

# CSE610: Web Programming & Security

## 3. Client-side Security

Seongil Wi

# Notice: Paper Presentation

---



- Paper list distribution: **3/5 (Today!), 6 PM**
- Selection of papers for presentation will be on a first-come, first-served basis (선착순)
- If I send you the Google Sheets link via email, you should fill in your names next to the desired papers!

# Notice: Term Project

---



- 1~2 persons for one team
- The topics must be related to the web security/web-related security
- Submit your proposal by **3/15, 11:59 PM**

# Proposal Submission Guidelines

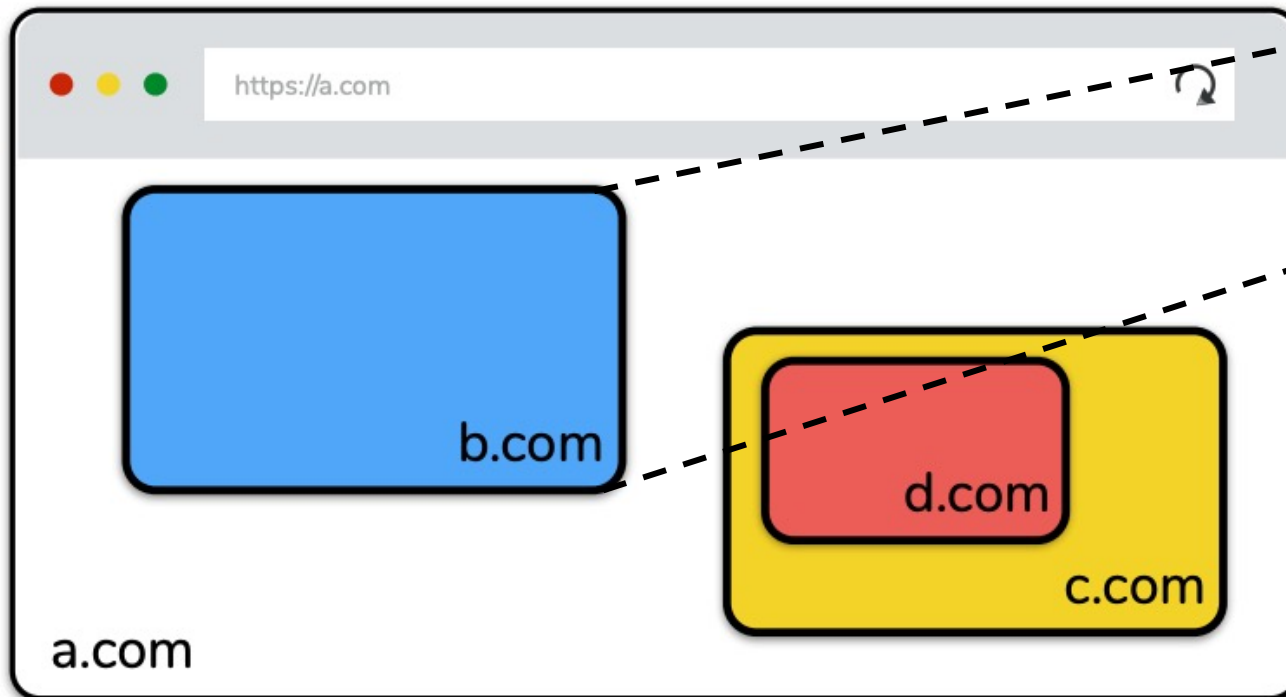
---



- You should upload a single PDF file on BlackBored.
- The name of the PDF file should have the following format: [your ID-last name.pdf]
  - If your name is Gil-dong Hong, and your ID is 20231234, then you should submit a file named “20231234-Hong.pdf”
  - If your team consists of two people, each member must submit a PDF file
- **Your proposal must follow the following format:**
  - Template: Double-Column ACM format (Sigconf style) – provided on BlackBored
  - 2 pages maximum (reference is excluded)
  - Format: Background, Motivation, Proposed Idea, Expected Results, Research Timeline, Reference

# Recap: Nested Execution Model

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



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

# Recap: Web Threat Models

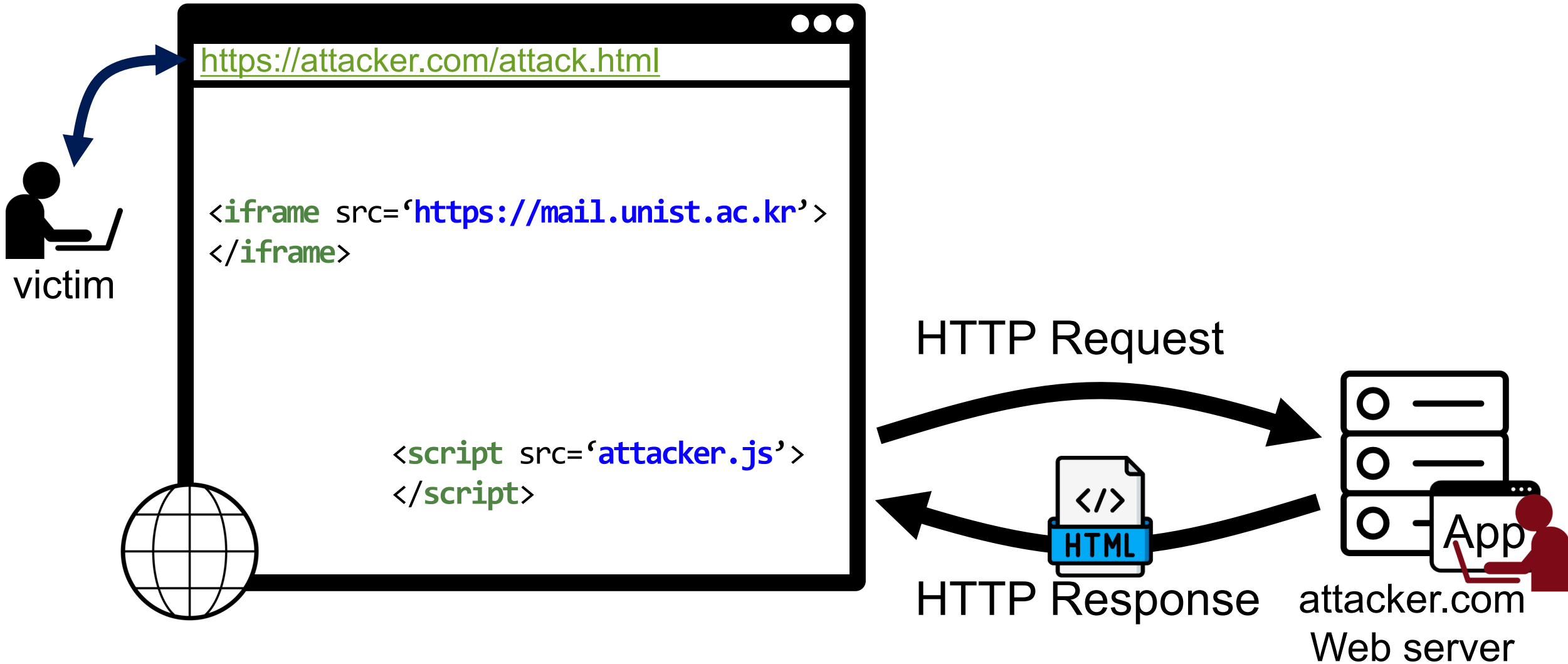
- **Network attacker:** resides somewhere in the communication link between client and server
  - Passive: eavesdropping
  - Active: modification of messages, replay...
- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server
- **Web attacker:** controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com



# Web Attacker

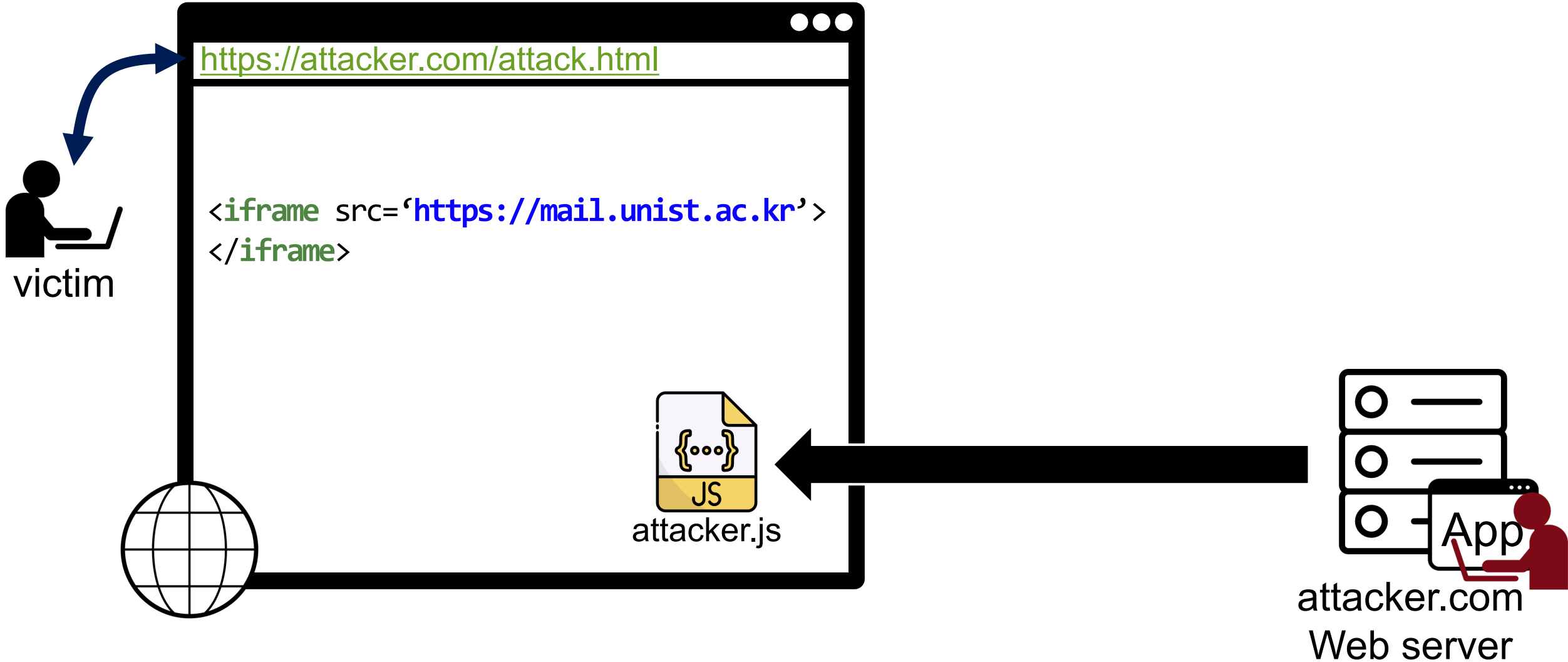


# Motivation of the Client-side Security

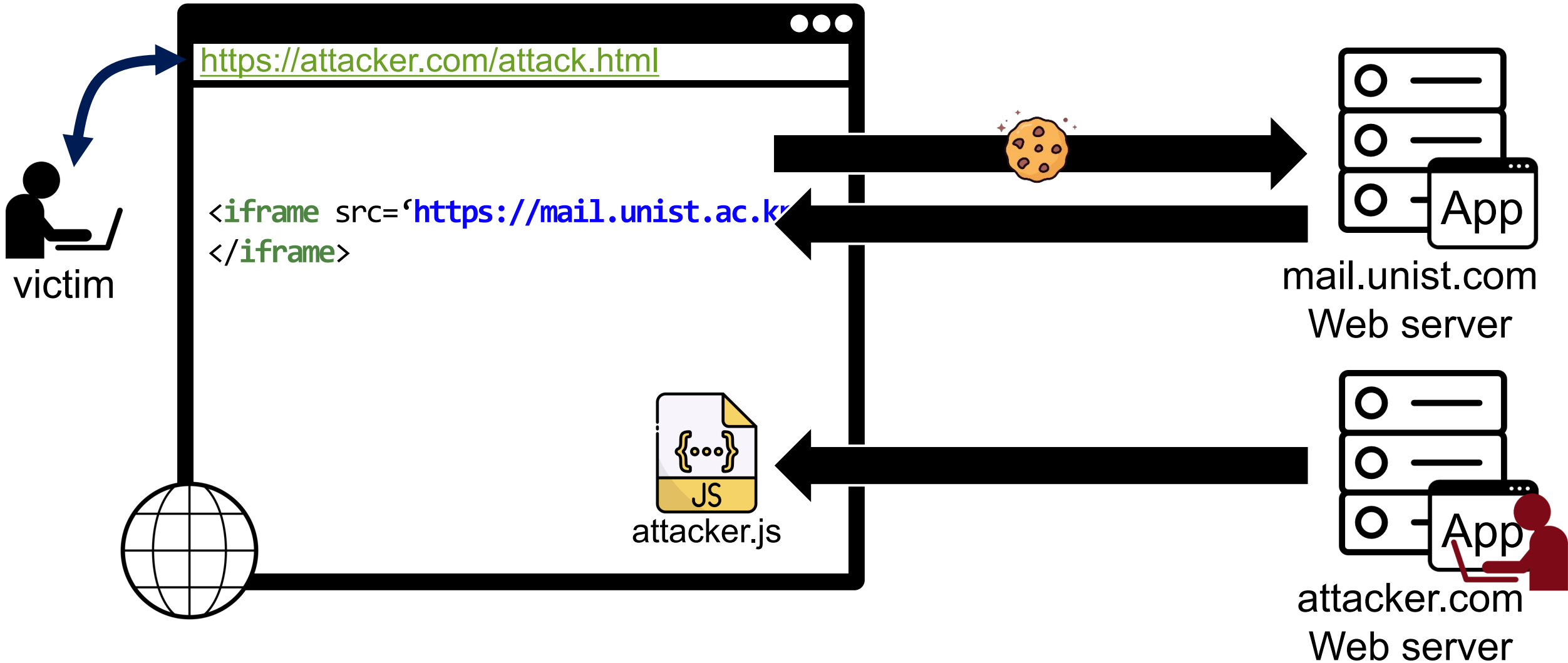




# Motivation of the Client-side Security



# Motivation of the Client-side Security



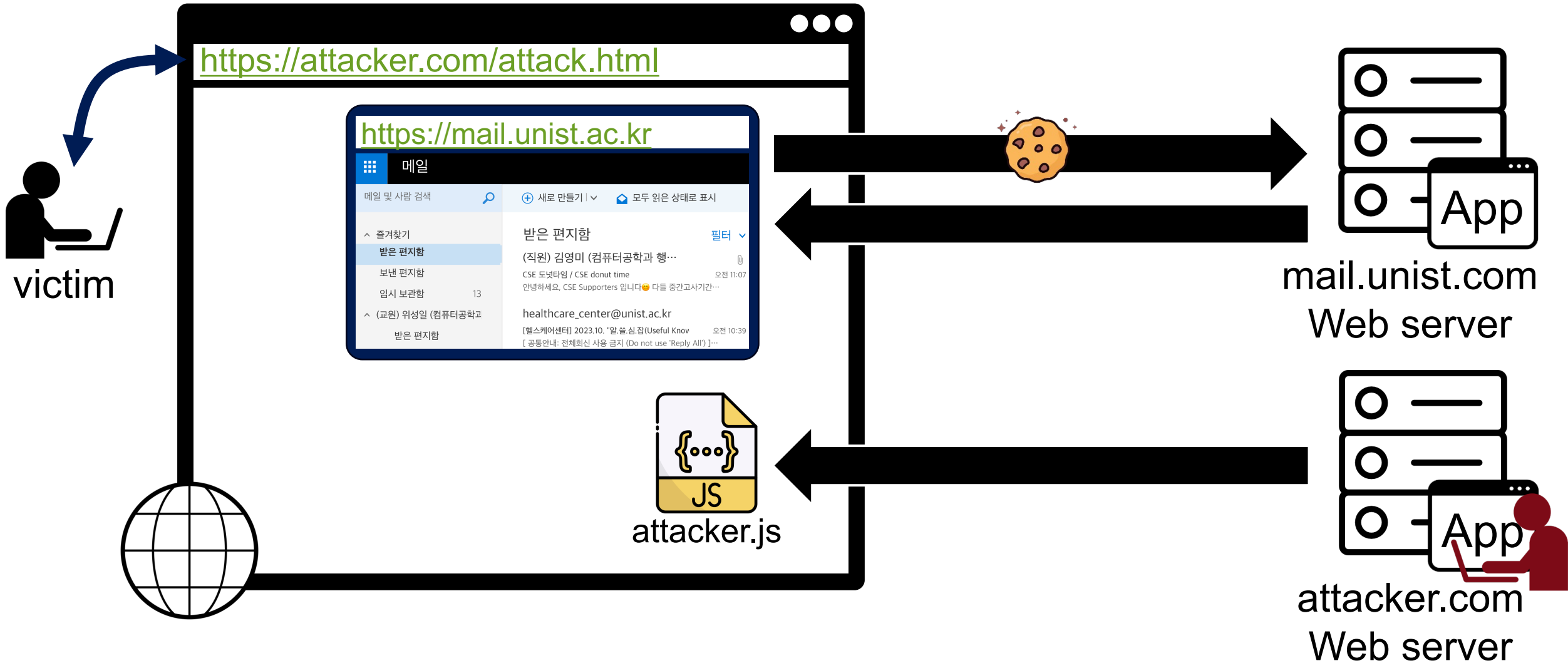
# Cookie: Making HTTP Stateful



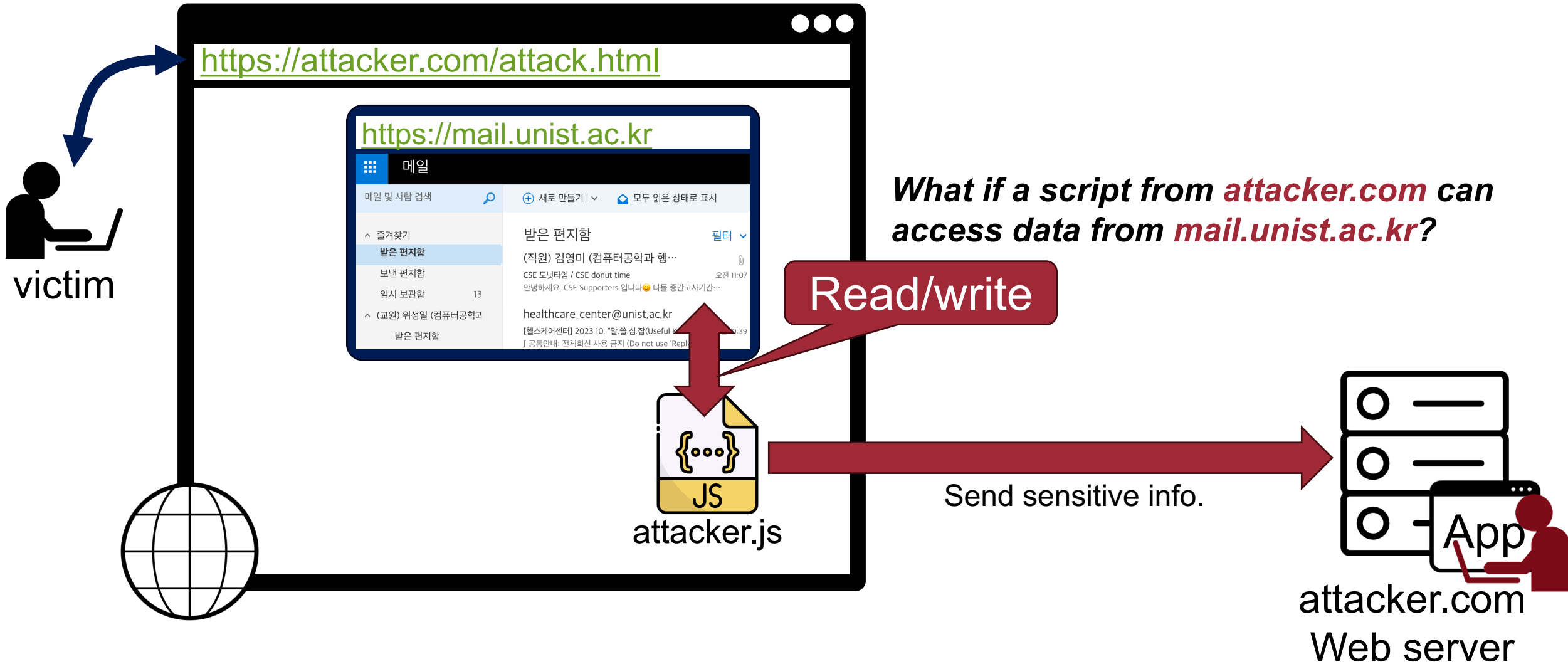
- Store a server-created file (cookie) in the browser
- Examples
  - Authentication (log in)
  - Personalization (language preference, shopping cart)
  - User tracking
- We can access all cookies for current document by `alert(document.cookie)`



# 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*?

Read/write

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

---



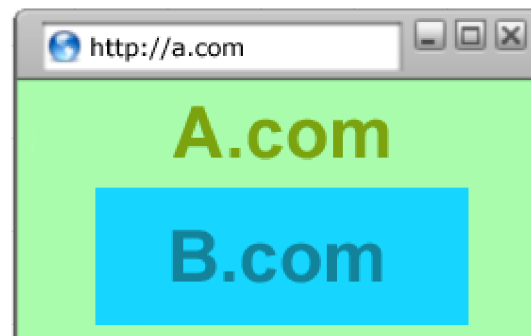
- Safe to visit an evil website



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



- Allow safe delegation

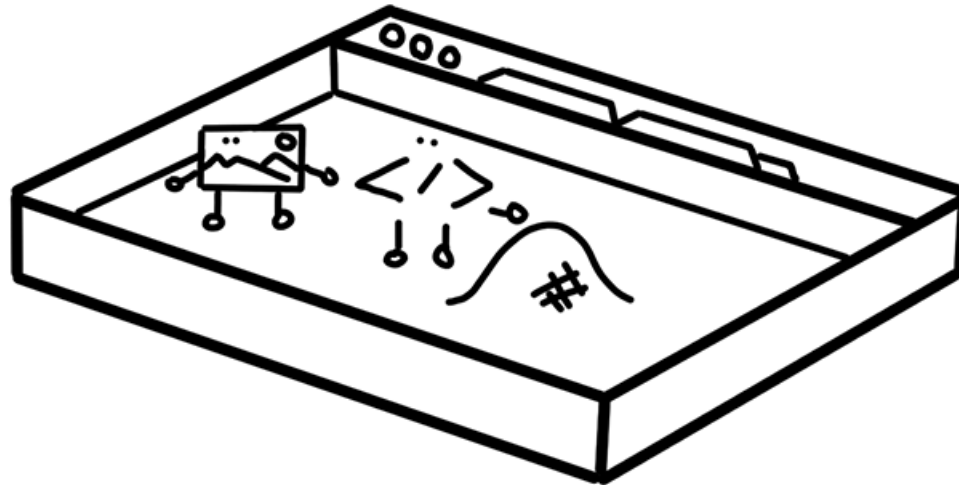




# Browser Sandbox



- 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<sup>18</sup>

---

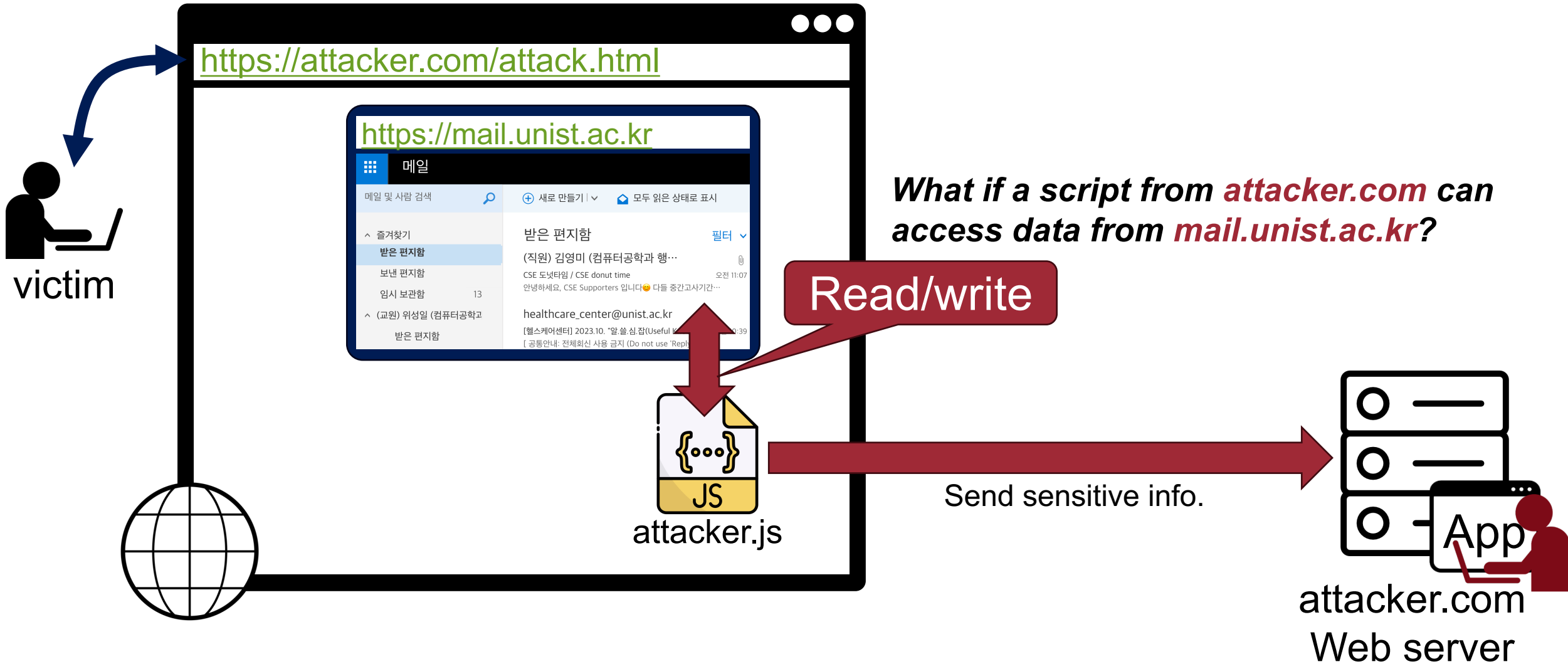
- Related to memory-level vulnerabilities, including Use-After-Free (UAF), heap overflow,...
  
- CVE-2013-6632
- CVE-2014-3188
- CVE-2015-6767
- CVE-2019-5850

# 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)

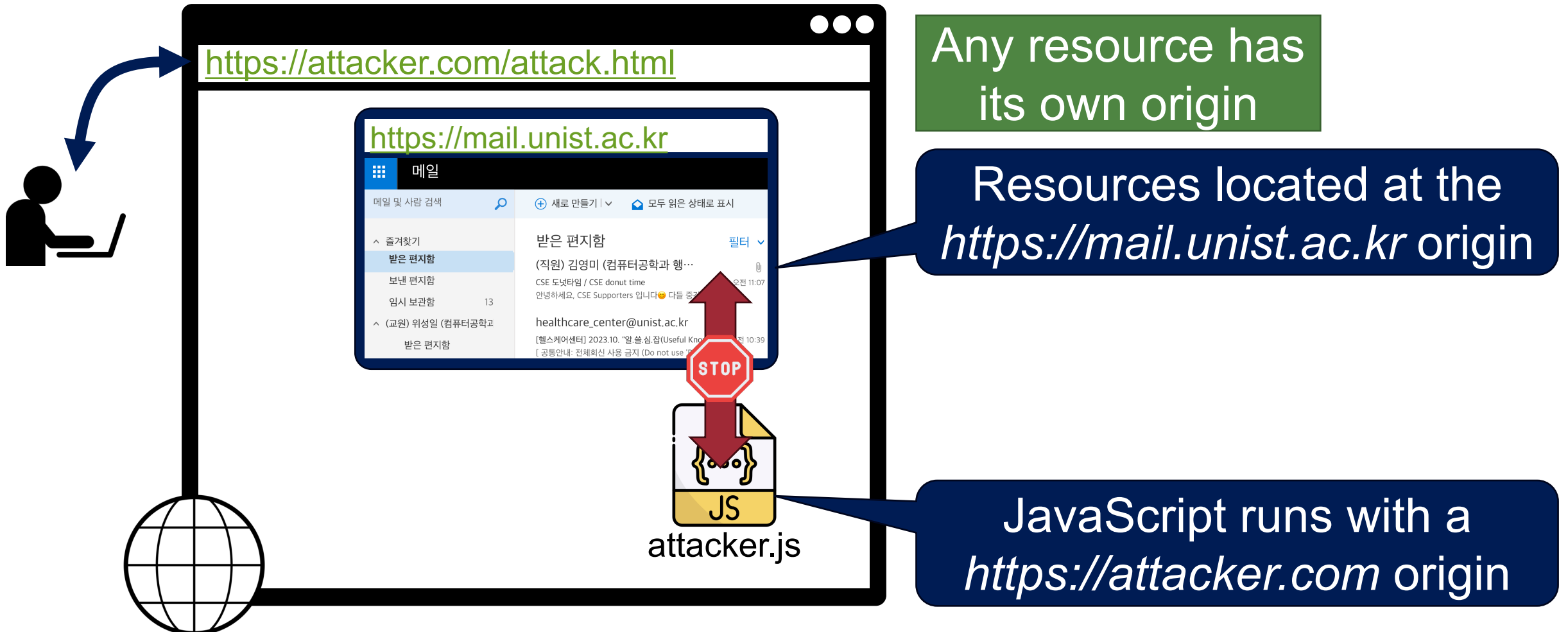
---



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

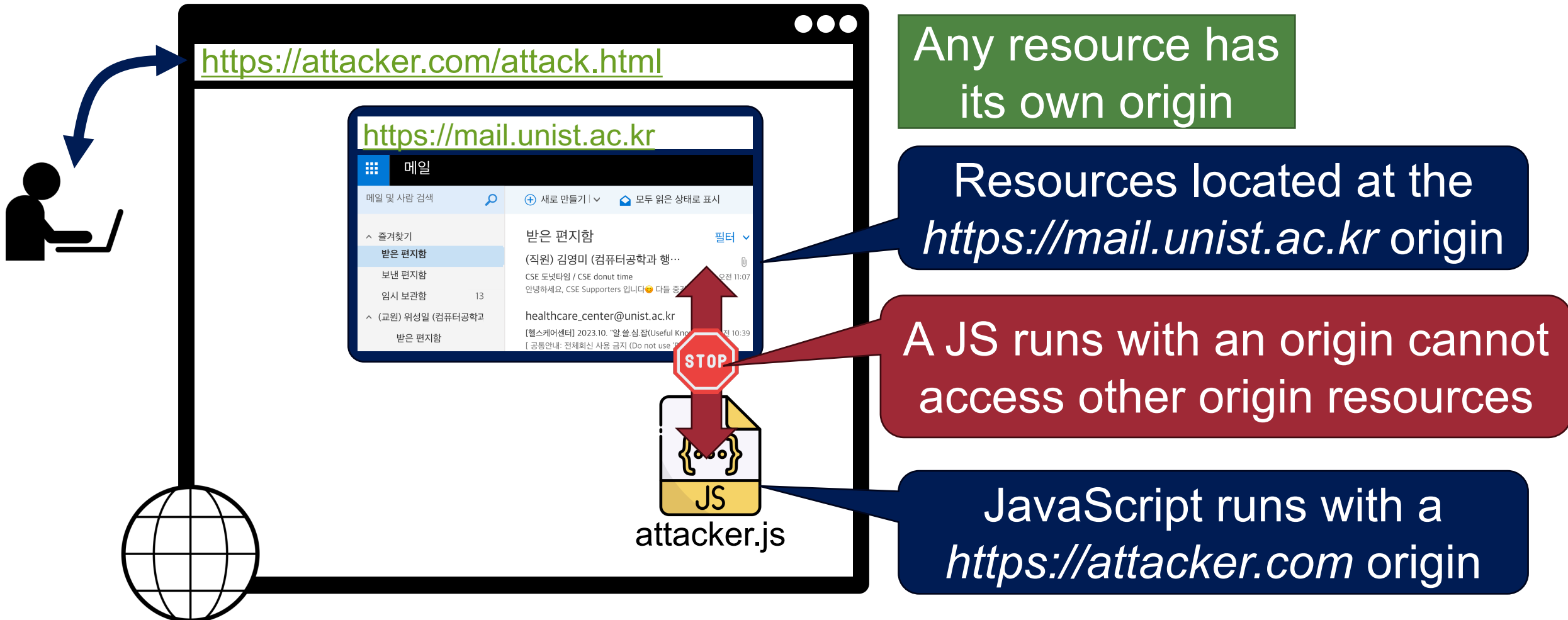
# Same Origin Policy (SOP)

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



# 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

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	<code>http://websec-lab.github.io</code>	
2	<code>https://www.websec-lab.github.io</code>	
3	<code>https://websec-lab.github.io:443</code>	
4	<code>https://websec-lab.github.io:8081</code>	
5	<code>https://websec-lab.github.io/cse610</code>	

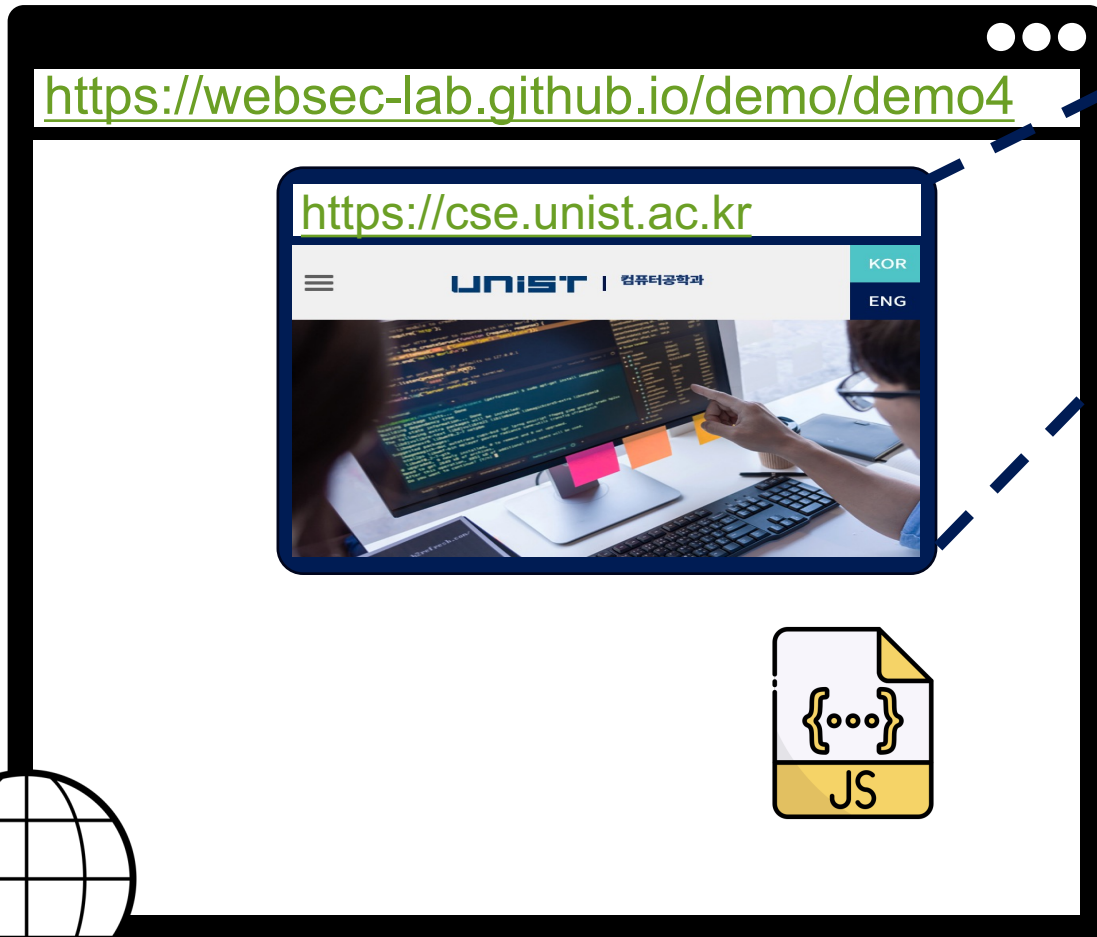
# What is an Origin?

---



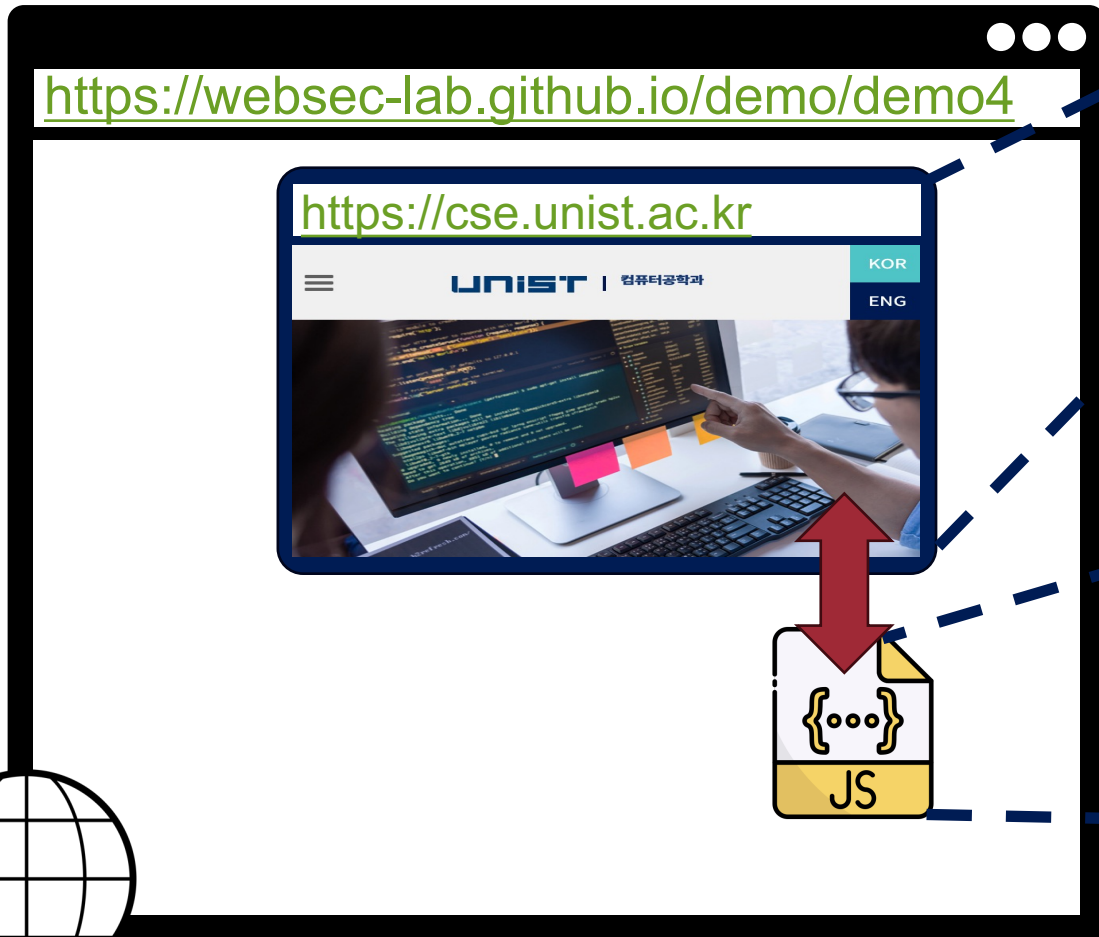
- **Origin = Protocol + Domain Name + Port**
  - origin = protocol://domain:port
- Any resource has its own origin (owner)
- Two URLs have the same origin if the **protocol, domain name** (not subdomains), **port** are the same for both URLs
  - All three must be equal origin to be considered the same

# 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/2024s-cse610/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>`, ...)



```
https://attacker.com/attack.html  
  
<script  
  src='https://cdn.com/bootstrap.js'>  
</script>  
  
<img  
  src='https://seongil.com/profile.png'>  
</img>
```

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)

# Cookie Security

How to make HTTP stateful securely?

# Same Origin Policy: "High Level"

- Recap: Same Origin Policy (SOP) for DOM:

Origin A can access origin B's DOM if match on:  
**(protocol, domain, port)**

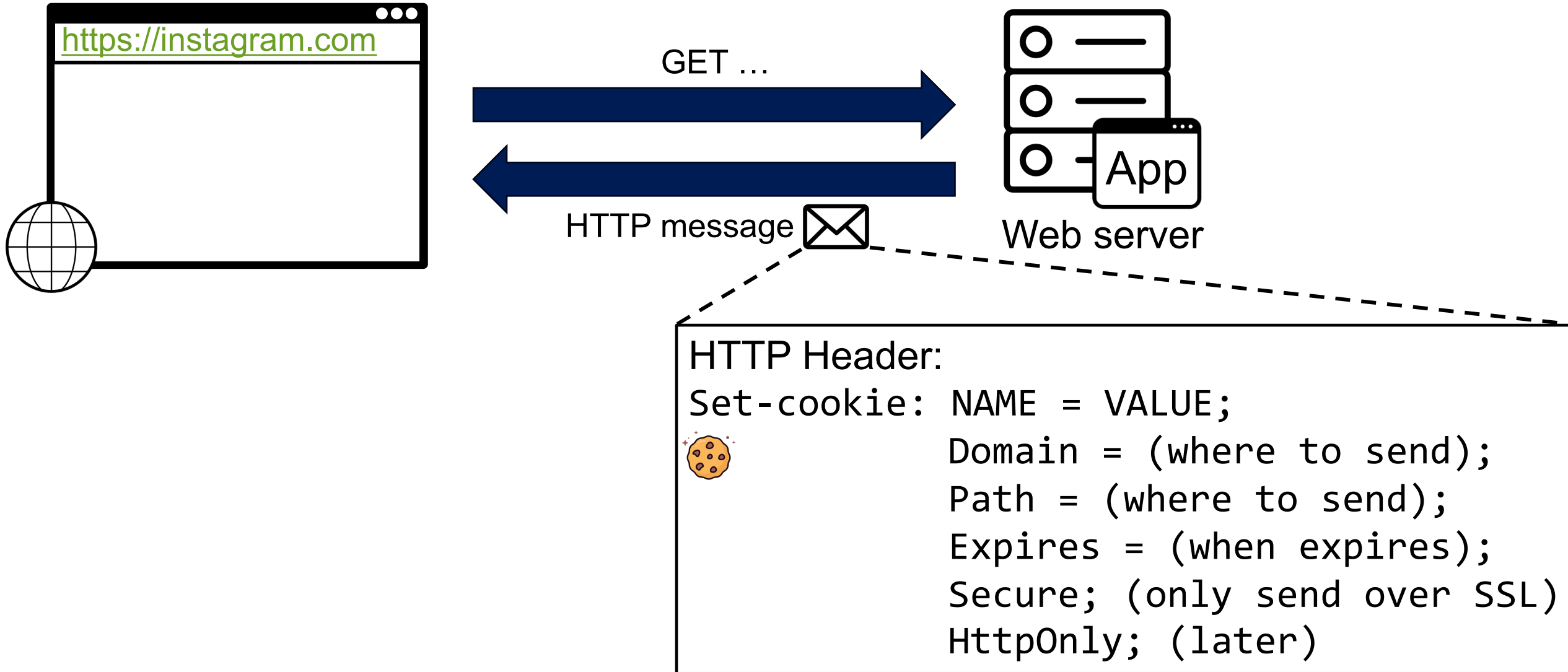
- Today: Same Origin Policy (SOP) for cookies: 

Generally speaking, based on:  
**([protocol], domain, *path*)**

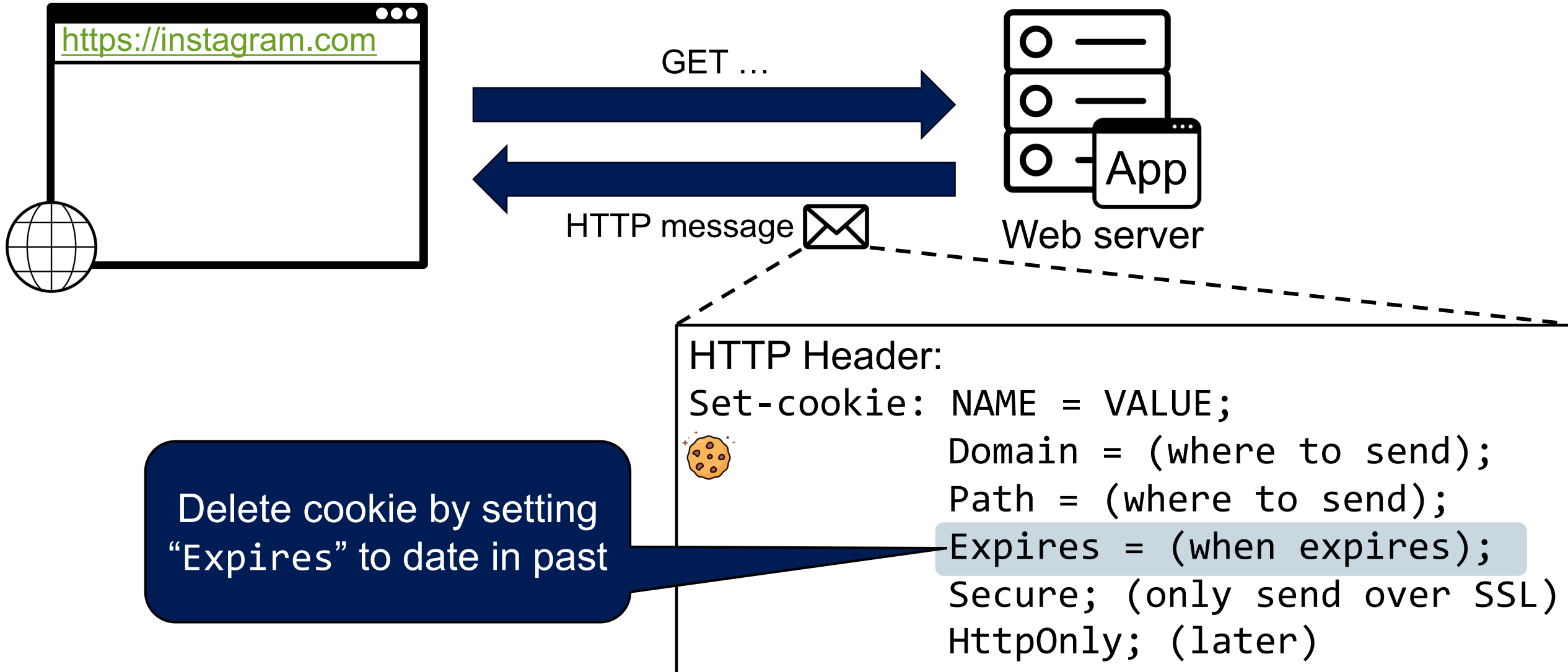
Optional

```
protocol://domain:port/path?params
```

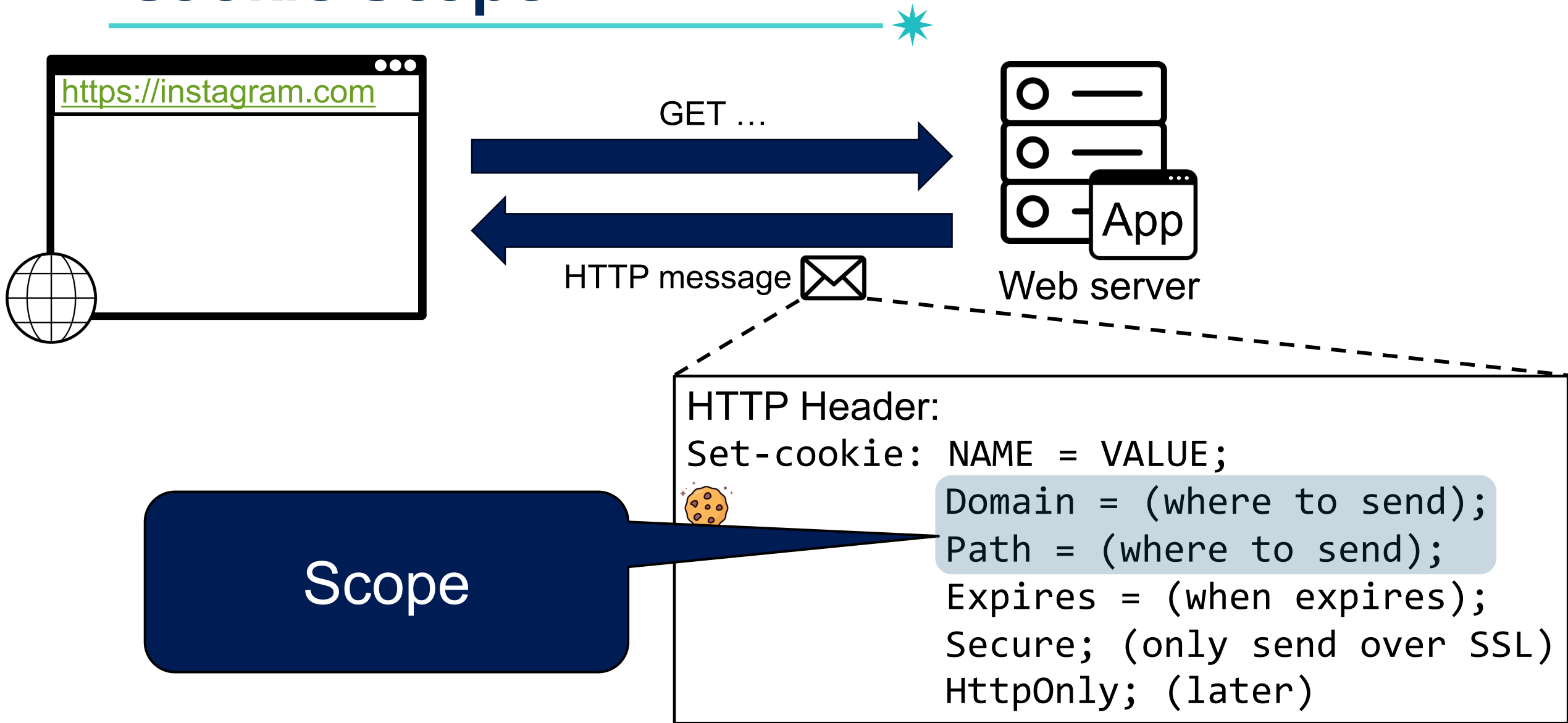
# Setting Cookies by Server



# Deleting Cookies by Server

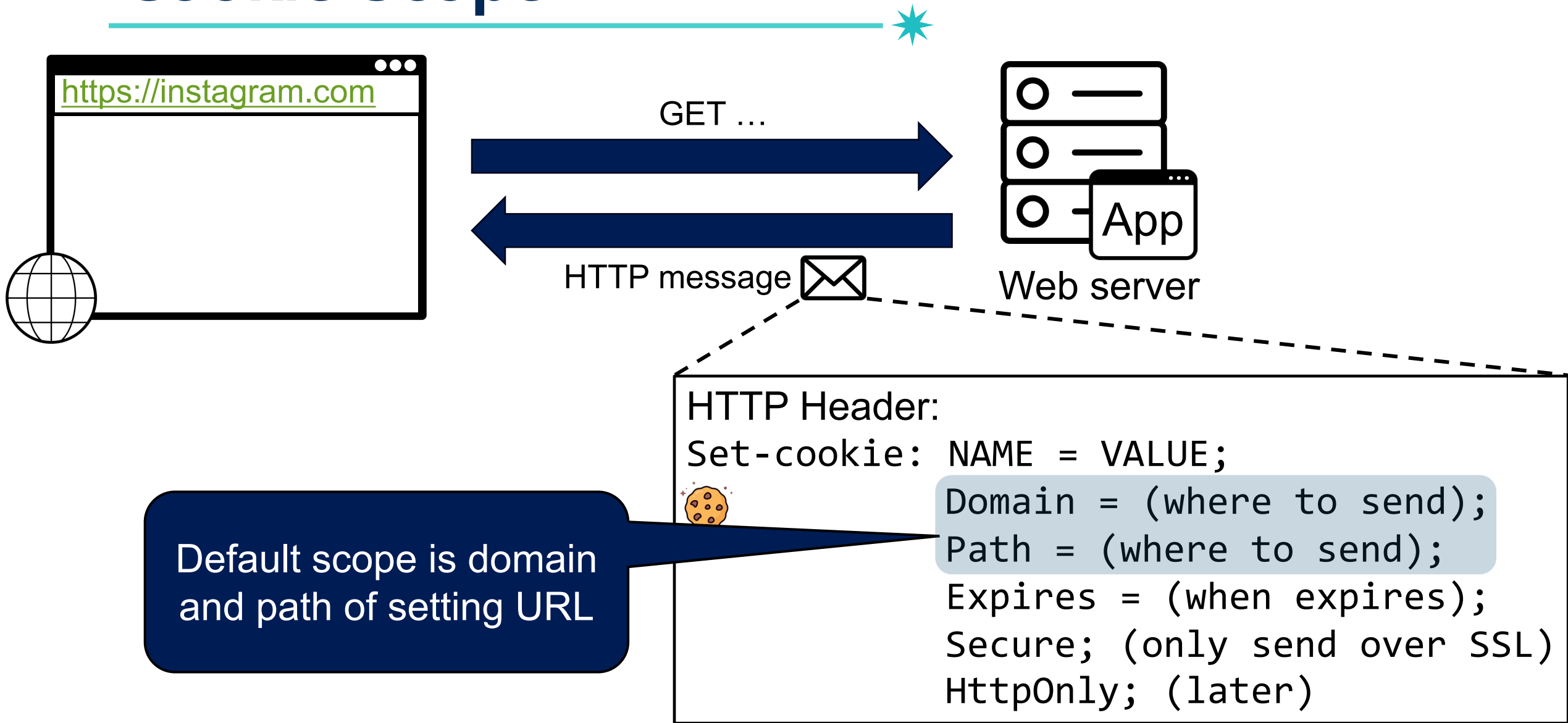


# Cookie Scope



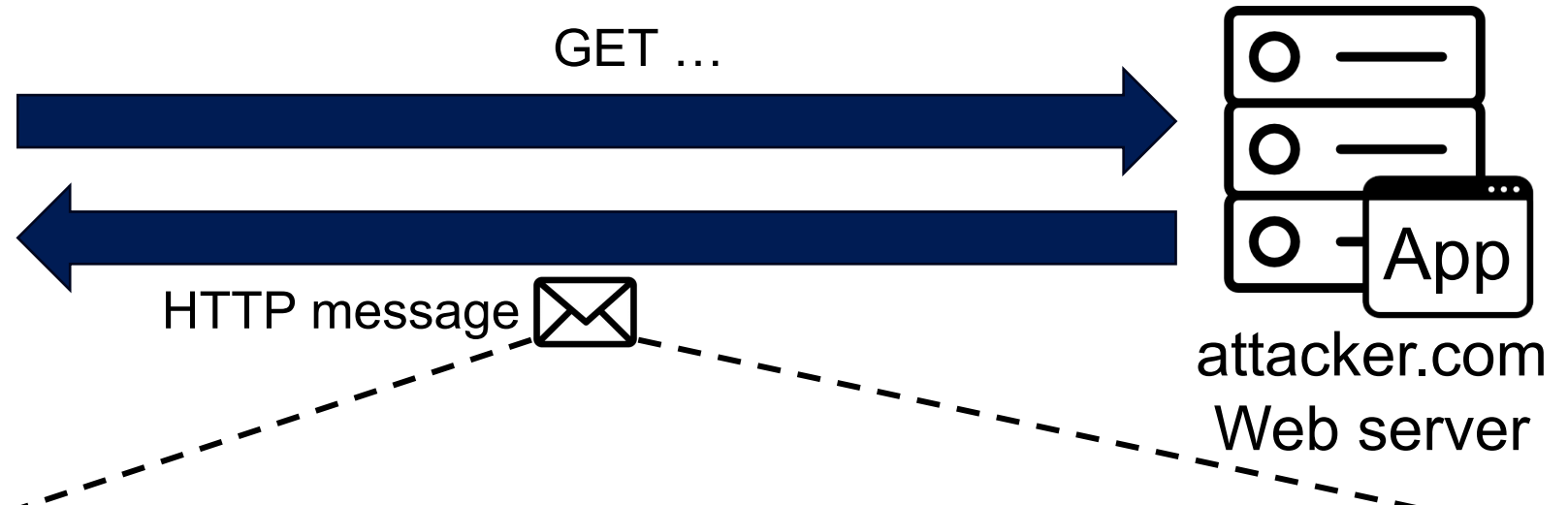
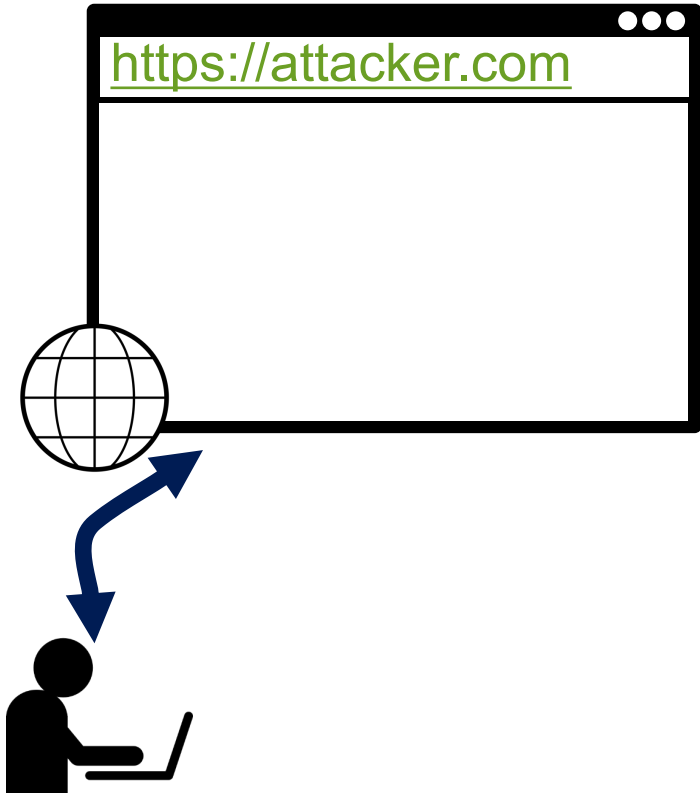


# Cookie Scope



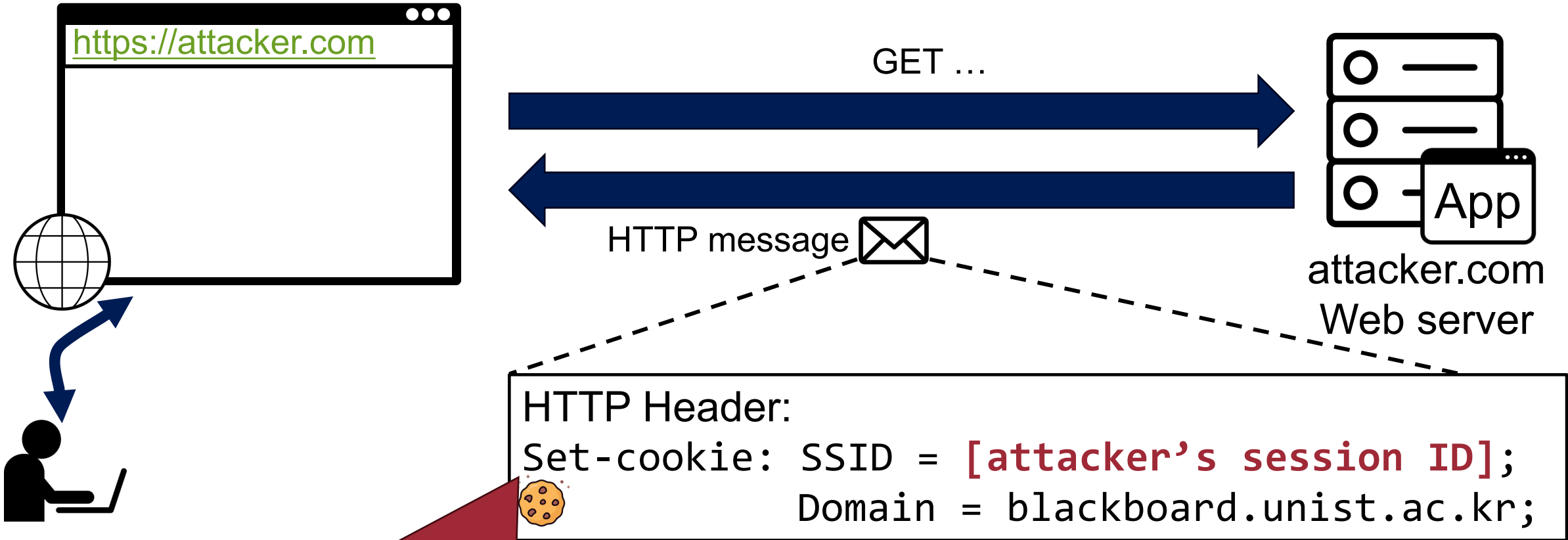
# Motivating Example of the Cookie Write SOP

43



```
HTTP Header:  
Set-cookie: SSID = [attacker's session ID];  
Domain = blackboard.unist.ac.kr;
```

# Motivating Example of the Cookie Write SOP



The attacker can write arbitrary values to the arbitrary cookies

# Scope Setting Rules (Write SOP)

---

- **Domain:** any domain-suffix of URL-hostname, except Top Level Domain (TLD)

- **Path:** can be set to anything

# Scope Setting Rules (Write SOP)

- **Domain:** any domain-suffix of URL-hostname, except Top Level Domain (TLD)

Question: which cookies can be set by login.site.com?

Idx	Cookie's domain	Write Allowed?
1	login.site.com	
2	.site.com	
3	.com	
4	seongyun.site.com	
5	othersite.com	

- **Path:** can be set to anything

# Cookies are Identified by (name, domain, path)

48

Cookie 1 🍪

name = **userid**

Value = test

domain = **login.site.com**

Path = /

secure



Cookie 2 🍪

name = **userid**

Value = test123

domain = **.site.com**

Path = /

secure

Distinct  
cookies

# Browser's Cookie Jar

Cookie 1 🍪

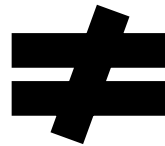
name = **userid**

Value = test

domain = **login.site.com**

Path = /

secure



Cookie 2 🍪

name = **userid**

Value = test123

domain = **.site.com**

Path = /

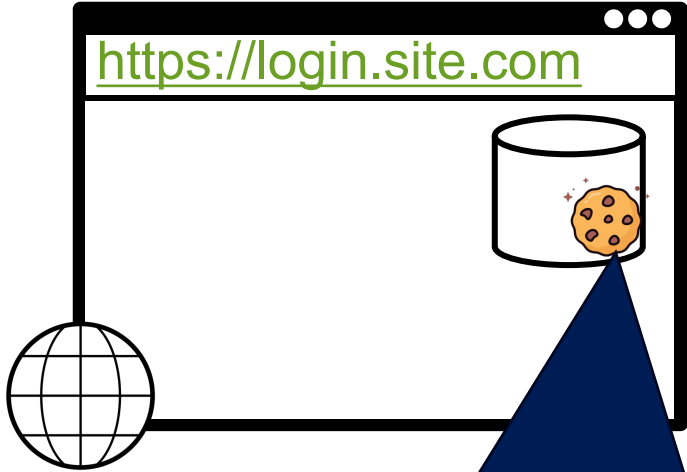
secure

<https://login.site.com>

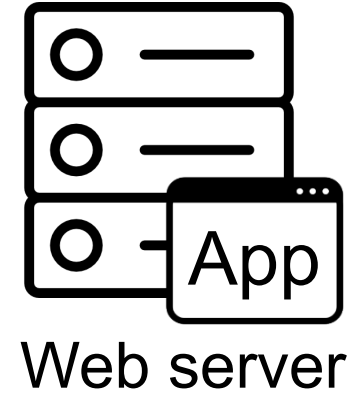


Both cookies are stored in browser's cookie jar;  
Both are in scope of login.site.com

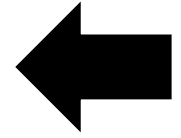
# Reading Cookies on Server (Read SOP)



GET `protocol://URL-domain/URL-path`  
Cookie: Name= value 🍪



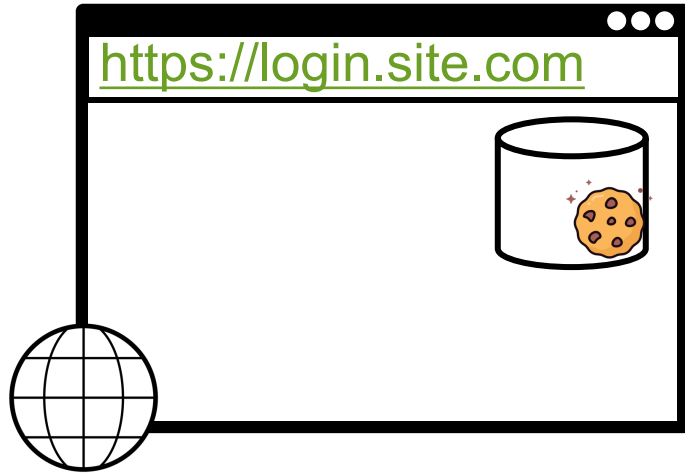
Which cookies will be sent from the browser?



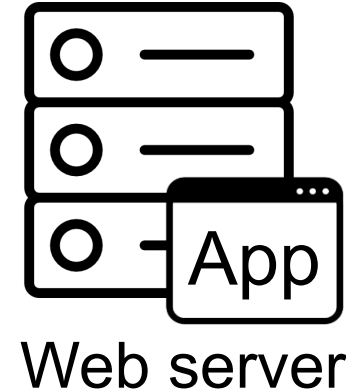
Which cookies will be visible from the server?



# Reading Cookies on Server (Read SOP)

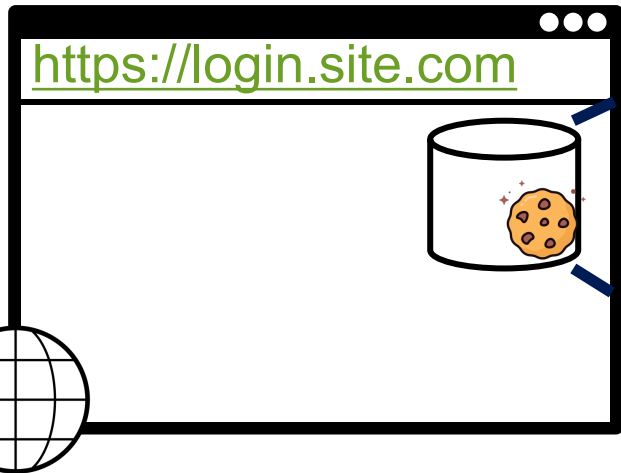


GET `protocol://URL-domain/URL-path`  
Cookie: Name= value 🍪



- **Browser sends all cookies in URL scope:**
  - Cookie domain is domain-suffix of `URL-domain`, and
  - Cookie path is prefix of `URL-path`, and
  - [`protocol`=HTTPS if cookie is “secure”]
- **Goal:** server only sees cookies in its scope

# Quiz: Read SOP



Cookie 1 🍪  
name = **userid**  
Value = u1  
domain = **login.site.com**  
Path = /  
secure

Cookie 2 🍪  
name = **userid**  
Value = u2  
domain = **.site.com**  
Path = /  
non-secure

Idx	Request Domains	Sent Cookies
1	http://checkout.site.com/	
2	http://login.site.com/	
3	https://login.site.com/	

# Client-side Read/Write: `document.cookie`


54

- Setting a cookie in JavaScript:
  - E.g., `document.cookie = "name=value; expires=...;"`
  - E.g., `document.cookie = "ssid=AB31FBS5; domains=.unist.ac.kr"`
- Reading a cookie: `alert(document.cookie)`
  - Prints string containing all cookies available for document
- Deleting a cookie:
  - `Document.cookie = "name=; expires=Thu, 01-Jan-70"`

# Client-side Read/Write: `document.cookie`

55

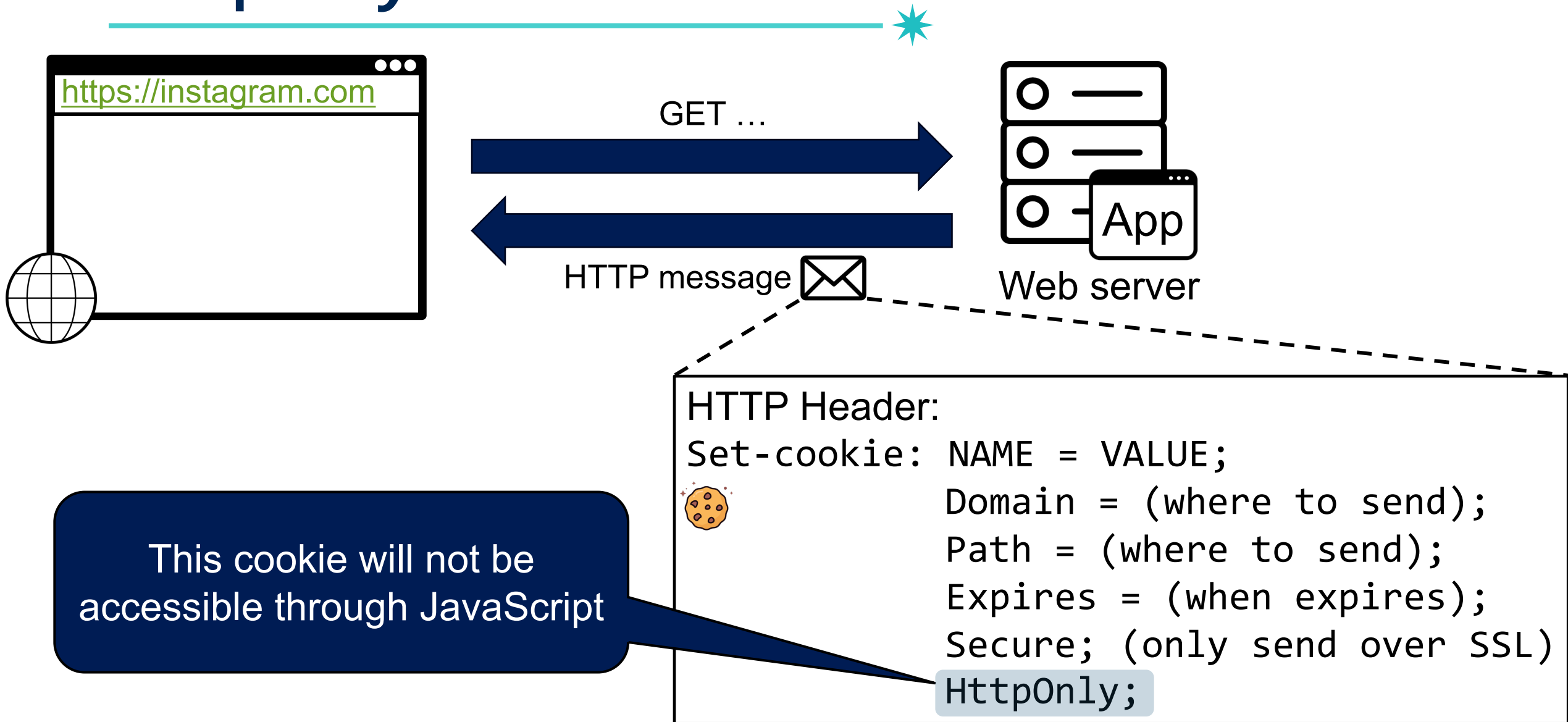
We can access all cookies for current document by `alert(document.cookie)`



```
security=low; PHPSESSID=ca5213aba0449128c7caf0902b77f1e0
```

OK

# HttpOnly Cookies



# HttpOnly Cookies



- Cookie sent over HTTP(s), but not accessible to scripts
  - Cannot be read via `document.cookie`
  - Helps prevent cookie theft attacks

The screenshot shows the Chrome DevTools Application tab with the 'Cookies' section expanded. A table lists several cookies, with the 'HttpOnly' and 'Secure' columns highlighted in red for the 'AEC' and 'NID' cookies.

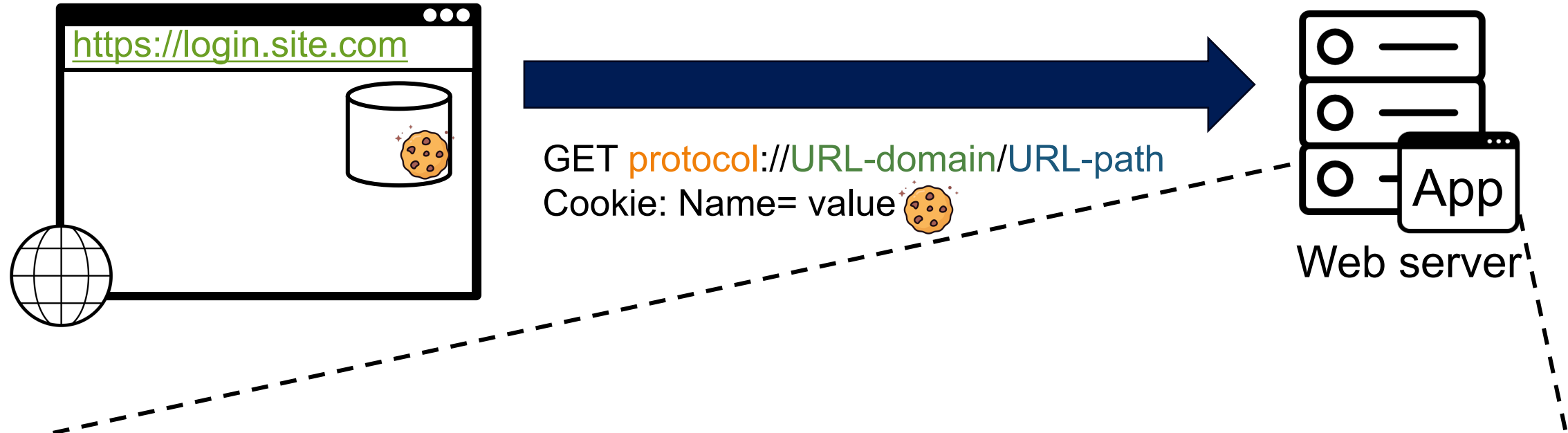
Name	Value	Domain	Path	Expire...	Size	HttpOnly	Secure
1P_JAR	2024-3-5-0	.google.com	/	2024-0...	16		
AEC	Ae3NU9OFRb9IfWD3Zj...	.google.com	/	2024-0...	61	✓	✓
DV	E_oD5fj1VMMu0J-HU_...	www.google.com	/	2024-0...	49		
NID	512=DY8q1spa2bEC5...	.google.com	/	2024-0...	210	✓	✓
OGPC	19037049-1:	.google.com	/	2024-0...	15		
OTZ	7454897_20_20__20_	www.google.com	/	2024-0...	21		✓

# HttpOnly Cookie Demo

# Cookie Protocol Problems



# Cookie Protocol Problems

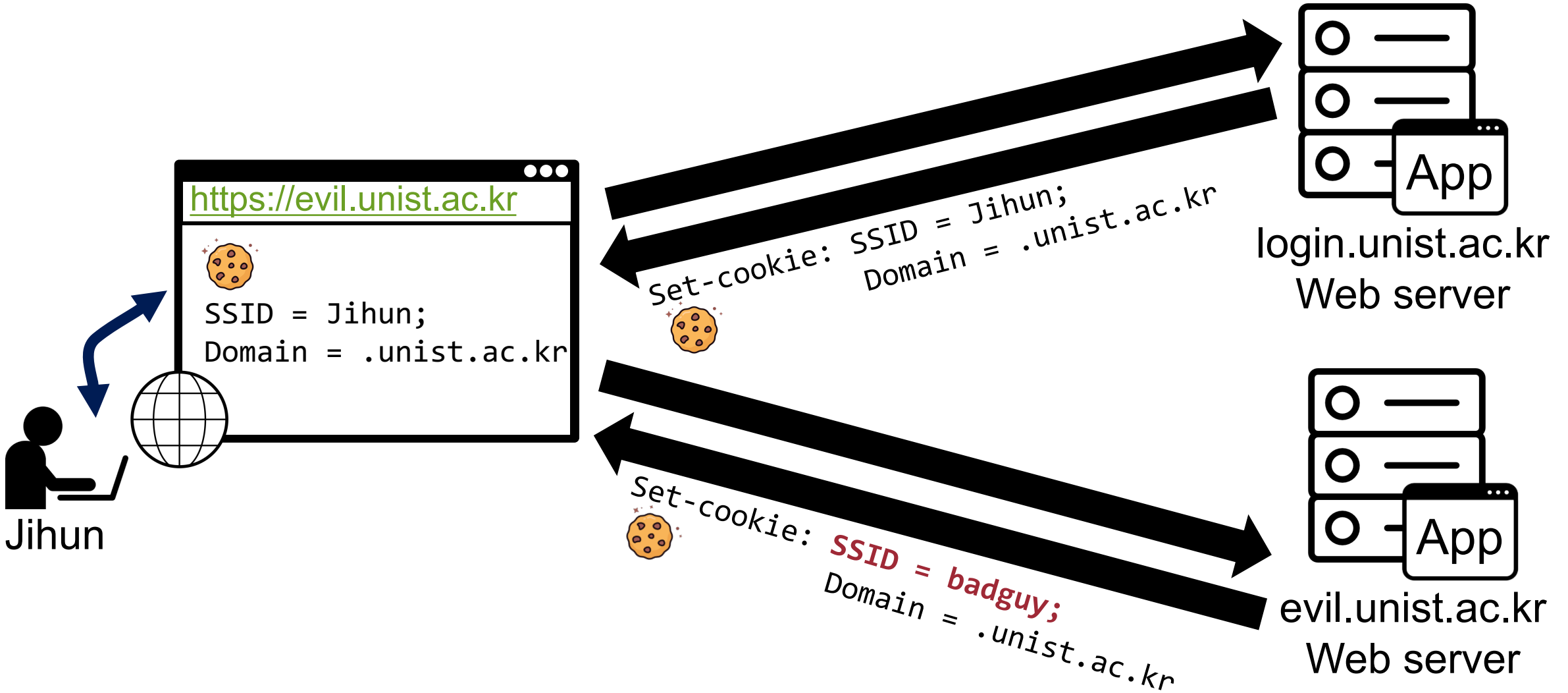


- Server is blind:
  - Does not see cookie attributes (e.g., secure, HttpOnly attributes)
  - Does not see which domain set the cookie
  - **Server only sees: cookie: NAME=VALUE**

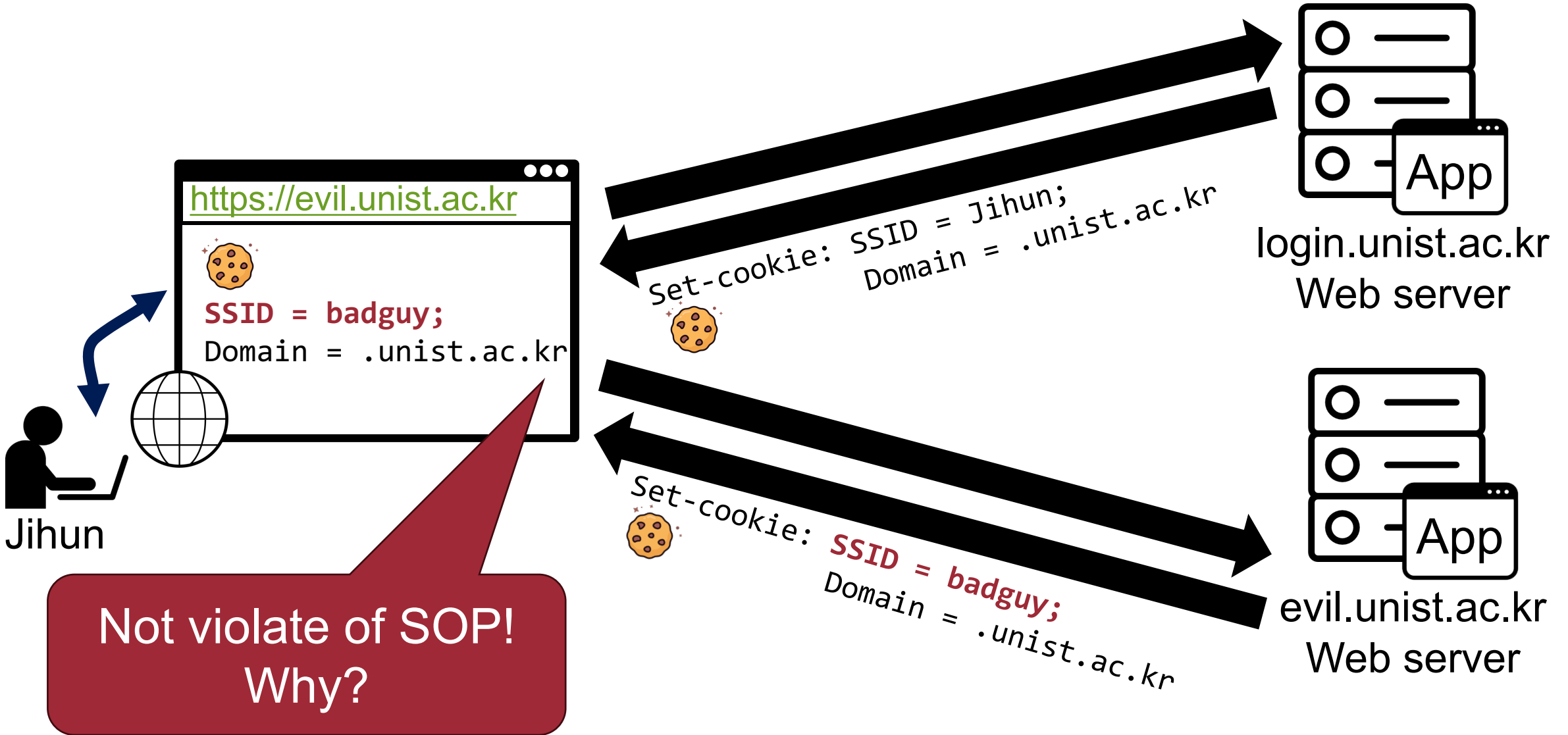
# Example 1: Login Server Problems



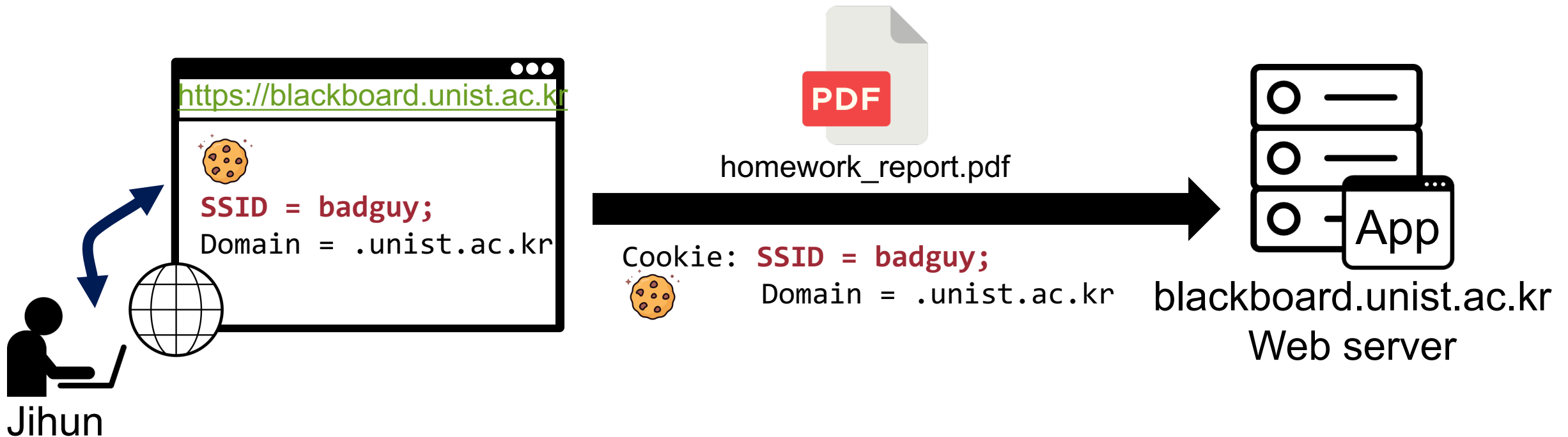
# Example 1: Login Server Problems



# Example 1: Login Server Problems



# Example 1: Login Server Problems



Jihun's homework will be submitted under the name "badguy"

# Example 1: Login Server Problems

---

1. Jihun logs in at **login.unist.ac.kr**  
login.site.com sets session-id cookie for **.unist.ac.kr**
2. Jihun visits **evil.unist.ac.kr**  
overwrites **.unist.ac.kr** session-id cookie with session-id of user “badguy”
3. Jihun visits **blackboard.unist.ac.kr** to submit homework  
Jihun’s homework will be submitted under the name “badguy”

The web server **blackboard.unist.ac.kr** expects session-id from **login.unist.ac.kr**; But it cannot know that session-id cookie was overwritten

# Example 2: “secure” Cookies are not Secure

1. Alice logs in at **https://accounts.google.com**
  - Set-Cookie: SSID=Au7\_ESAgDpKY5TGnf; Domain=.google.com; Path=/; Expires=Wed, 09-Mar-2026; **Secure; HttpOnly**
2. Alice visits **http://www.google.com** (cleartext)
  - **Network attacker** can inject into response **Set-Cookie: SSID=badguy; secure** and overwrite secure cookie

**Network attacker can re-write HTTPS cookies  
⇒ HTTPS cookie value cannot be trusted**

# Recap: Web Threat Models

- **Network attacker:** resides somewhere in the communication link between client and server
  - Passive: eavesdropping
  - 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





## Example 3: Path Separation is not a Security Measure



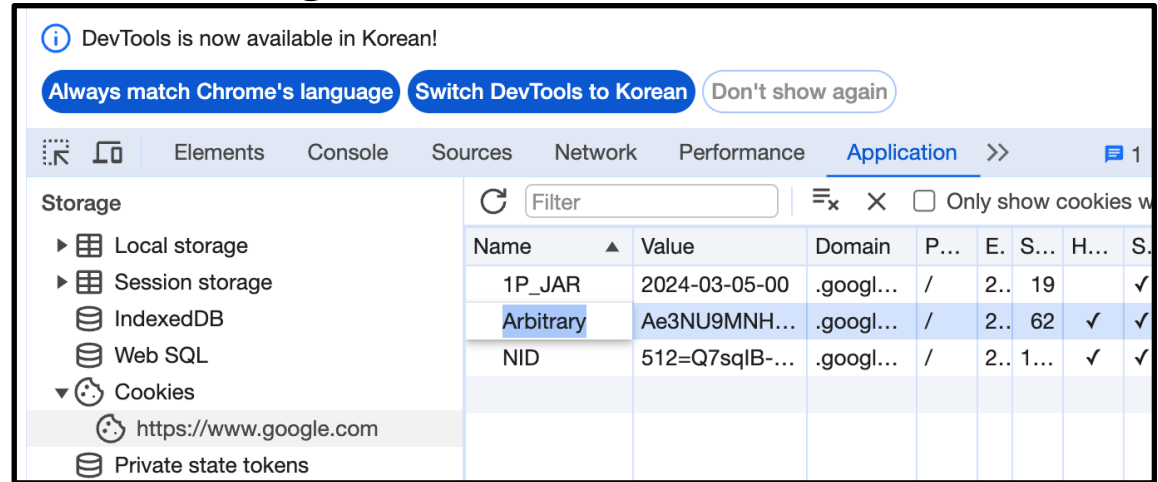
- Cookie SOP: path separation
  - **x.com/A** does not see cookies of **x.com/B**
- Not a security measure:
  - DOM SOP: **x.com/A** has access to DOM of **x.com/B**

```
<iframe src="x.com/B"></iframe>  
alert(frames[0].document.cookie);
```

- Path separation is done for efficiency not security:
  - x.com/A is only sent the cookies it needs

# Root Cause: Cookies have No Integrity

- End users and attackers can change and delete cookie values



- Simple example: shopping cart software

**Set-cookie: shopping-cart-total = 150 (\$)**

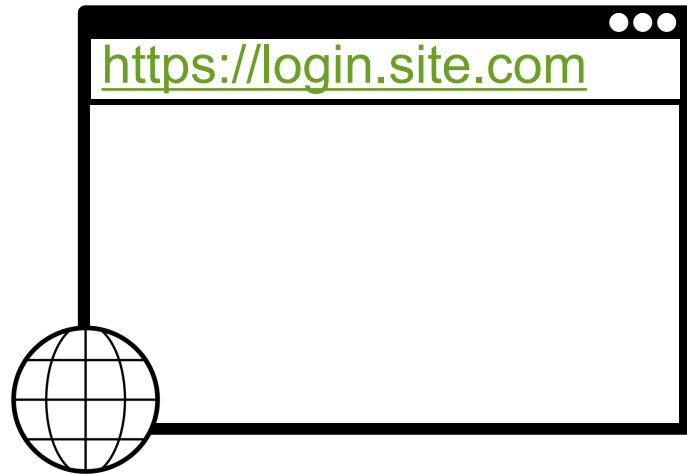
User edits cookie contents (cookie poisoning):

**Cookie: shopping-cart-total = 15 (\$)**

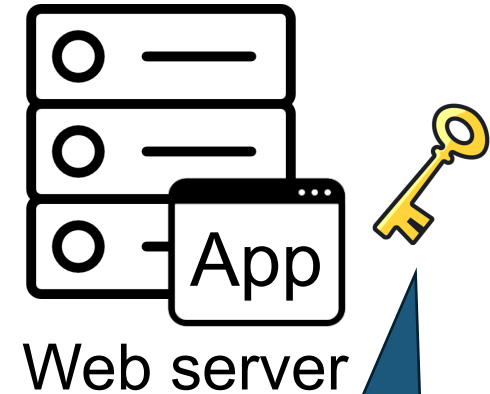
# Solution: Cryptographic Checksums

- **Goal:** data integrity

Generating tag  $T$   
 $\leftarrow \text{MACsign}(k, \text{SSID} || \text{name} || \text{value})$



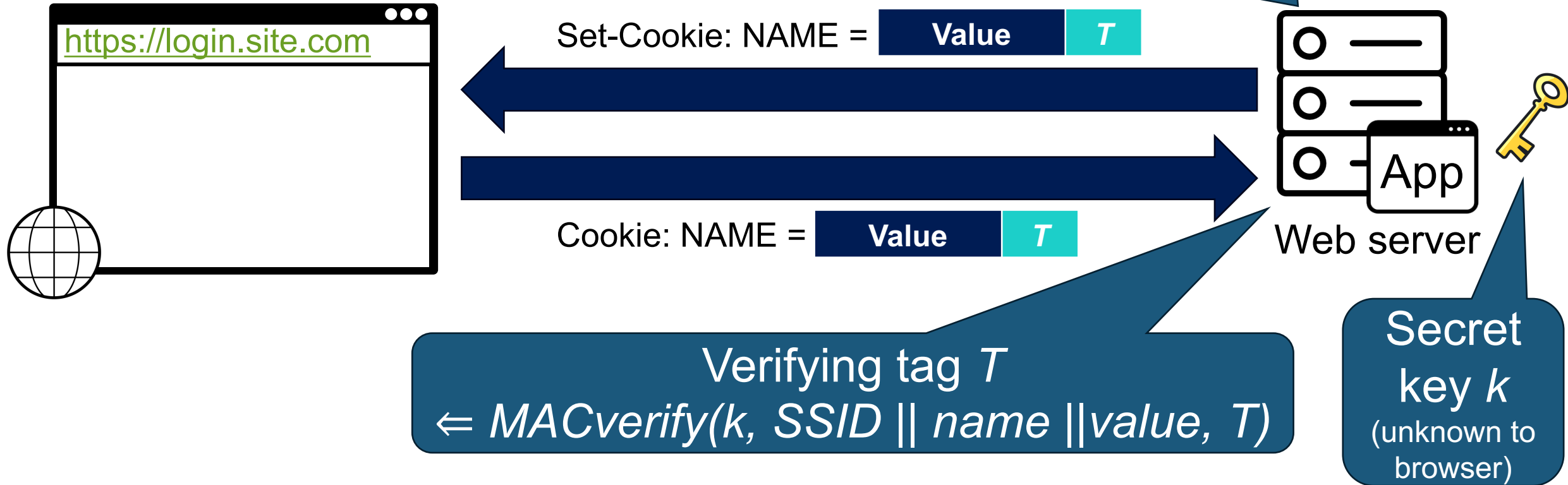
Set-Cookie: NAME = Value T



Secret  
key  $k$   
(unknown to  
browser)

# Solution: Cryptographic Checksums

- **Goal:** data integrity



Binding to session-id (SSID) makes it harder to replay old cookies

# Example: ASP.NET

---



- `System.Web.Configuration.MachineKey`
  - Secret web server key intended for cookie protection
- Creating an encrypted cookie with integrity:

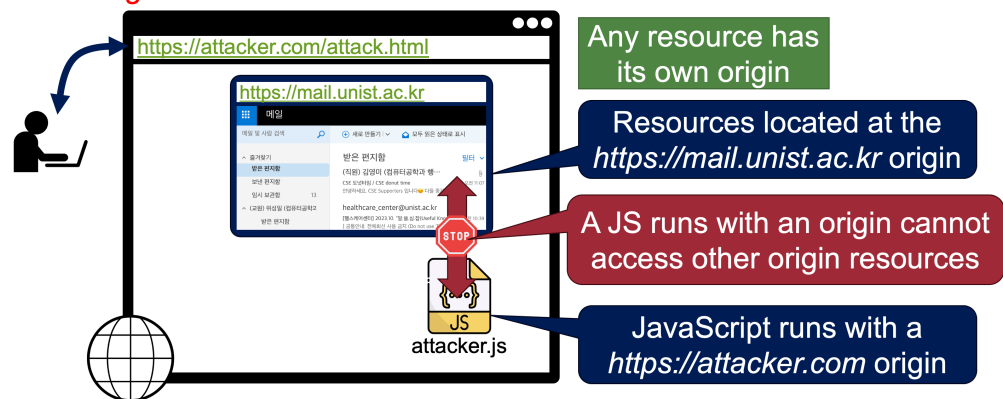
```
HttpContext cookie = new HttpContext(name, val);
HttpContext encodedCookie = HttpContextSecureCookie.Encode(cookie);
```
- Decrypting and validating an encrypted cookie:

```
HttpContextSecureCookie.Decode(cookie);
```

# Conclusion

## Same Origin Policy (SOP) <sup>23</sup>

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



## Same Origin Policy: "High Level" <sup>38</sup>

- Recap: Same Origin Policy (SOP) for DOM:

Origin A can access origin B's DOM if match on:  
(**protocol**, **domain**, **port**)

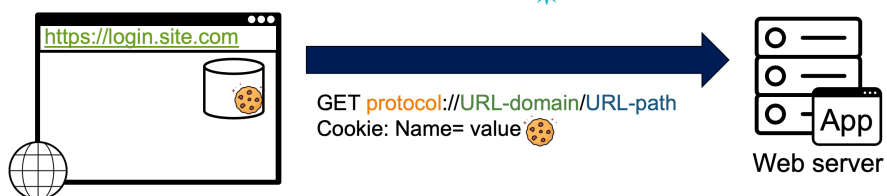
- Today: Same Origin Policy (SOP) for cookies: 🍪

Generally speaking, based on:  
(**[protocol]**, **domain**, **path**)

Optional

`protocol://domain:port/path?params`

## Reading Cookies on Server (Read SOP) <sup>51</sup>



- Browser sends all cookies in URL scope:**
  - Cookie domain is domain-suffix of **URL-domain**, and
  - Cookie path is prefix of **URL-path**, and
  - `[protocol=HTTPS` if cookie is "secure"]
- Goal:** server only sees cookies in its scope

## Scope Setting Rules (Write SOP) <sup>45</sup>

- Domain:** any domain-suffix of URL-hostname, except Top Level Domain (TLD)
- Path:** can be set to anything

**Question?**