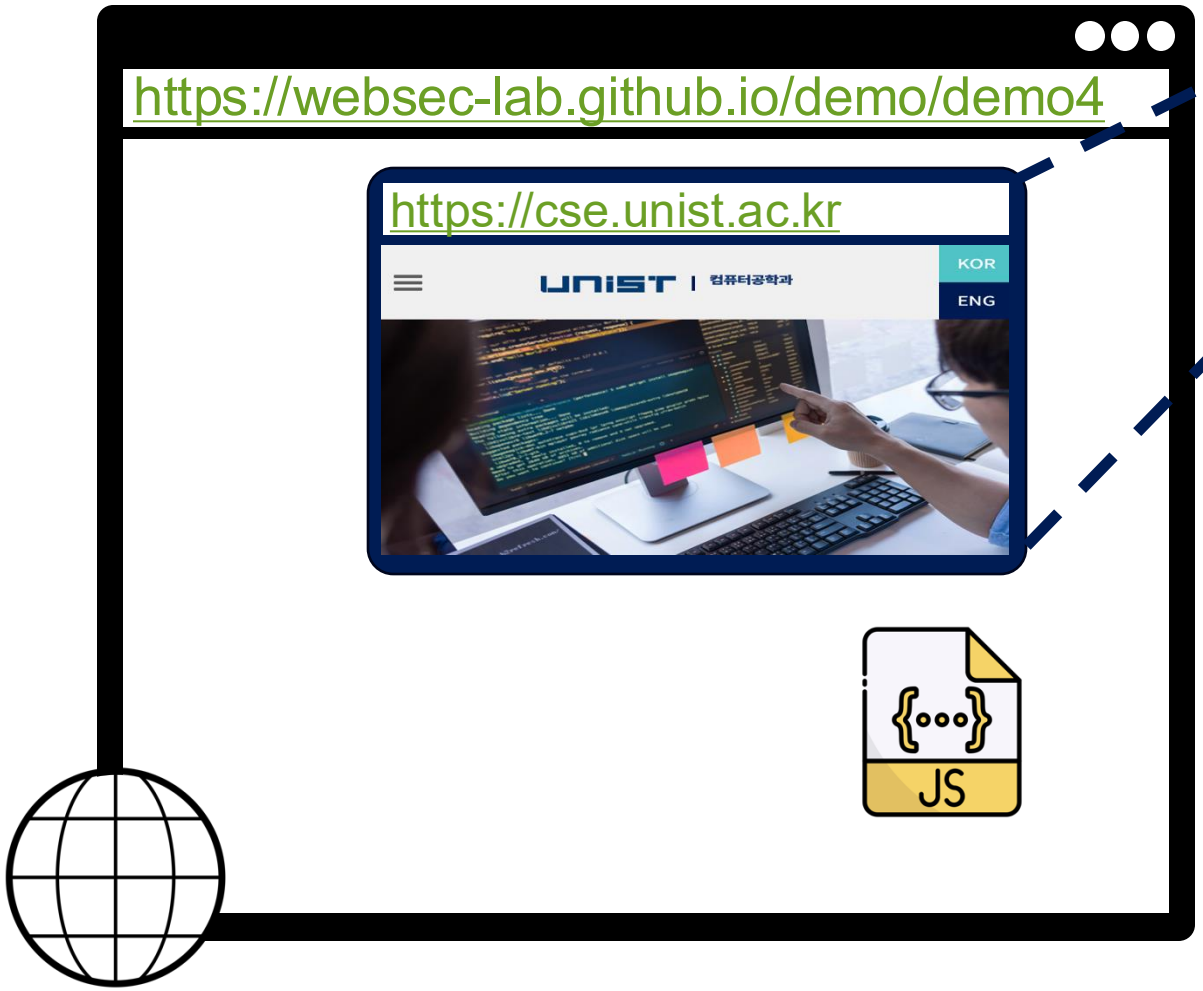
Idx	URL	Same Origin? ✓ (yes) or ✗ (No)
1	http://websec-lab.github.io[:80]	✗ (different protocol and different port)
2	https://www.websec-lab.github.io	✗ (different domain)
3	https://websec-lab.github.io:443	✓ (same protocol, domain, and port)
4	https://websec-lab.github.io:8081	✗ (different port)
5	https://websec-lab.github.io/cse610	✓ (same protocol, domain, and port)

What is an Origin?

- **Origin = Protocol + Domain Name + Port**
  - `origin = protocol://domain:port`
- Any resource has its own origin (owner)

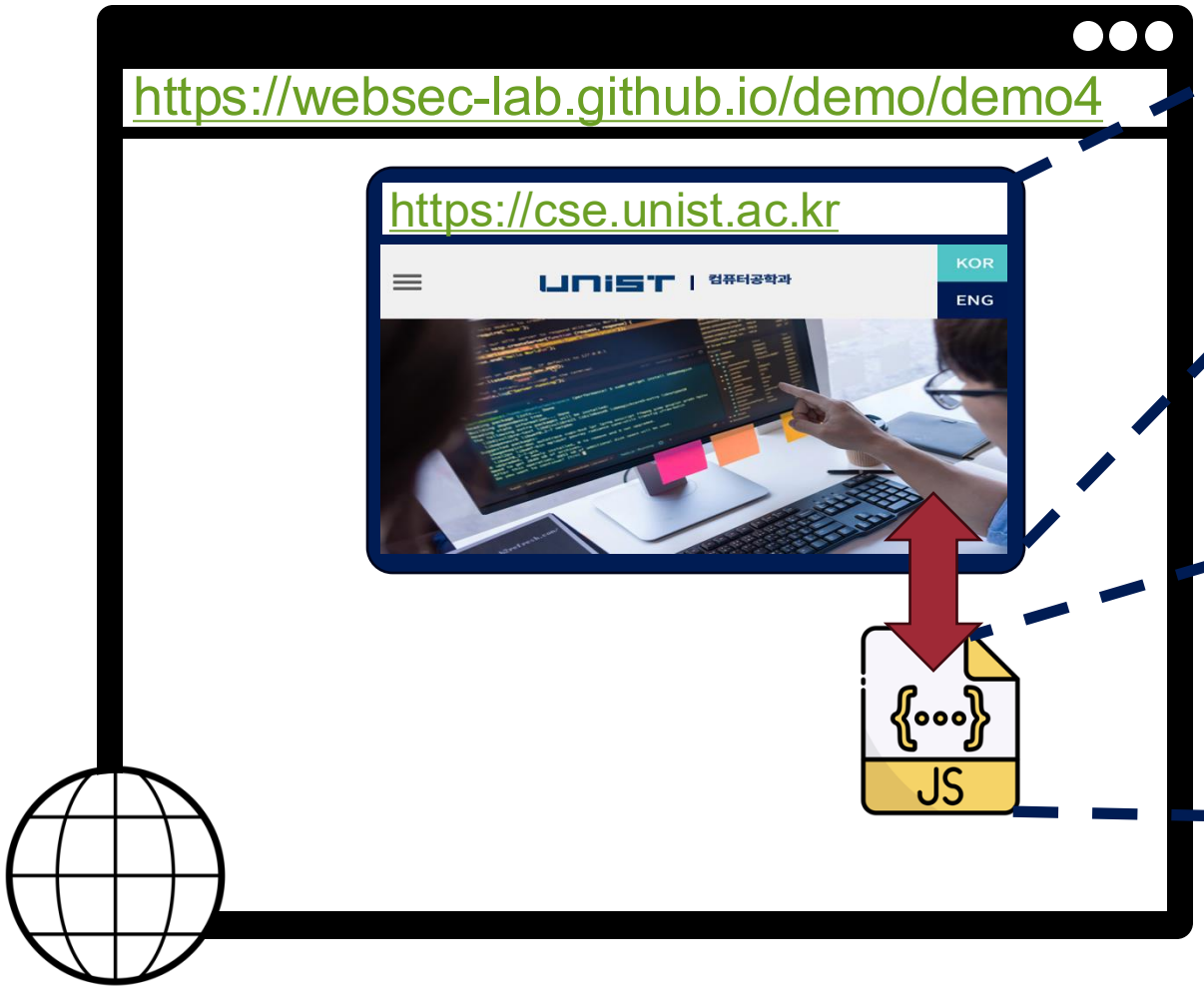
- Two URLs have the same origin if the **protocol**, **domain name** (not subdomains), **port** are the same for both URLs
  - All three must be equal origin to be considered the same

Demo: Same Origin Policy




```
<iframe id="UNIST_CSE”
        src=https://cse.unist.ac.kr/>
</iframe>
```

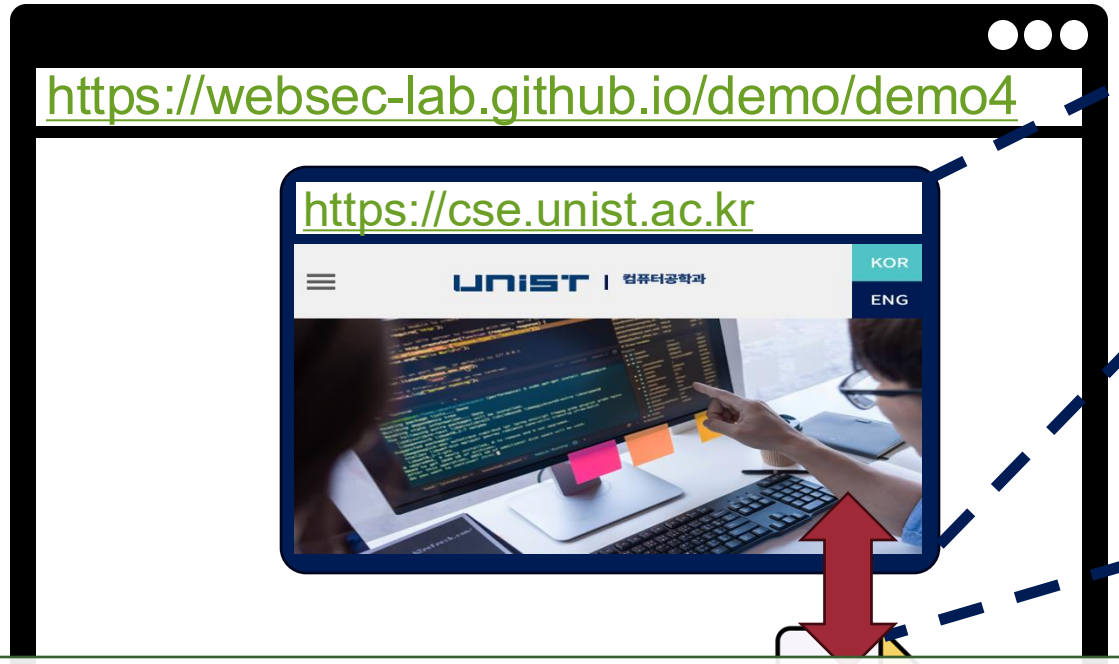
Demo: Same Origin Policy




```
<iframe id="UNIST_CSE”
        src=https://cse.unist.ac.kr/>
</iframe>
```

```
cookie =
   document.getElementById(‘UNIST_CSE’).
         contentWindow.document.cookie;
console.log(cookie)
```

Demo: Same Origin Policy




```
<iframe id="UNIST_CSE”
        src=https://cse.unist.ac.kr/>
</iframe>
```

```
cookie =
   document.getElementById(‘UNIST_CSE’).
            contentWindow.document.cookie;
console.log(cookie)
```

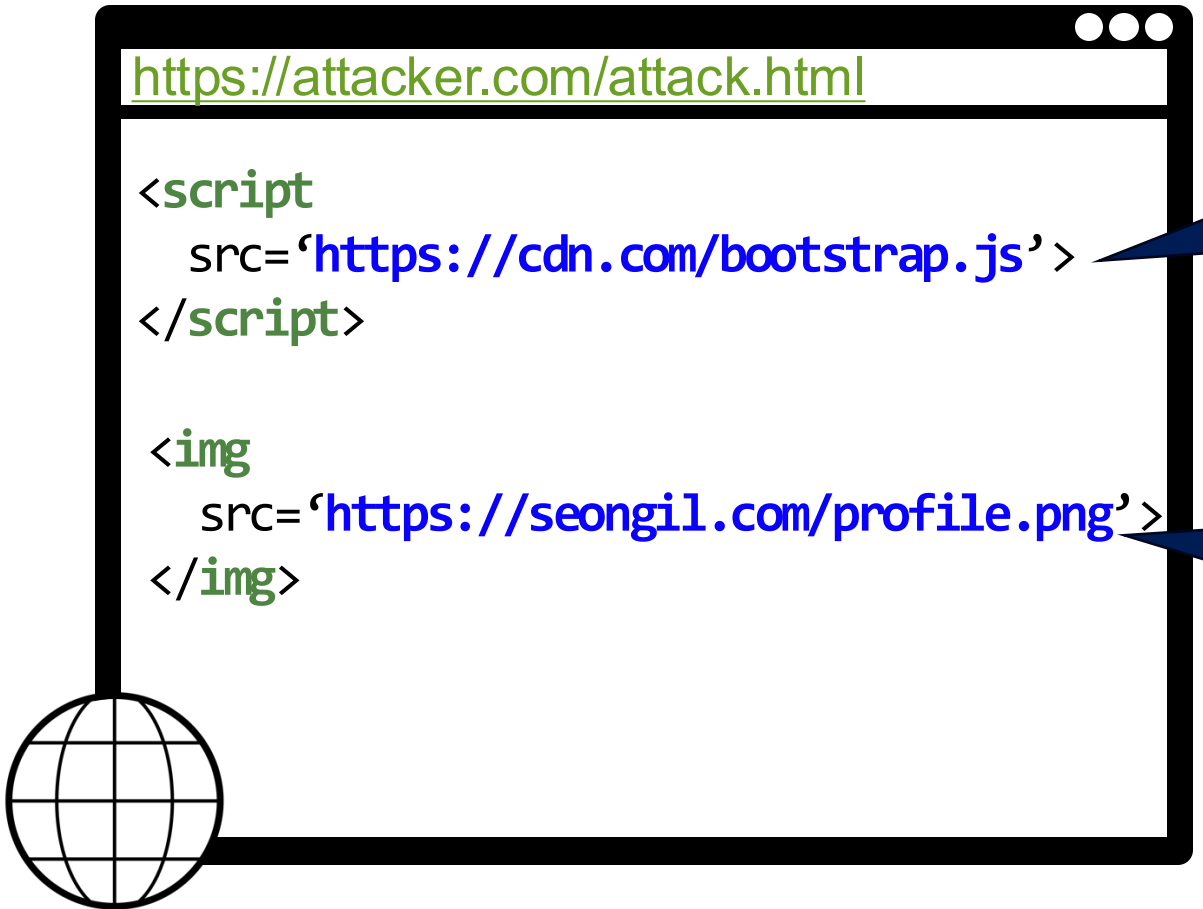
Uncaught DOMException: Blocked a frame with origin "https://websec-lab.github.io" from accessing a cross-origin frame

DEMO

https://websec-lab.github.io/courses/2025f-cse467/demo/demo4.html

For Your Information…

- **Cross-origin loading** of page resources is generally permitted
  - E.g., the SOP allows embedding of external resources via HTML tags (e.g., <img>, <video>, <script>, ...)



The origin of the loaded script is https://attacker.com

The origin of the loaded image is https://attacker.com

Analogy

- **Operating system**
  - Primitives (Resources)
    - System calls
    - Processes
    - Disk
  - Principals: Users
    - Discretionary access control
  - Vulnerabilities
    - Buffer overflow
    - Root exploit

- **Web browser**
  - Primitives (Resources)
    - Document object model
    - Frames
    - Cookies / localStorage
  - Principals: “Origins”
    - Mandatory access control
  - Vulnerabilities
    - Cross-site scripting
    - Cross-site request forgery
    - Cache history attacks
    - …

If I need to communicate with other websites, what methods should be used?

Cross-Origin Resource Sharing (CORS)

PostMessage (PM)

Question?