

**CSE 484 / CSE M 584: Computer Security and  
Privacy**

# **Web Security**

Autumn 2018

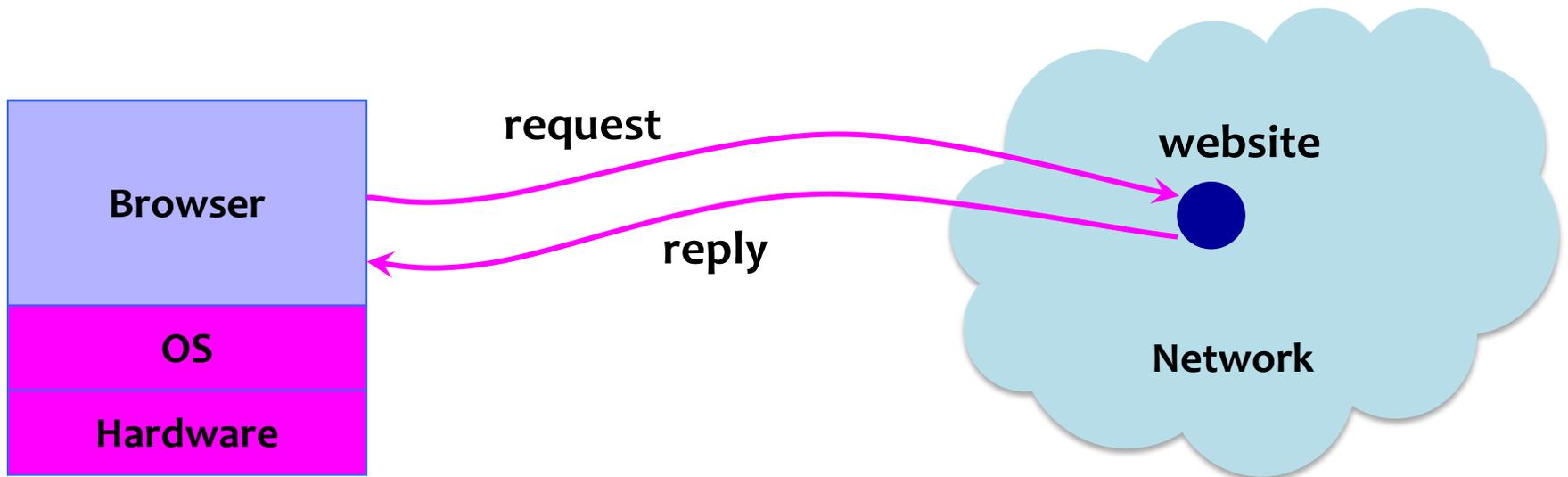
Tadayoshi (Yoshi) Kohno  
[yoshi@cs.Washington.edu](mailto:yoshi@cs.Washington.edu)

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

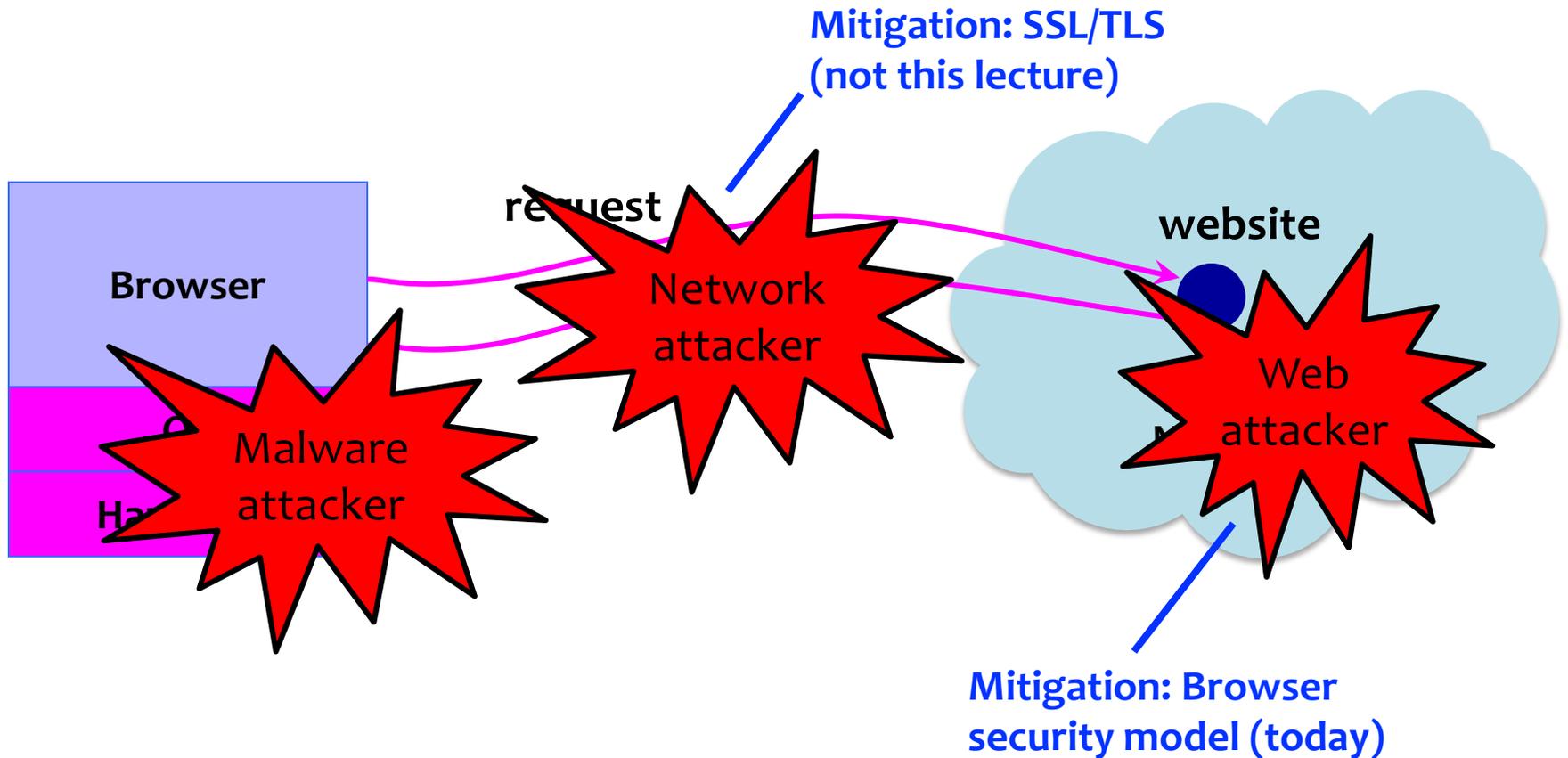
# Admin

- HW2: Due Nov 7, 4:30pm
- Looking ahead, rough plan:
- Lab 2 out ~Nov 5, due ~Nov 19 (Quiz Section on Nov 8)
- HW 3 out ~Nov 19, due ~Nov 30
- Lab 3 out ~Nov 26, due Dec 7 (Quiz Section on Nov 29)
- HW1s were **awesome**

# Big Picture: Browser and Network



# Where Does the Attacker Live?



# Web Attacker

- Controls a malicious website (**attacker.com**)
  - Can even obtain SSL/TLS certificate for site 
- User visits attacker.com – why?
  - Phishing email, enticing content, search results, placed by an ad network, blind luck ...
- Attacker has no other access to user machine!
- Variation: good site **honest.com**, but:
  - An iframe with malicious content included
  - Website has been compromised

# 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, Ruby, ASP, JSP
  - Client-side code written in JavaScript
- Many potential bugs: XSS, XSRF, SQL injection

# All of These Should Be Safe

- Safe to visit an evil website



- Safe to visit two pages at the same time



- 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  
(plus sandbox)



# 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 [1]
Sandbox Escape [5]	\$15,000

From Chrome Bug Bounty Program

# Same Origin Policy

Goal: Protect/isolate web content from other web content

Website origin = (scheme, domain, port)

Compared URL	Outcome	Reason
<a href="http://www.example.com/dir/page.html">http://www.example.com/dir/page.html</a>	Success	Same protocol and host
<a href="http://www.example.com/dir2/other.html">http://www.example.com/dir2/other.html</a>	Success	Same protocol and host
<a href="http://www.example.com:81/dir/other.html">http://www.example.com:81/dir/other.html</a>	Failure	Different origin
<a href="https://www.example.com/dir/other.html">https://www.example.com/dir/other.html</a>	Failure	
<a href="http://en.example.com/dir/other.html">http://en.example.com/dir/other.html</a>	Failure	
<a href="http://example.com/dir/other.html">http://example.com/dir/other.html</a>	Failure	
<a href="http://v2.www.example.com/dir/other.html">http://v2.www.example.com/dir/other.html</a>	Failure	

[Example from Wikipedia]

# Same Origin Policy

Goal: Protect/isolate web content from other web content

Website origin = (scheme, domain, port)

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

[Example from Wikipedia]

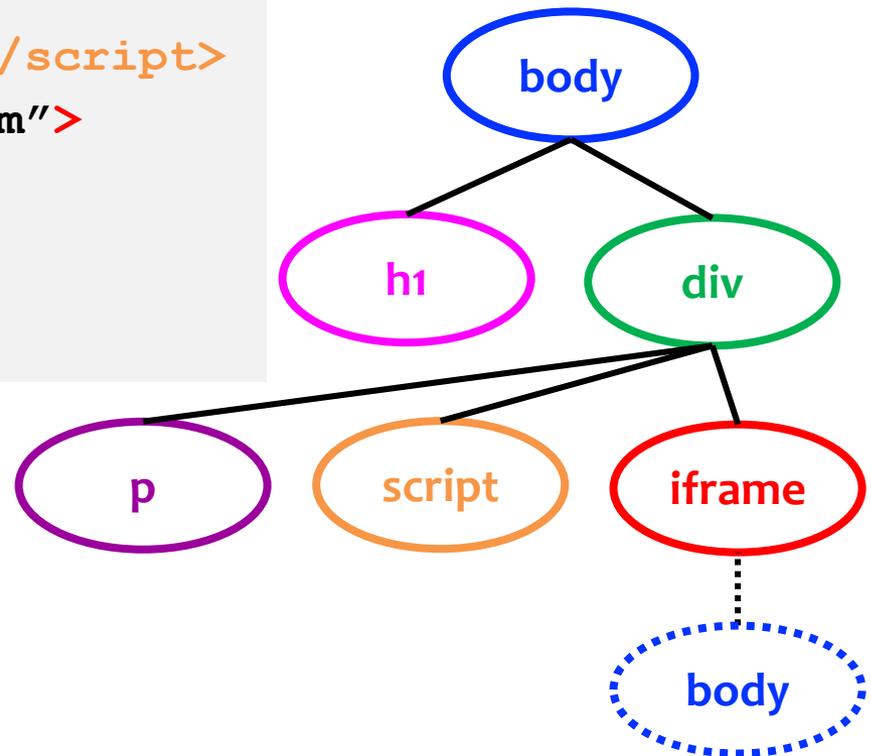
# Same Origin Policy is Subtle!

- Some examples of how messy it gets in practice...
- Browsers don't (or didn't) always get it right...
- We'll talk about:
  - DOM / HTML Elements
  - Navigation
  - Cookie Reading
  - Cookie Writing
  - Iframes vs. Scripts

# HTML + Document Object Model

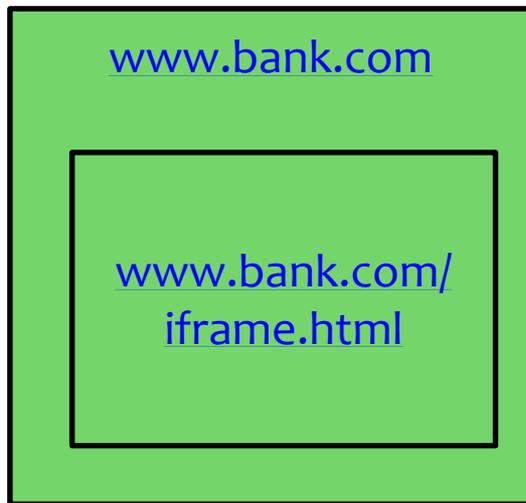
```
<html> <body>
<h1>This is the title</h1>
<div>
<p>This is a sample page.</p>
<script>alert("Hello world");</script>
<iframe src="http://example.com">
</iframe>
</div>
</body> </html>
```

Document Object Model (DOM)

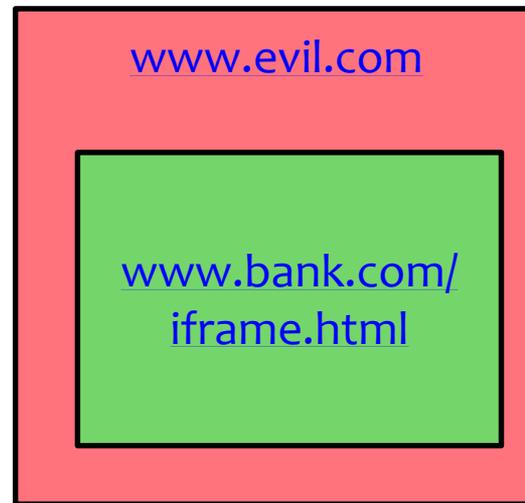


# Same-Origin Policy: DOM

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



[www.bank.com](http://www.bank.com) (the parent) **can** access HTML elements in the iframe (and vice versa).



[www.evilm.com](http://www.evilm.com) (the parent) **cannot** access HTML elements in the iframe (and vice versa).

# Question: Who Can Navigate a Frame?

Older Issue



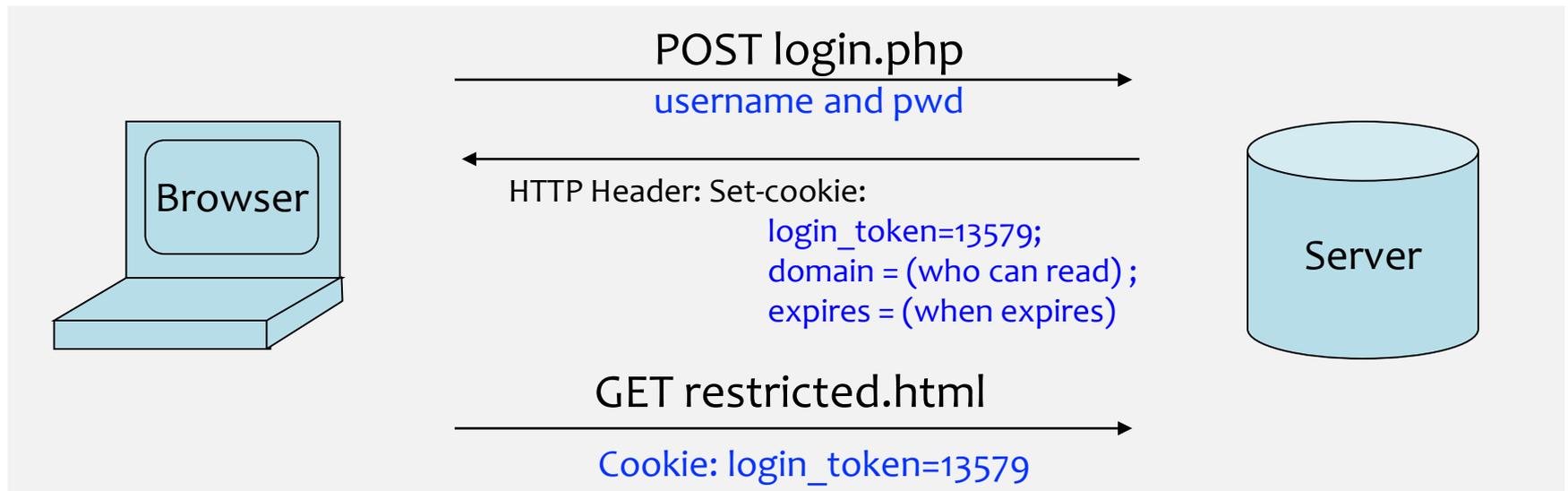
**Solution:** Modern browsers only allow a frame to navigate its “descendent” frames



If bad frame can **navigate** sibling frames, attacker gets password!

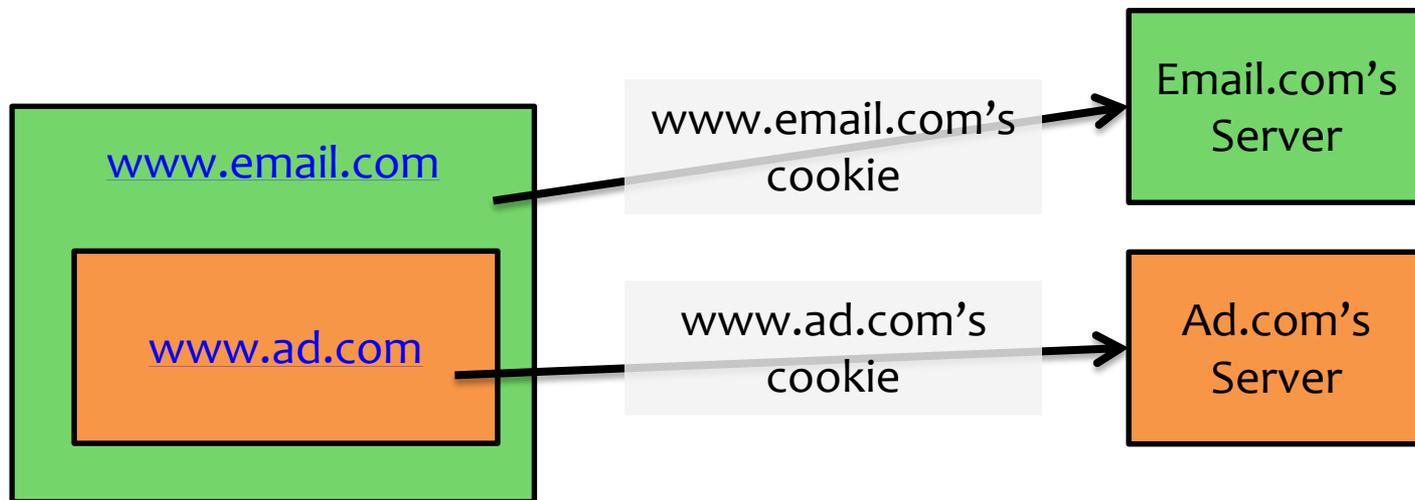
# Browser Cookies

- 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 😊



# Same Origin Policy: Cookie Writing

Which cookies can be set by **login.site.com**?

## allowed domains

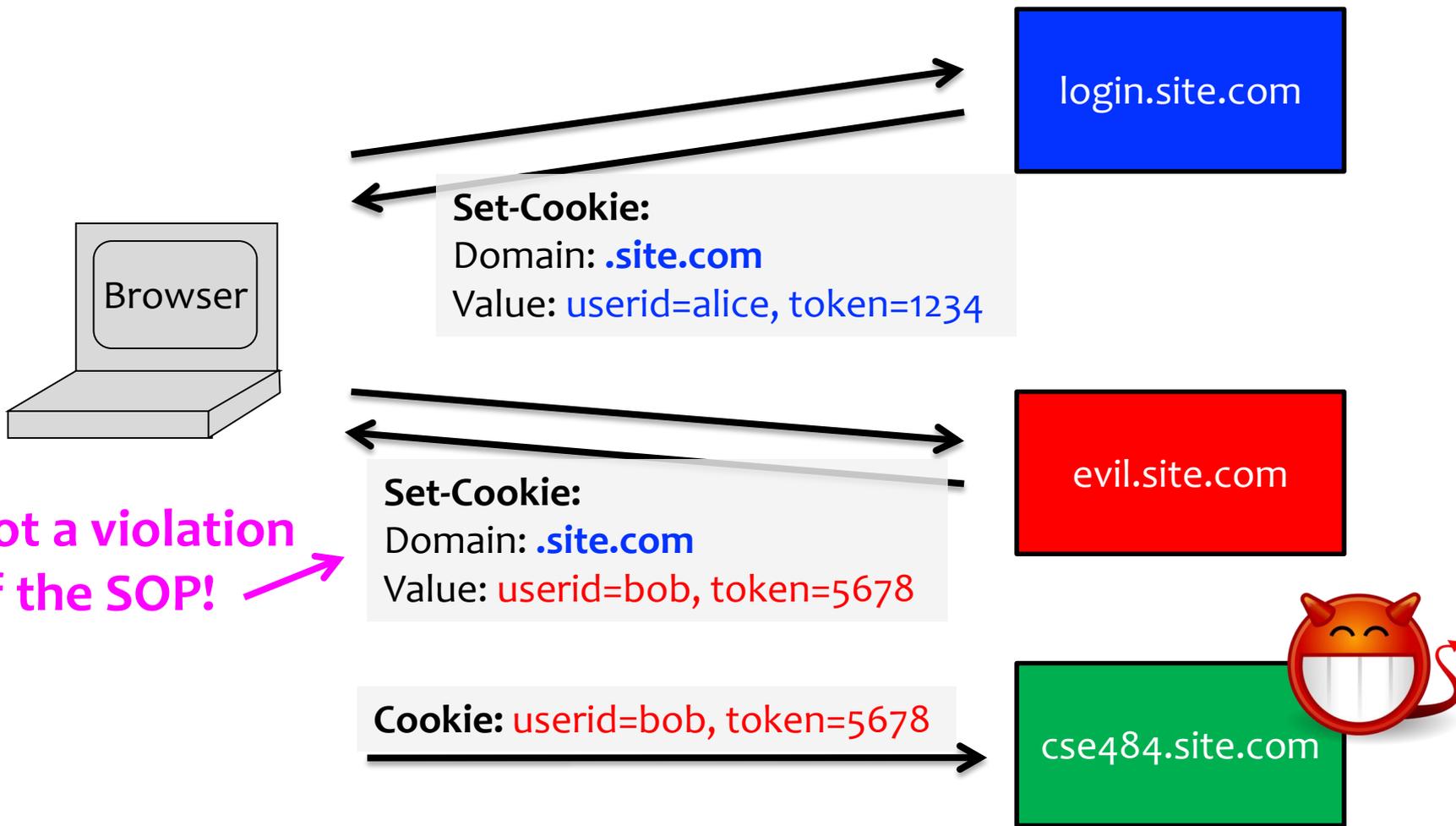
- ✓ **login.site.com**
- ✓ **.site.com**

## disallowed domains

- ✗ **othersite.com**
- ✗ **.com**
- ✗ **user.site.com**

**login.site.com** can set cookies for all of **.site.com (domain suffix)**, but not for another site or top-level domain (TLD)

# Problem: Who Set the Cookie?



Not a violation of the SOP!

# Same-Origin Policy: Scripts

# Same-Origin Policy: Scripts

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

```
www.example.com  
  
<script  
src="http://otherdomain  
.com/library.js">  
</script>
```

The code from <http://otherdomain.com> **can** access HTML elements and cookies on [www.example.com](http://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: leak info via image

```

```

# Cross-Origin Communication

- Sometimes you want to do it...
- Cross-origin network requests
  - Access-Control-Allow-Origin: <list of domains>
    - Unfortunately, often:  
Access-Control-Allow-Origin: \*
- Cross-origin client side communication
  - HTML5 postMessage between frames
    - Unfortunately, many bugs in how frames check sender's origin

# What about Browser Plugins?

- **Examples:** Flash, Silverlight, Java, PDF reader
- **Goal:** enable functionality that requires transcending the browser sandbox
- **Increases browser's attack surface**

## Java and Flash both vulnerable—again—to new 0-day attacks

Java bug is actively exploited. Flash flaws will likely be targeted soon.

by Dan Goodin (US) - Jul 13, 2015 9:11am PDT

- **Good news:** plugin sandboxing improving, and need for plugins decreasing (due to HTML5 and extensions)

# News: Goodbye Flash

## Get ready to finally say goodbye to Flash — in 2020

Posted Jul 25, 2017 by [Frederic Lardinois \(@fredericl\)](#)



Next Story

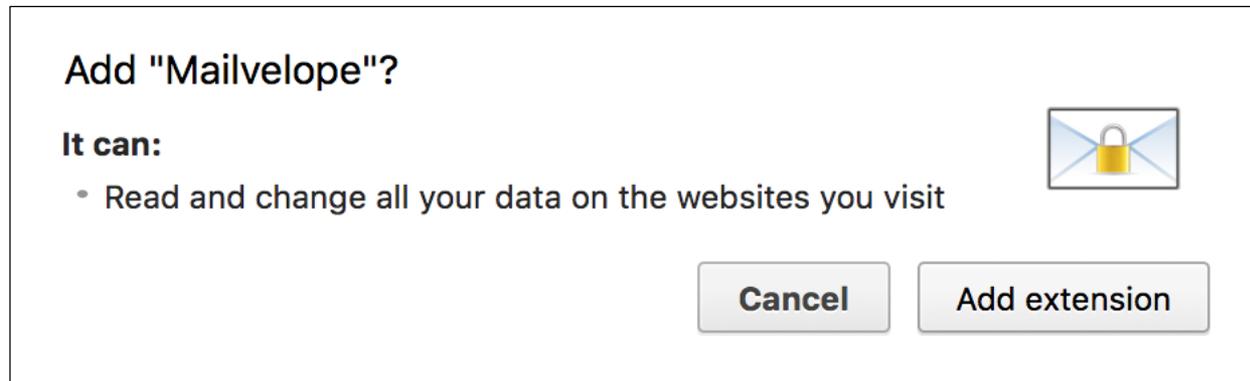


# What about Browser Extensions?

- Most things you use today are probably extensions
- **Examples:** Adblock, Ghostery, Mailvelope
- **Goal:** Extend the functionality of the browser
  
- (Chrome:) Carefully designed security model to **protect from malicious websites**
  - **Privilege separation:** extensions consist of multiple components with well-defined communication
  - **Least privilege:** extensions request permissions

# What about Browser Extensions?

- But be wary of malicious extensions: **not subject to the same-origin policy** – can inject code into any webpage!



# Summary of Overview

- Browser security model (thus far)
  - **Browser sandbox**: isolate web from local machine
  - **Same origin policy**: isolate web content from different domains
  - Also: Isolation for **plugins and extensions**
- Web application security (upcoming)
  - How (not) to build a secure website