

CSE 484 / CSE M 584: Web Security: CSRF and Browser Security Model

Fall 2023

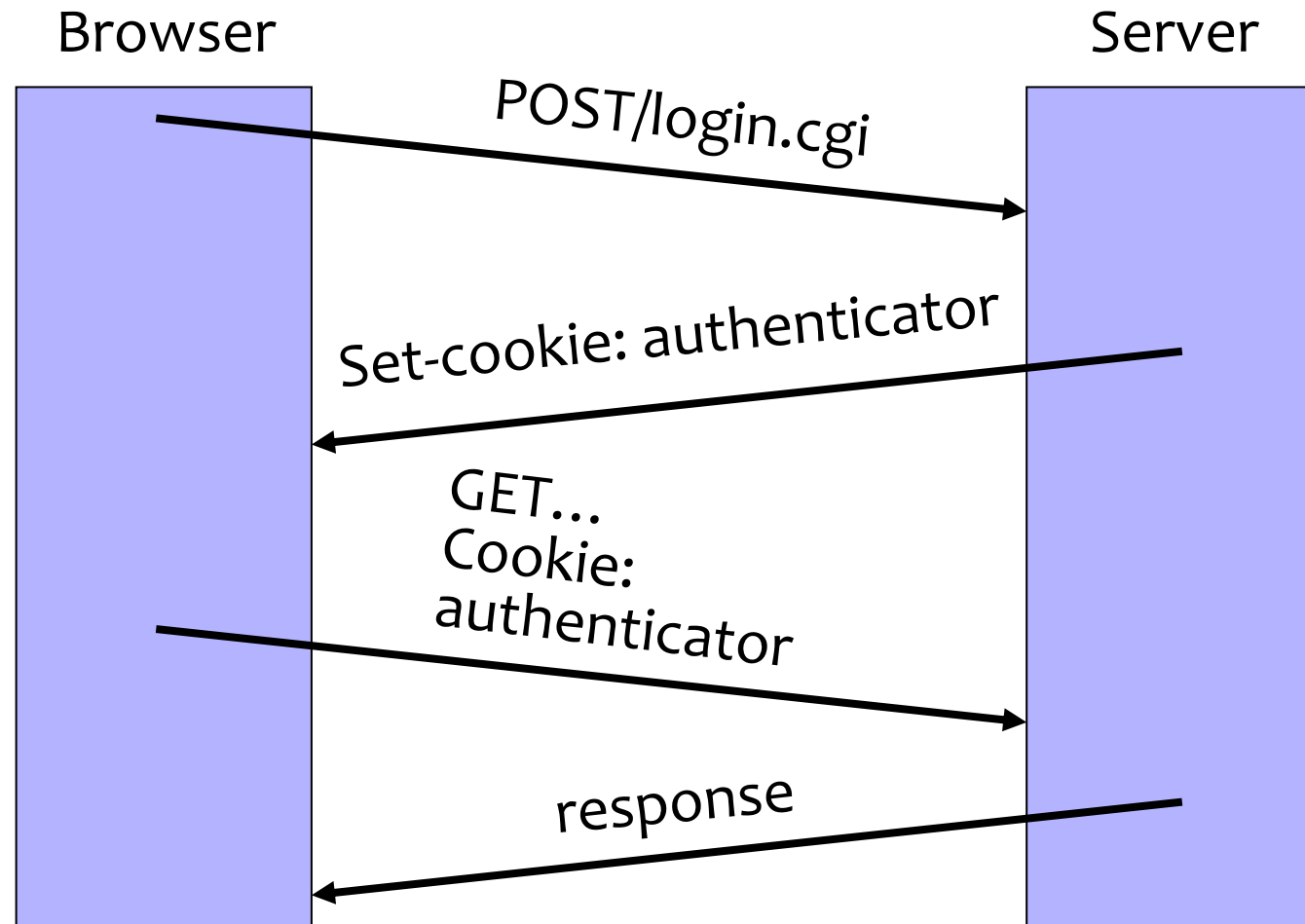
Franziska (Franzi) Roesner
franzi@cs

Announcements

- Friday is a holiday, no class! (Veterans' Day)
- Coming up:
 - **Lab 2 (ongoing)** (due end of next week)
 - **Homework 3** (out next week, due after Thanksgiving)
 - Some “hands on” experience with security tools
 - Don't expect this to take a huge amount of time
 - **Final project** (due during finals week)
 - Out after Thanksgiving
 - Vulnerability reports and patching

Cross-Site Request Forgery (CSRF/XSRF)

Cookie-Based Authentication Review



Same Origin Policy Review

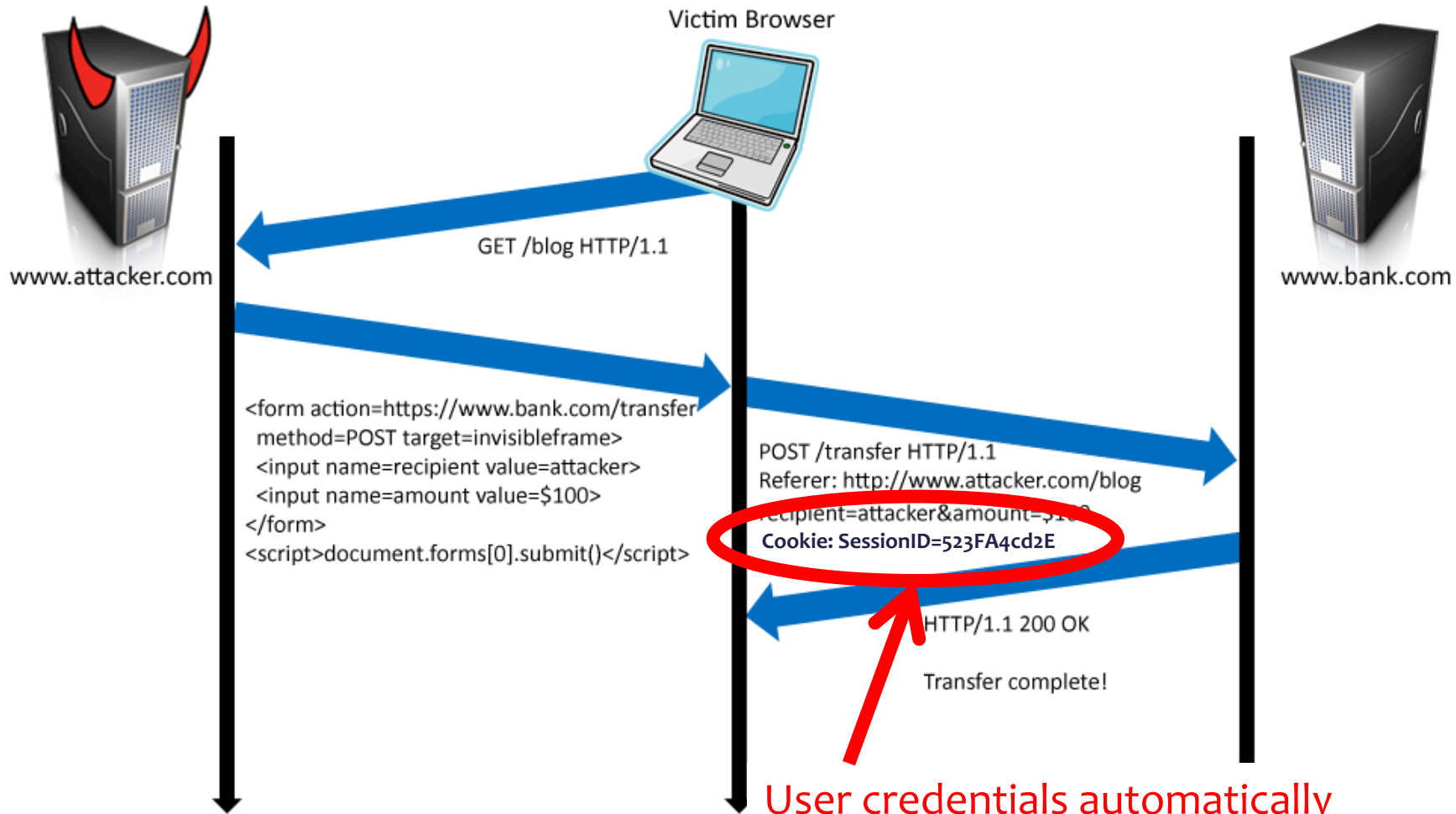
- SOP prevents cross-origin requests, DOM accesses, etc.
- **But: Active content (scripts) can send anywhere!**
 - For example, can submit a POST request
 - Some ports inaccessible -- e.g., SMTP (email)
- Can only *read* response from the *same origin*
 - ... but you can do a lot with just sending!

Cross-Site Request Forgery

- Users logs into bank.com, forgets to sign off
 - Session cookie remains in browser state
- User then visits a malicious website containing

```
<form name=BillPayForm action=http://bank.com/BillPay.php>  
<input name=recipient value=attacker> ...  
  
<script> document.BillPayForm.submit(); </script>
```
- Browser sends cookie, payment request fulfilled!
- Lesson: cookie authentication is not sufficient when side effects can happen

Cookies in Forged Requests



User credentials automatically sent by browser

Sending a Cross-Domain POST

```
<form method="POST" action=http://othersite.com/action >
```

```
...
```

```
</form>
```

```
<script>document.forms[0].submit()</script>
```

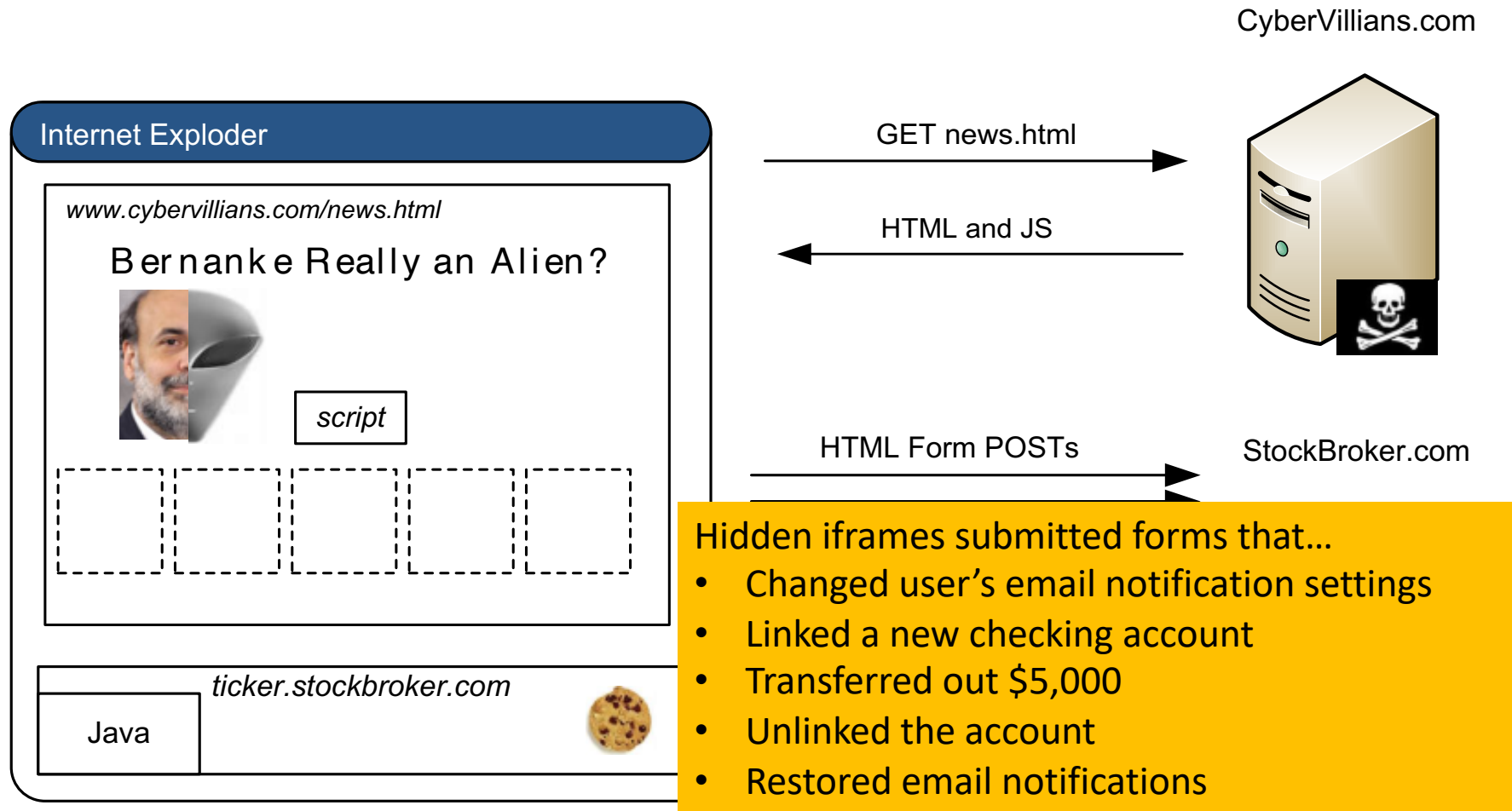
← submit post

- Hidden iframe can do this in the background
- User visits a malicious page, **browser submits form on behalf of user**

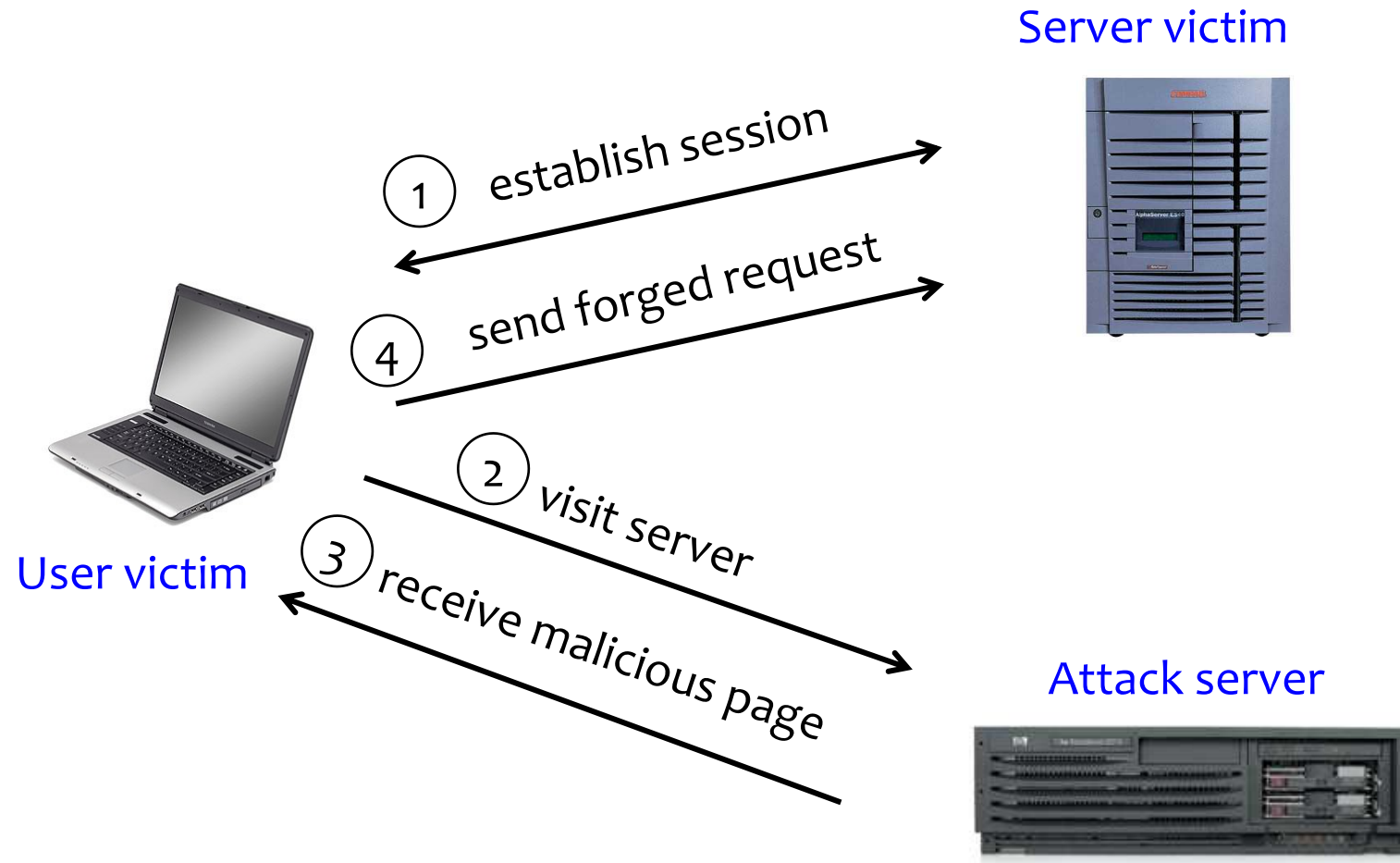
Impact

- Hijack any ongoing session (if no protection)
 - Netflix: change account settings, Gmail: steal contacts, Amazon: one-click purchase
- Reprogram the user's home router
- Login to the *attacker's* account
 - Why might an attacker want this?

XSRF True Story [Alex Stamos]



XSRF (aka CSRF): Summary



Q: how long do you stay logged on to Gmail? Financial sites?

Broader View of XSRF

- Abuse of cross-site data export
 - SOP does not control data export
 - Malicious webpage can initiate requests from the user's browser to an honest server
 - Server thinks requests are part of the established session between the browser and the server (automatically sends cookies)

Canvas Activity

How might a web application defend itself against CSRF?

XSRF Defenses

- Secret validation token



```
<input type=hidden value=23a3af01b>
```

- Referer validation



```
Referer:  
http://www.facebook.com/home.php
```

Referer Validation

Facebook Login

For your security, never enter your Facebook password on sites not located on Facebook.com.

Email:

Password:

Remember me

[Login](#) or [Sign up for Facebook](#)

[Forgot your password?](#)



Referer:
`http://www.facebook.com/home.php`



Referer:
`http://www.evil.com/attack.html`



Referer:

- **Lenient** referer checking – header is optional
- **Strict** referer checking – header is required

Why Not Always Strict Checking?

- Why might the referer header be suppressed?
 - Stripped by the organization's network filter
 - Stripped by the local machine
 - Stripped by the browser for HTTPS → HTTP transitions
 - User preference in browser
 - Buggy browser
- Web applications can't afford to block these users
- **Many web application frameworks include CSRF defenses today**

Better Idea: Add Secret Token to Forms

```
<input type=hidden value=23a3af01b>
```

- “Synchronizer Token Pattern”
- Include a **secret challenge token** as a hidden input in forms
 - Token often based on user’s session ID
 - Server must verify correctness of token before executing sensitive operations
- Why does this work?
 - **Same-origin policy**: attacker can’t read token out of legitimate forms loaded in user’s browser!
 - *So: can’t create fake forms with correct token!*

Stepping Back: 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

Review: Browser Security Model

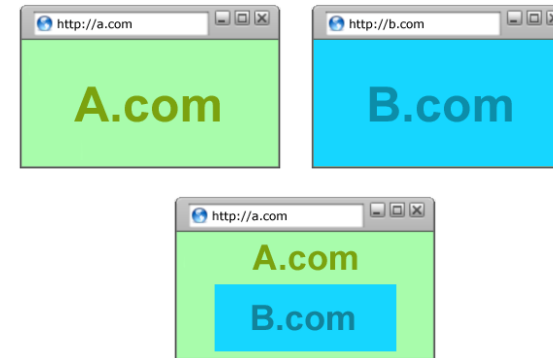
Goal 1: Protect local system from web attacker

→ Browser Sandbox



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

→ Same Origin Policy



Cross-Origin Communication

- Sometimes you want to do it...
- Cross-origin Resource Sharing (CORS)
 - 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)

Goodbye Flash

Get ready to finally say goodbye to Flash — in 2020

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



Next Story



“As of mid-October 2020, users started being prompted by Adobe to uninstall Flash Player on their machines since Flash-based content will be blocked from running in Adobe Flash Player after the EOL Date.”

<https://www.adobe.com/products/flashplayer/end-of-life.html>

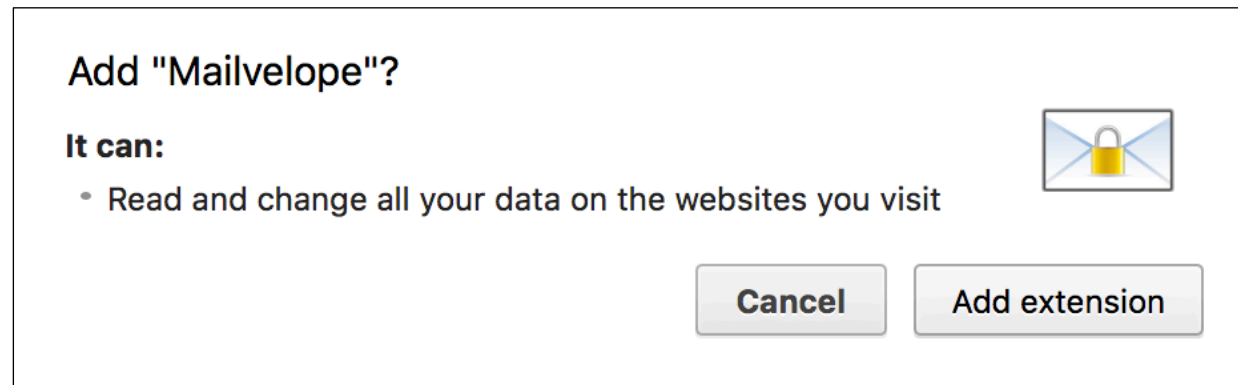
What about Browser Extensions?

- 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!



- Today: Extensions in flux – new “Manifest v3” specification from Google, trying to make things safer.

Web Security Summary

- Browser security model
 - **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
 - How (not) to build a secure website
- Next lecture: web **privacy**