CSIT 5740 Introduction to Software Security

Note set 5A

Dr. Alex LAM

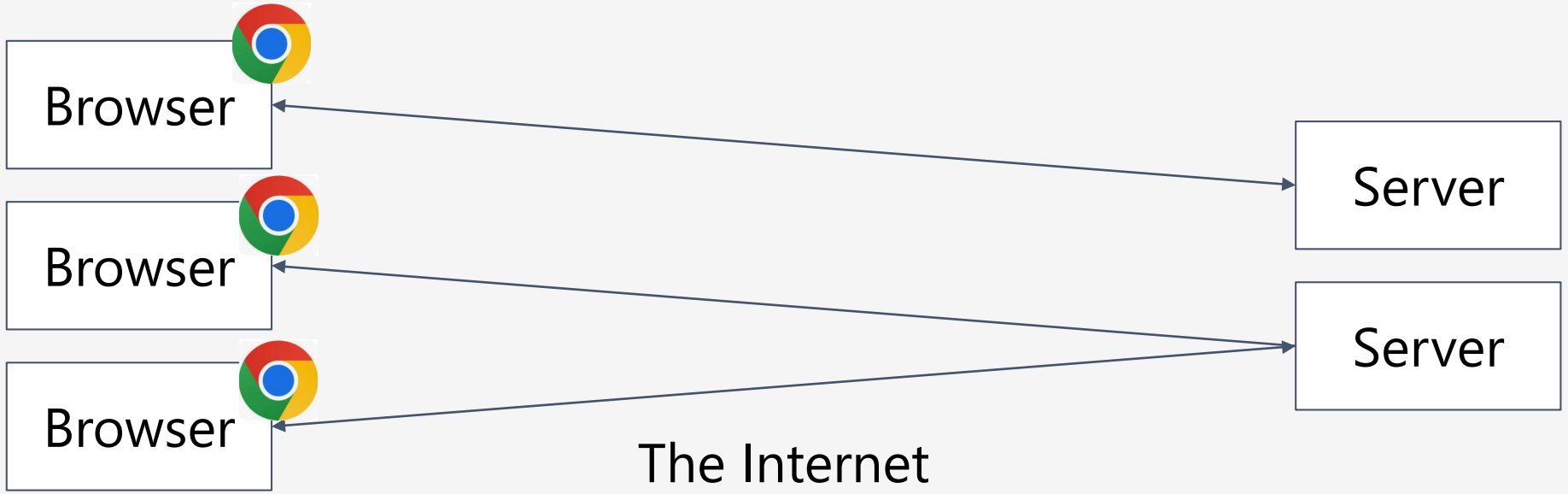
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY | DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The set of note is adapted and converted from a software security course by Prof. Antonio Bianchi and Prof. David Wagner

Introduction to 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



Today: Elements of the Web

- **URLs (Uniform Resource Locators):** It is the address of a unique resource on the internet. It is a mechanism used by browsers to retrieve a piece of data uniquely from the web?
- **HTTP (Hypertext Transfer Protocol).** It is an application protocol that contains a list of guidelines for the communications of (typically) web browsers and web servers on the World Wide Web (WWW)
- Data on a webpage can contain:
  - **HTML (Hypertext Markup Language)**: A markup language for creating webpages
  - **CSS (Cascading Style Sheets)**: A style sheet language for defining the appearance of webpages
  - **JavaScript**: A programming language for running code in the web browser

URLs

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: Protocols

- Located just before the double slashes
- Defines how to retrieve the data over the Internet (which Internet protocol to use)
- Common Protocols you may know
  - **`http`**: Hypertext Transfer Protocol
  - **`https`**: A secure version of HTTP
- Other protocols include:
  - **`ftp`**: File Transfer Protocol
  - **`file`**: fetching a local file (e.g. on your computer)
  - You don't need to know the details about these protocols

`https://course.cse.ust.hk/comp2633/index.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://course.cse.ust.hk/comp2633/index.html

Parts of a URL: Location

- Location: The domain with some additional information
  - Username: **lamngok@cse.ust.hk**
    - Identifies one specific user on the web server
    - Rarely seen
  - Port: **course.cse.ust.hk:443**
    - Identifies one specific application on the web server
    - We will see ports again in the networking unit

https://course.cse.ust.hk:443/comp2633/index.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://course.cse.ust.hk/comp2633/index.html`**: Look in the **`comp2633`** folder for **`index.html`** file
  - **`https:// course.cse.ust.hk/`**: Return the root directory

`https://course.cse.ust.hk/comp2633/index.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://www.youtube.com/watch?v=8GZOAnR6GLk`

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
  - Usage: Supplies content to code in the web browser (JavaScript) without sending the content to the server

https://course.cse.ust.hk/comp2633/#lectures

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!)

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
- 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). They are typically used for getting data from the server
  - **POST**: Requests that update server-side state ("*post*" information to the server). They are typically used for passing and submitting data to be processed by the server.
- 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 Chrome 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://course.cse.ust.hk/comp2633">Check
out the course!</a>
```

Webpage

Check out the course!

Clicking on this text will take you to **https://course.cse.ust.hk/comp2633**

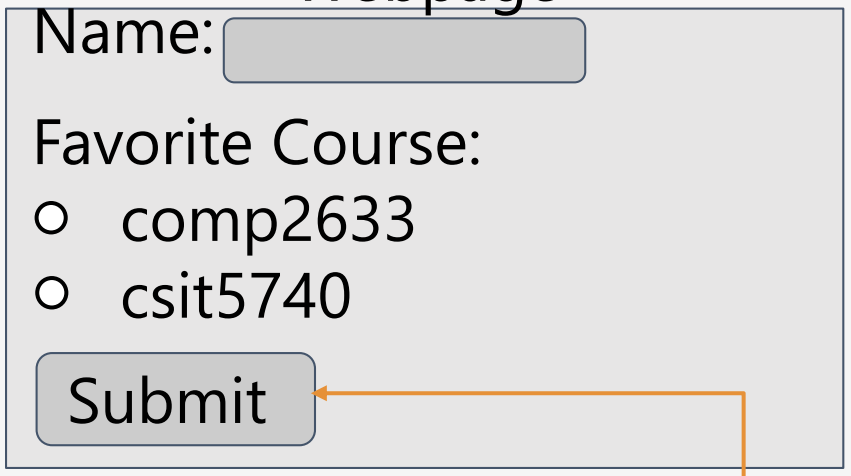
Features of HTML: Create a Form

HTML

```
<form action="/feedback" method="POST">
  <label for="name">Name:</label>
  <input type="text" id="name">
  <br>
  <label for="bot">Favorite
Course:</label><br>
  <input type="radio" id="comp2633">
  <label for="html">comp2633</label><br>
  <input type="radio" id="csit">
  <label for="css">csit5740</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://course.cse.ust.hk/feedback** with the contents of the form

Features of HTML: Embed an Image

HTML

```
<p>Look at my cartoon pic!</p>
<img src=" https://www.cse.ust.hk/~lamngok/images/1.jpg ">
```

Webpage

Look at my cartoon pic!

The browser will make a GET request to `https://www.cse.ust.hk/~lamngok/images/1.jpg` and display the returned image on the page.

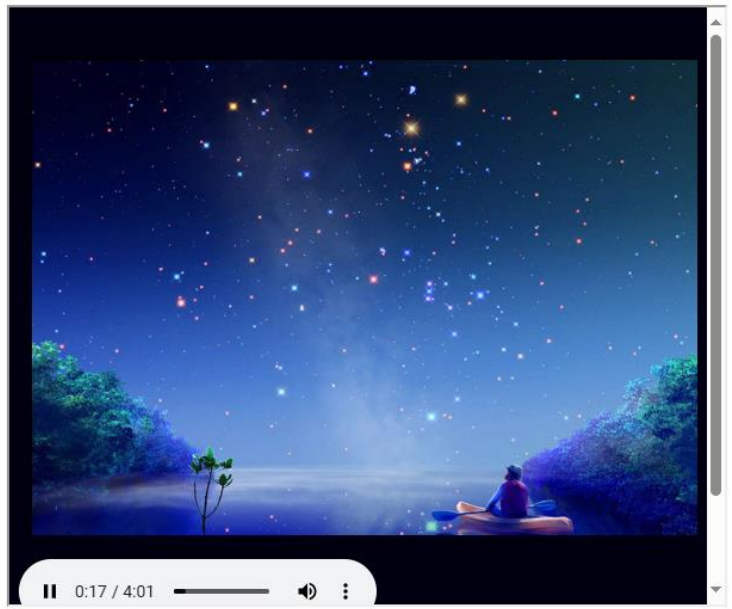
Features of HTML: Embed Another Webpage

HTML

```
<iframe
src="https://www.cse.ust.hk/~lamngok/music.html
" height="500" width="600"></iframe>
<p>Alex's music page above.</p>
```

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

Webpage



The browser will make a GET request to **`https://www.cse.ust.hk/`** and display the returned webpage in a 500 pixel × 600 pixel box.

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!

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 an evil website should not infect our computer with malware
  - Example: If we visit an evil website, the attacker who owns `it` 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 an evil website, the attacker who owns `it` should not be able to read our emails or buy things with our accounts
- Protection: Same-origin policy
  - The web browser prevents a website from accessing other unrelated websites

The 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 URL has an **origin** defined by three parts:
  - **Protocol**: The protocol in the URL
  - **hostname**: The host 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.cse.ust.hk:443/~lamngok/images/2.jpg`

`http://www.cse.ust.hk/~lamngok/images/2.jpg`

`80` (default port)

Same-Origin Policy

- Two URLs have the same origin *if and only if* the protocol, hostname, and port of the URLs all **match exactly**
  - Effecticve string matching

First URL	Second URL	Same origin?
https://www.cse.ust.hk/file.html	https://www.cse.ust.hk/~lamngok/files/test.html	YES, only paths are different
https://www.cse.ust.hk/file.html	https://www.cse.ust.hk/~alex	YES, only paths are different
https://www.cse.ust.hk/file.html	http://www.cse.ust.hk/anotherfile.html	No, protocols different (`http` ≠ `https`)
https://www.cse.ust.hk/file.html	https://www.cse.ust.hk:400/file.html	No, different ports (**http:// is port 443 by default**)
https://www.cse.ust.hk/file.html	https://course.cse.ust.hk/file.html	No, hostnames different (`www.cse.ust.hk` ≠ `course.cse.ust.hk`)

Same-Origin Policy

- Two websites with different origins cannot interact with each other
  - Example: If **`www.cse.ust.hk`** embeds **`google.com`** in an inner frame(iframe) the inner frame cannot interact with the outer frame, and the outer frame cannot interact with the inner-frame
- Exception: JavaScript
  - Javascript runs with the origin of the page that loads it
  - Example: If **`www.cse.ust.hk`** fetches JavaScript from **`google.com`**, the JavaScript has the origin of **`www.cse.ust.hk`**
  - Intuition: **`www.cse.ust.hk`** has “copy-pasted” JavaScript onto its webpage
- Exception: Images
  - 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
    - 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

HTTP: Summary

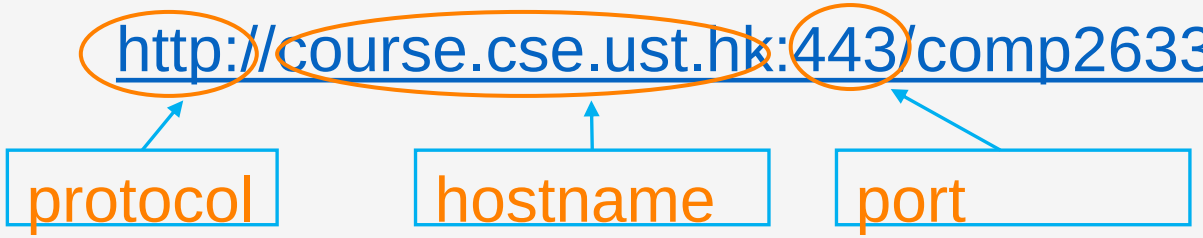
- HTTP: A protocol used to request and retrieve data from a web server
  - HTTPS: A secure version of HTTP
  - HTTP is a request-response protocol
- HTTP request
  - Method (GET or POST)
  - URL path and query parameters
  - Protocol
  - Data (only for POST requests)
- HTTP response
  - Protocol
  - 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),
    - an inline frame (iframe) is a HTML element that essentially puts another HTML page within the parent HTML page, as shown on slide 25. It is usually used for advertisements, embedded videos, etc
    - a frame on the other hand has its own contents
- CSS: A style sheet language for defining the appearance of webpages
  - As powerful as JavaScript if used maliciously!

Same-Origin Policy: Summary

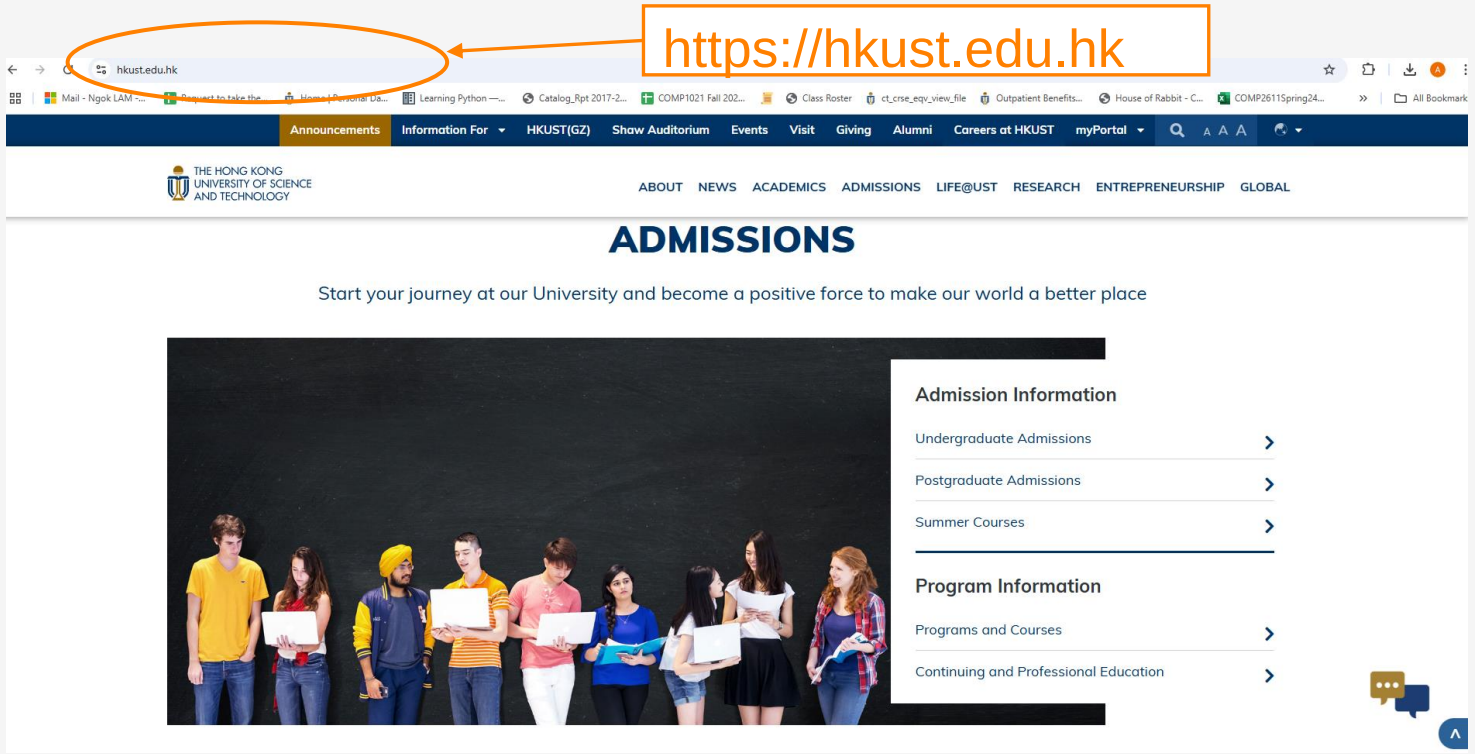
- Rule enforced by the browser: Two websites with different origins cannot interact with each other
- Two webpages have the same origin *if and only if* the protocol, domain, and port of the URL all match exactly (it is **effectively string matching**)



- 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

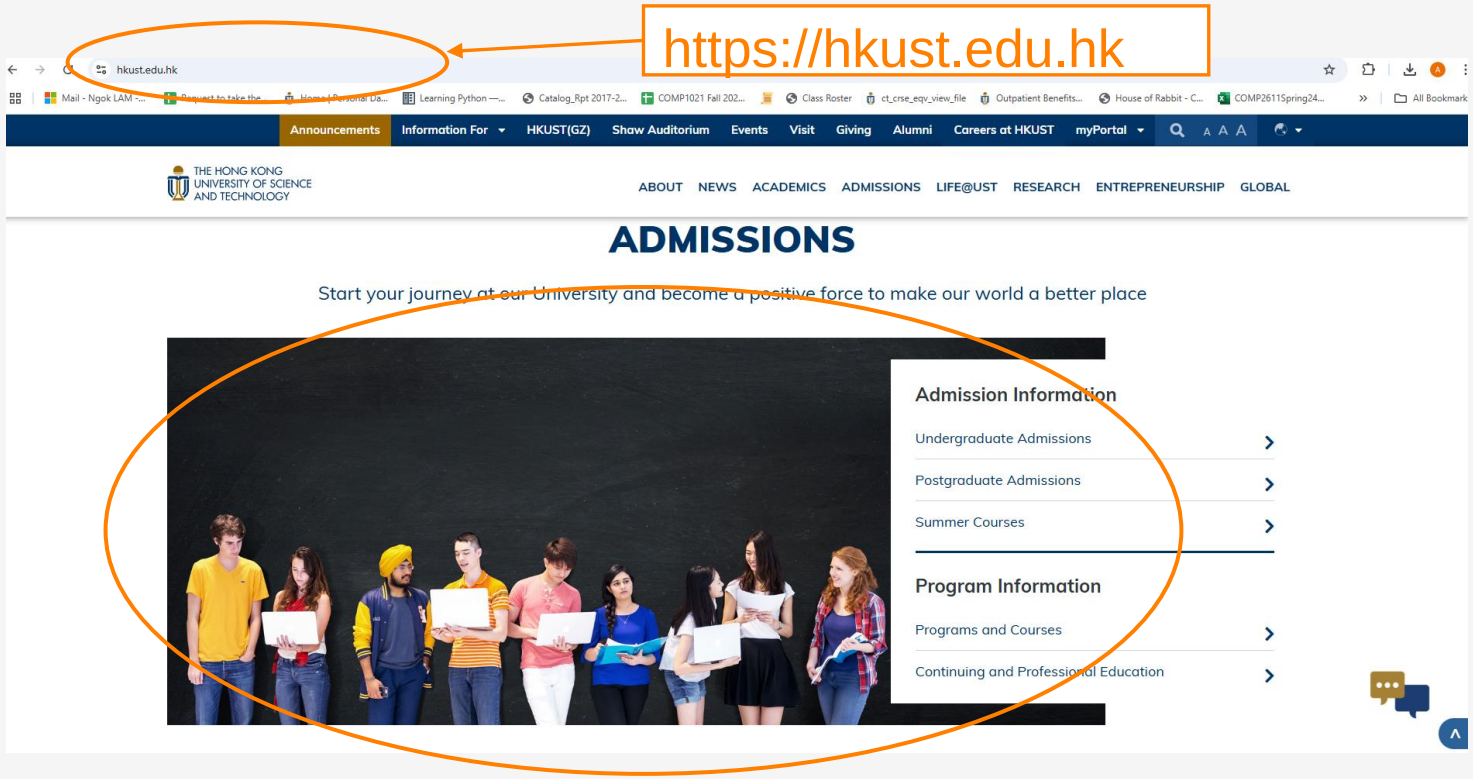
Same-Origin Policy: Webpage components

- The origin of a page is derived from the URL it was loaded from
- Special case: Javascript runs with the origin of the page that loaded it, if there is a Javascript embedded, it runs with the origin of https://hkust.edu.hk



Same-Origin Policy: Webpage components

- Special case: <img src=“”> even if the image is copied from a remote host, it has the origin of the embedded page (just like Javascript), but not origin of the remote host



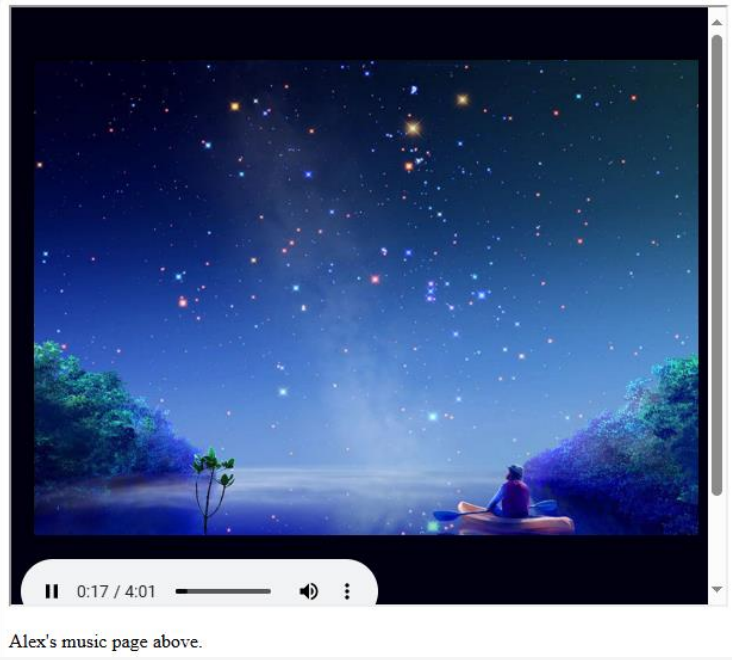
Same-Origin Policy: Webpage components

- iframe has the same origin as its original URL, not the origin of the HTML page it is loaded. For example, if the following iframe is in a HTML page at the origin

  https://course.cse.ust.hk/

```
<iframe
src="https://www.cse.ust.hk/~lamngok/music
.html" height="500" width="600"></iframe>
<p>Alex's music page above.</p>
```

  Then it will have the origin "**https://www.cse.ust.hk/**"



Javascript code always follows the origin of the frame or iframe embedding. If we have a Javascript code in the above iframe, it will have the origin **https://www.cse.ust.hk**, and cannot Modify files in the origin **https://course.cse.ust.hk**, even though that's the URL embedding it.

Same-Origin Policy: Webpage components

- A Javascript code always follows the origin of the frame or iframe embedding it. If we have a Javascript code in the HTML file below , it will have the origin **https://www.cse.ust.hk**, and cannot modify files at the origin **https://course.cse.ust.hk**, even though that's the URL embedding it.

```
<html>
:
<iframe
src="https://www.cse.ust.hk/~lamngok/music.html"
height="500" width="600"></iframe>
<p>Alex's music page above.</p>
:
</html>
```

HTML file at https://course.cse.ust.hk

Cookies

Cookies

- HTTP is stateless
- Using cookies is a way to add state. This state helps link the same user's requests and helps customize websites for the user

Cookies

A way of maintaining state in the browser

Browser maintains cookie container storing all cookies it has received

Setting/deleting cookies by server

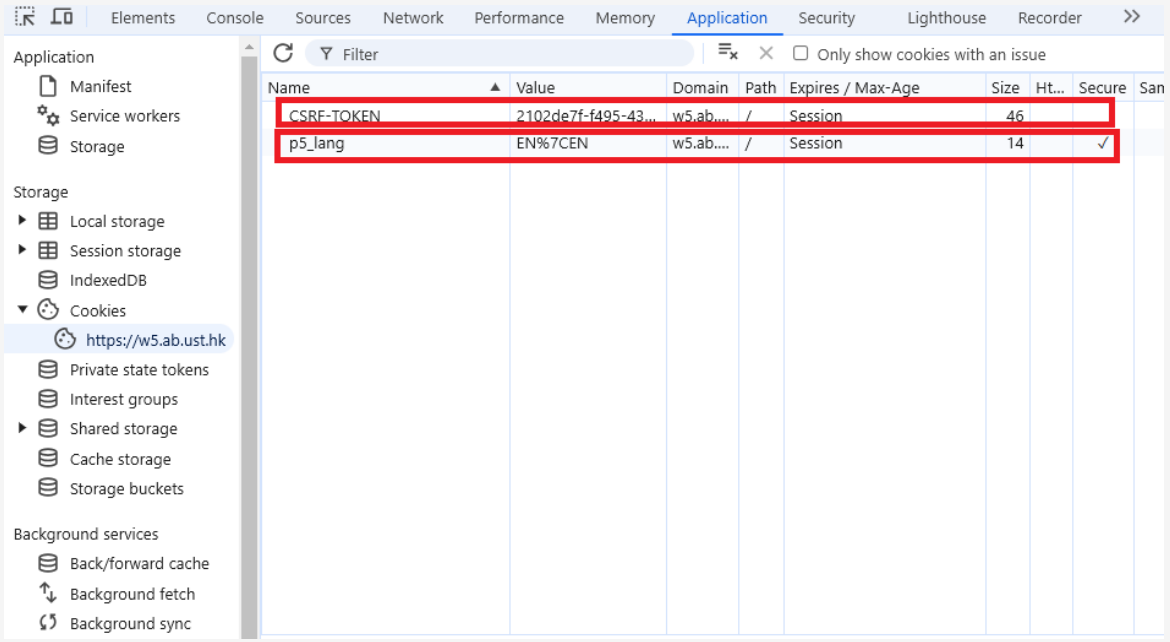
- The first time a browser connects to a particular web server, it has no cookies for that web server
- When the web server responds, it includes a **Set-cookie:** header that defines a cookie
- Each cookie is just a name-value pair (with some extra metadata)

Viewing a cookie

Chrome: More tools- > Developer tools, then in the web console, type

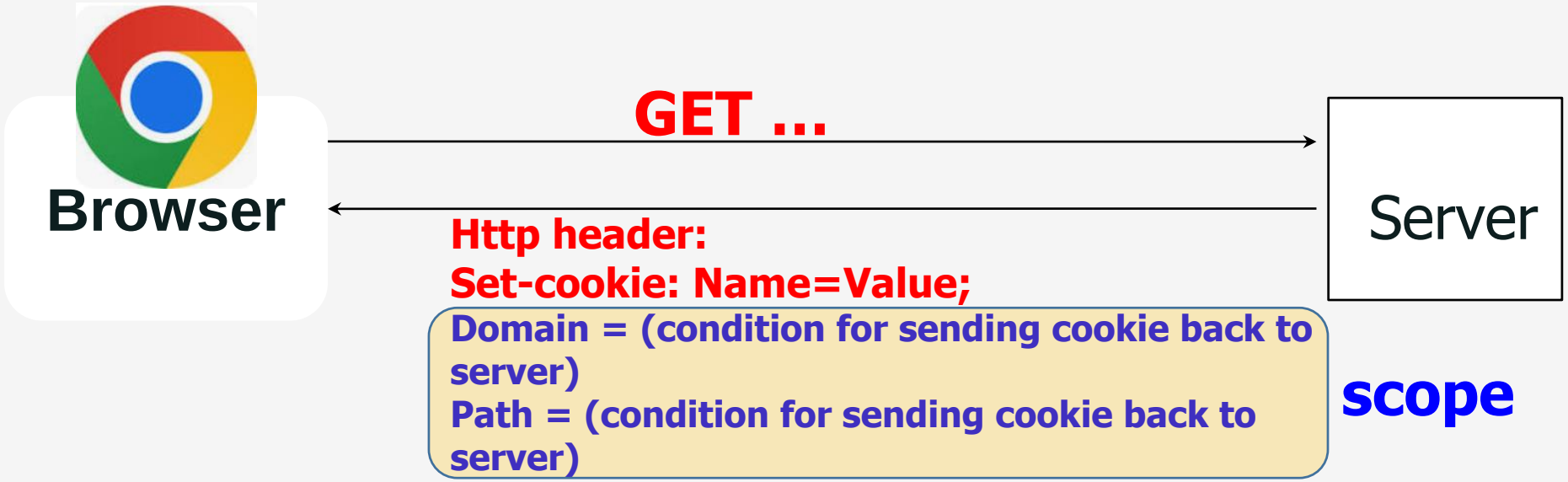
document.cookie

to see the cookie for that site



Each name=value is one cookie.
document.cookie lists all cookies in scope for document

Cookie scope

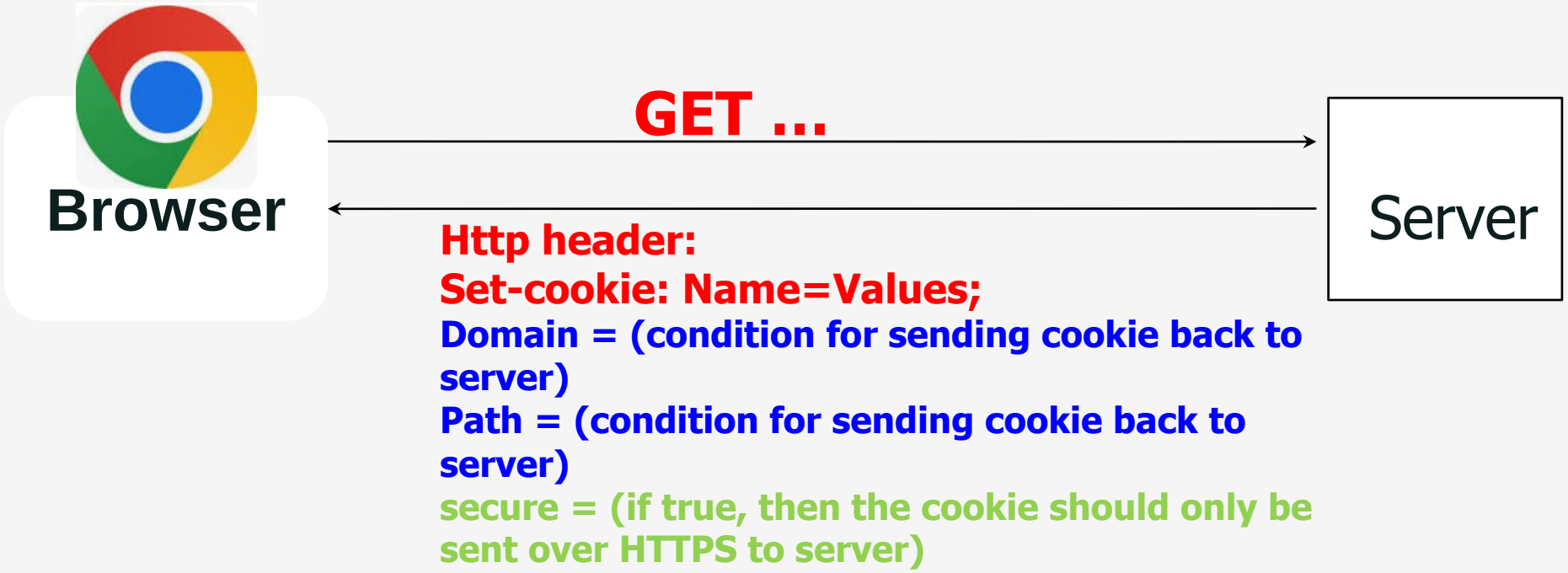


- The “Domain” and “Path” attributes define the scope of the cookie. They tell the browser what website the cookie belongs to.
- When the browser connects to the same server later, it uses the scope (Domain and Path) to determine whether it should send back the cookie and automatically attaches the appropriate cookie(s).

Cookie scope

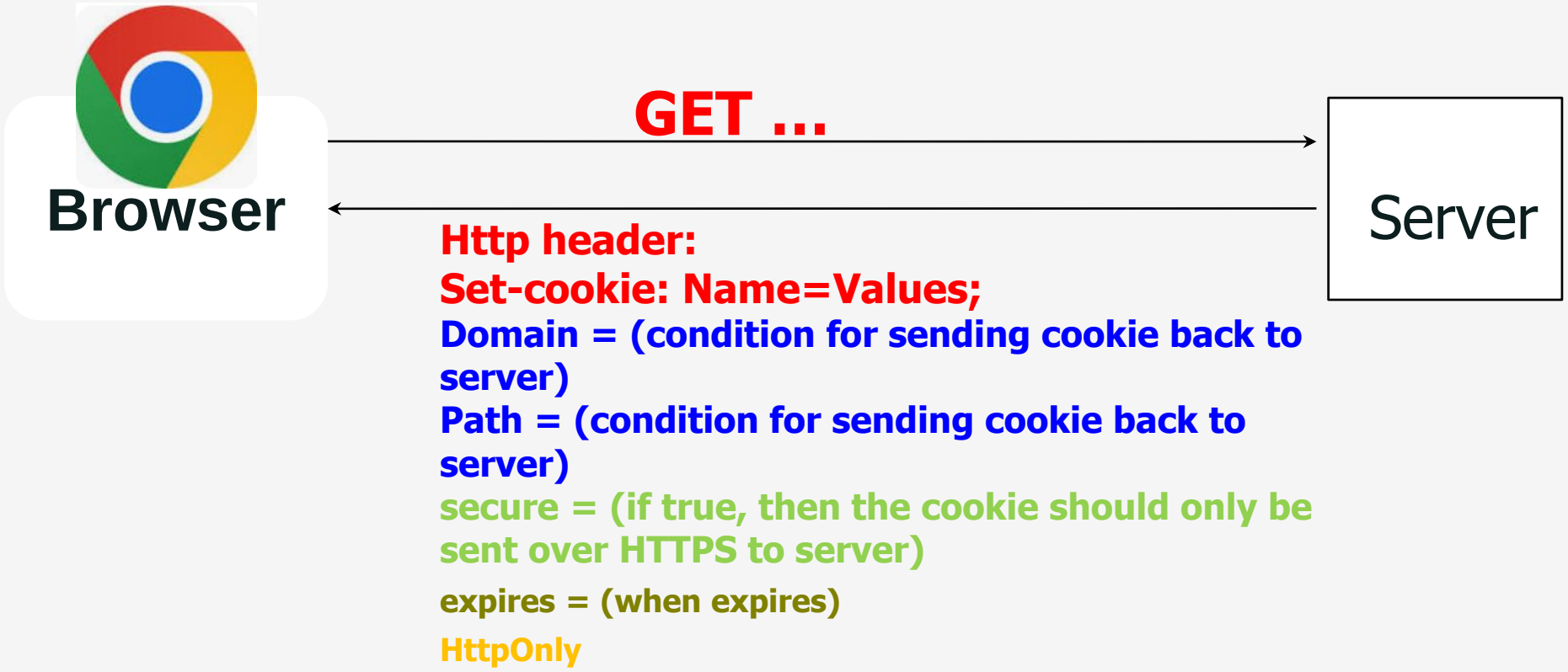
- The browser only sends the cookie's **name** and **value** to the server, the browser does not send other attributes.
- The server can use the **name** and **value** to connect related requests from the browser

Cookie scope



- The Secure attribute indicates the cookie should be sent back to server over https only

Cookie scope



- **The "expires" attribute indicates expiration time**
  - Delete cookie by setting "expires" to date in past, for example 1st of Jan, 1970
- **HttpOnly: cookie cannot be accessed by Javascript, but only sent by browser**

Cookie policy

The cookie policy governs how the web server and the browser should behave:

- First it defines the condition(s) that a **Server under a particular hostname** is allowed to set on (send) a cookie
- Second it defines the condition(s) for **the browser** to send (back) a cookie to a server

Cookie policy: for the server

- The Domain attribute in a cookie sent might not be exactly the same as the hostname of the web server sending it.
- But for security reasons, we don't want a malicious website to be able to set (send) a cookie with a Domain it does not belong to (for example we don't want evil.com to be able to set cookie with ust.hk as its Domain). Because this would allow an attacker to affect the functionality of the legitimate websites.
- To prevent this, the cookie policy specifies that when a server sets a cookie, the cookie's Domain must be a **URL suffix of the server's URL**. In other words, for the cookie to be set, the server's URL must end in the cookie's Domain attribute. Otherwise, the browser will reject the cookie.

Cookie policy: for the server

- Basically the Domain attribute in a cookie could be set (sent) by any server if its URL-hostname **contains the domain (except for the Top Level Domains, TLDs, such as ".com").** In other words, the Domain must be **URL suffix of the server's URL**.This is illustrated using the example below:

  example: **URL-hostname = "www.example.com"**

allowed domains	**disallowed domains examples**
www.example.com	**user.example.com**
example.com	**othersite.com**
	.com

Cookie policy: for the server

- Basically the Domain attribute in a cookie could be set (sent) by any server if its URL-hostname **contains the domain (except for the Top Level Domains, TLDs, such as ".com").** In other words, the Domain must be **URL suffix of the server's URL**.This is illustrated using the example below:

  example: **URL-hostname = "www.example.com"**

⇒ **www.example.com** can set cookies for **.example.com** but not for another site or **TLD**

In other words cse.ust.hk can set cookies using .ust.hk

path: can be set to anything

Examples

Web server at **foo.example.com** wants to set cookie with domain:

Setting the Domain with	Allowed ?	Remark
bar.foo.example.com	No	Domain not a URL suffix of foo.example.com
Foo.example.com	Yes	
bar.example.com	No	Domain not a URL suffix of foo.example.com
example.com	Yes	
ample.com	No	Domain name mis-matched
.com	No	Though domain is a URL suffix of foo.example.com, but this is not allowed due to security consideration

Cookie policy

The cookie policy governs how the web server and the browser should behave:

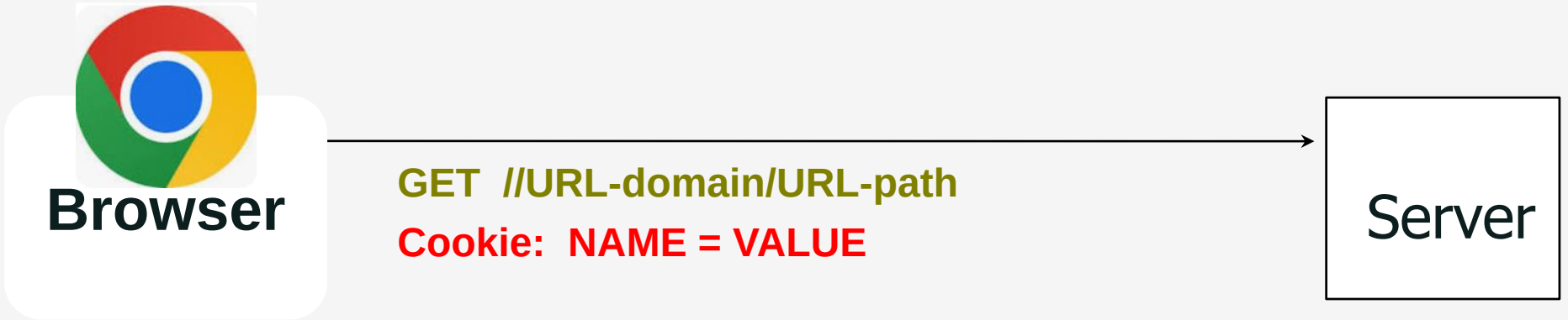
- First it defines the condition(s) that a **Server under a particular hostname** is allowed to set on (send) a cookie
- Second it defines the condition(s) for **the browser** to send (back) a cookie to a server

Cookie policy: for the browser

Browser sends all cookies in URL scope:

- Domain is **suffix of server's URL**, and
- Path is **prefix of the URL-path**, and
- whenever the Secure attribute is true [protocol=HTTPS]

Cookie policy: for the browser



A cookie with

Domain = example.com, and

Path = /some/path/

will be included on a request to

http://foo.example.com/some/path/subdirectory/hello.txt

Tricky Examples

cookie 1
name = userid
value = u1
domain = login.site.com
path = /
secure

cookie 2
name = userid
value = u2
domain = .site.com
path = /
non-secure

http://checkout.site.com/	**cookie: userid=u2**
http://login.site.com/	**cookie: userid=u2**
https://login.site.com/	**cookie: userid=u1; userid=u2**

Client side read/write: document.cookie

- Setting a cookie in Javascript:

  document.cookie = “name=value; expires=…; ”

- Reading a cookie: alert(document.cookie)

  prints string containing all cookies available for document (based on [protocol], domain, path)

- Deleting a cookie:

  document.cookie = “name=; expires= Thu, 01-Jan-00”

document.cookie often used to customize page in Javascript

Cookie policy vs the same-origin policy

Cookie policy versus same-origin policy

- Cookie policy is not the same as the same-origin policy:
  - Consider Javascript on a page loaded from a URL U
  - If a cookie is in scope for a URL U, it can be accessed by Javascript loaded on the page with URL U,

    unless the cookie has the httpOnly flag set.

The same-origin policy requires **exact matching** in the hostnames, but the cookie policy only need the **suffix of the hostname to match with the domain** in the cookie

Examples

cookie 1
name = userid
value = u1
domain = login.site.com
path = /
non-secure

cookie 2
name = userid
value = u2
domain = .site.com
path = /
non-secure

http://checkout.site.com/	**cookie: userid=u2**
http://login.site.com/	**cookie: userid=u1, userid=u2**
http://othersite.com/	**cookie: none**

JS on each of these URLs can access the corresponding cookies even if the hostnames are not always the same

Indirectly bypassing same-origin policy using cookie policy

- Since the cookie policy and the same-origin policy are different, there are corner cases when one can use cookie policy to bypass same-origin policy
- Ideas how?

Example

Victim user browser

Cookies from:

financial.example.com

blog.example.com

(assume attacker compromised the web server blog.example.com)

cookies with
Domain = example.com

The browser will send the cookie for financial.example.com to blog.example.com because Domain = example.com

The cookie document: RFC6265

- For further details on cookies, checkout the standard RFC6265 “HTTP State Management Mechanism”

https://tools.ietf.org/html/rfc6265

- Browsers are expected to implement this reference, and any differences are browser specific