CSE 484 / CSE M 584: Web Security

Winter 2023

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

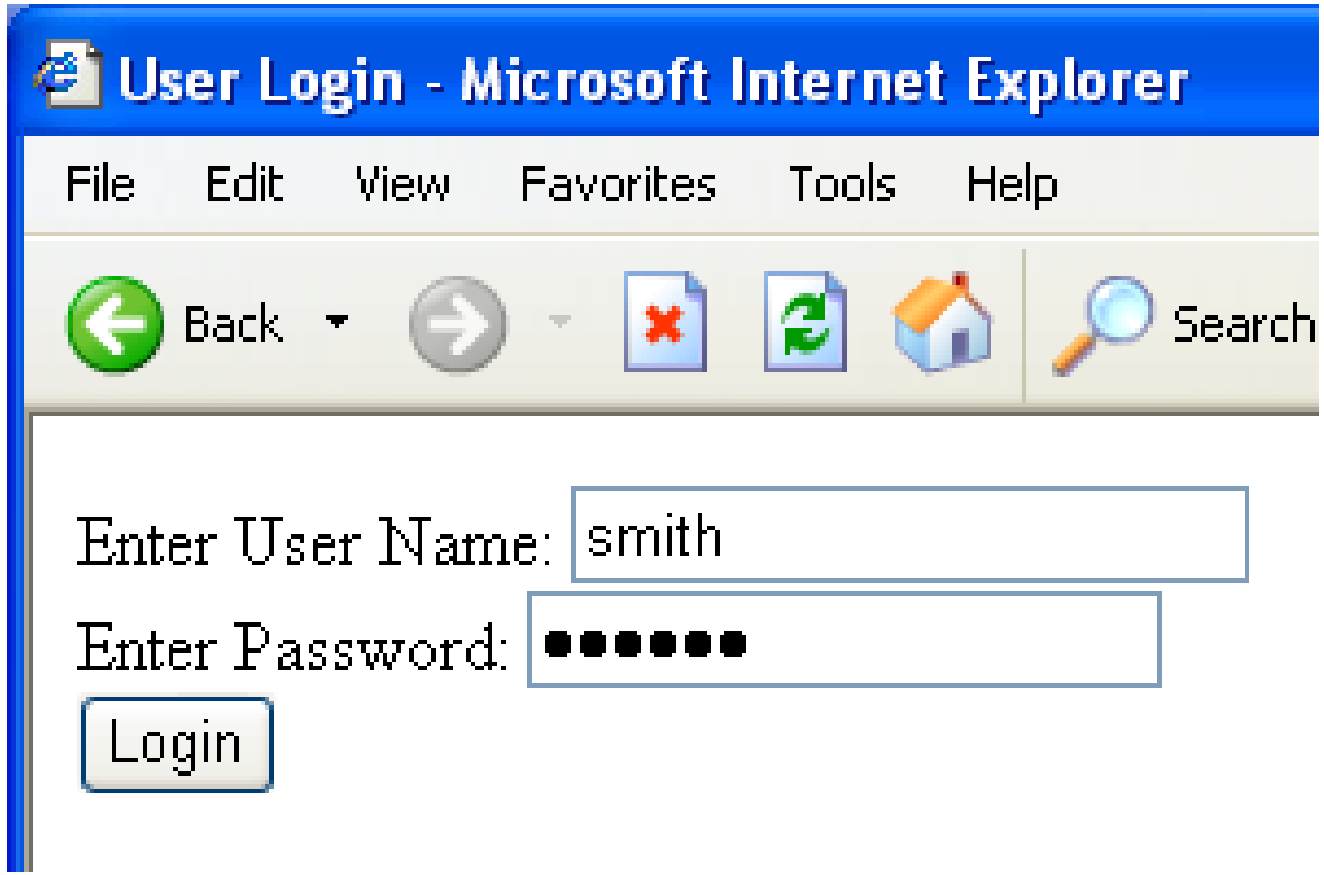
UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Announcements / Plan

- Monday (2/20): No class
- Wednesday (2/22): Update: Watch 2 Enigma talks
  - 2022 and earlier online: https://www.usenix.org/conference/enigma2022/program
  - Each ~20 minutes long
  - Reason: find two talks that help you with **your project** or are of interest to you for any other reasons (and Wednesday originally had Zoom guest lecture than was going)
  - Fill out "in class quiz" while/after watching
- Friday (2/24): Guest Lecture: Alex Gantman (Qualcomm) (On Zoom)
- Project submissions today; Yoshi will review over weekend / before Wednesday

Review: Another Common Web App Vulnerability: SQL Injection

Review: Typical Login Prompt

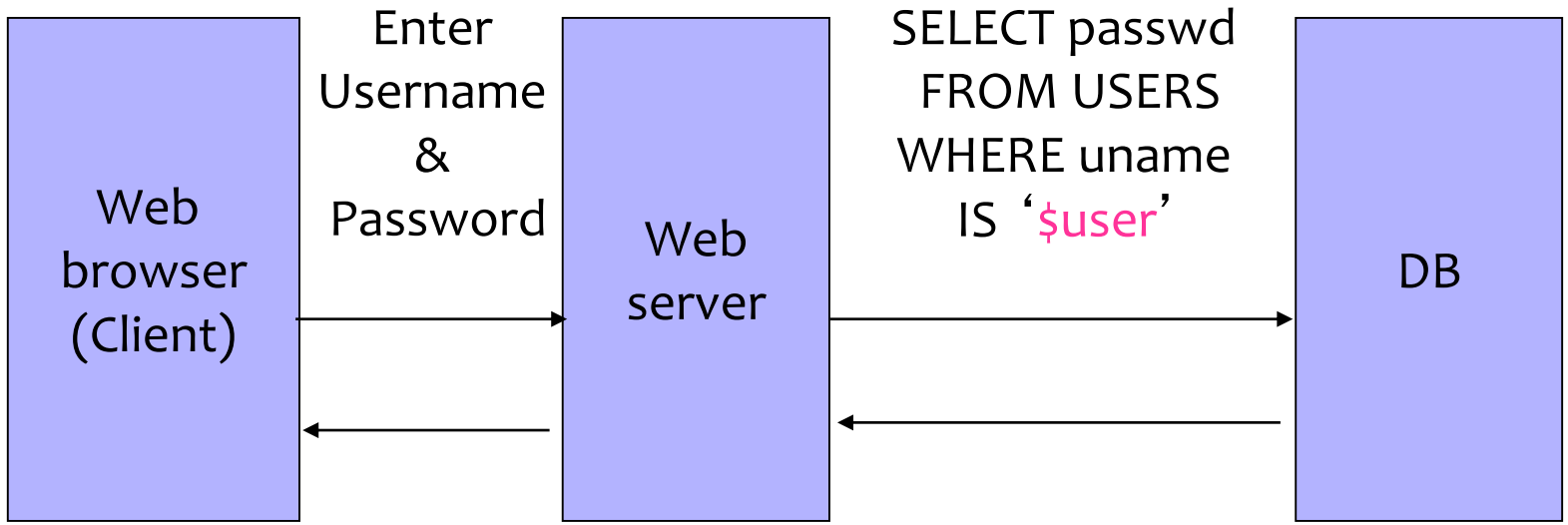


Review: Typical Query Generation Code

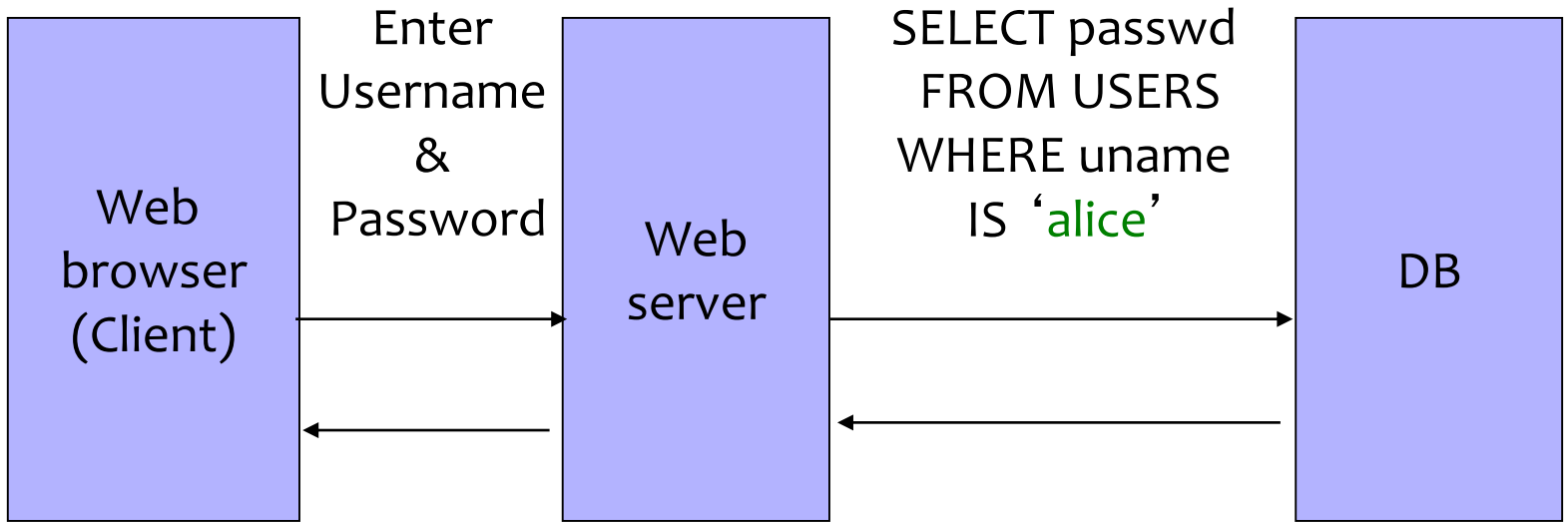
```
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key " .
       "WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

What if **'user'** is a malicious string that changes the meaning of the query?

Review: User Input Becomes Part of Query

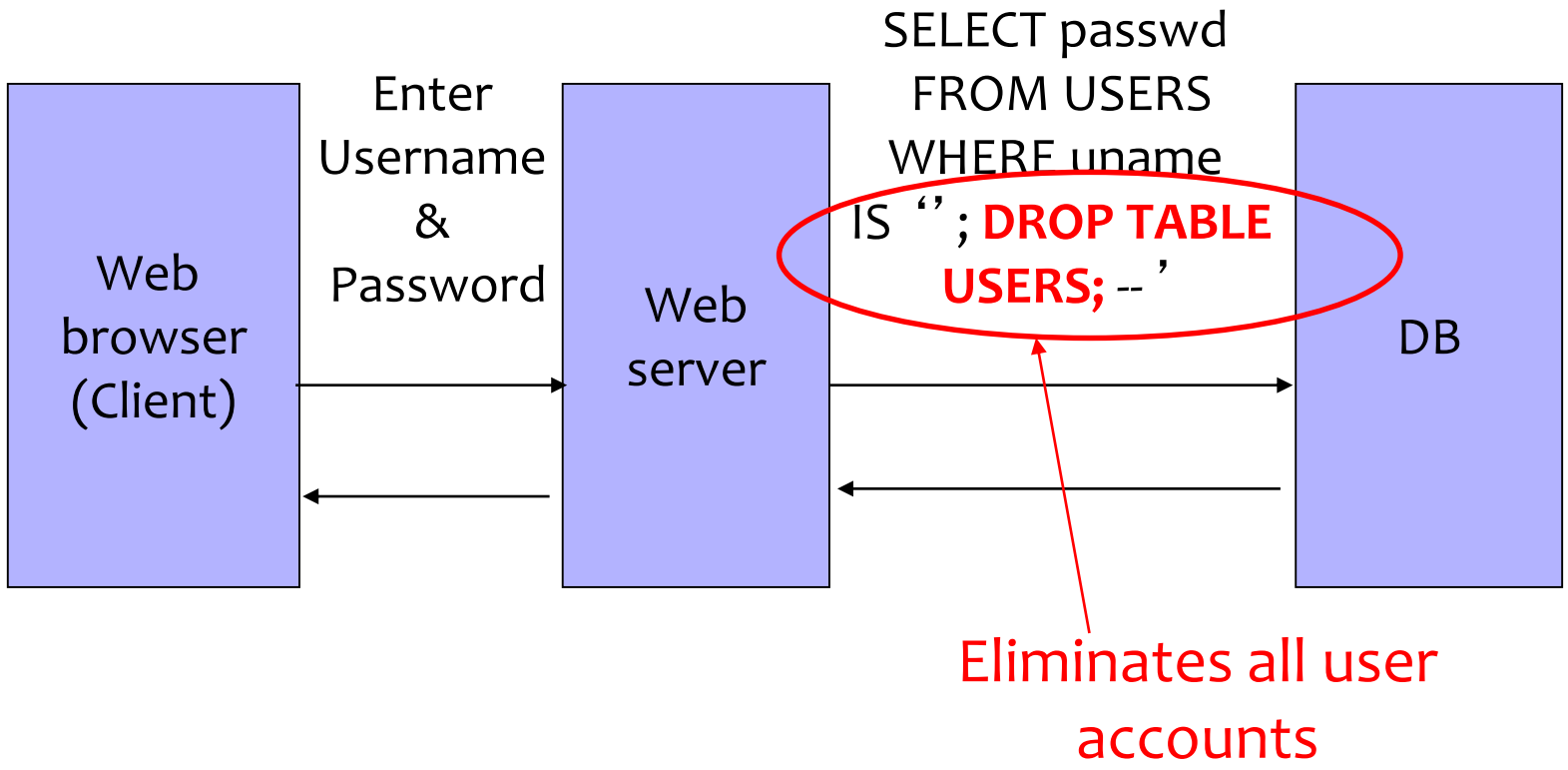


Review: Normal Login



Review: Malicious User Input

Review: SQL Injection Attack



Preventing SQL Injection

- Validate all inputs
  - Filter out any character that has special meaning
    - Apostrophes, semicolons, percent, hyphens, underscores, …
    - Use escape characters to prevent special characters form becoming part of the query code
      - E.g.: escape(O'Connor) = O\'Connor
  - Check the data type (e.g., input must be an integer)
- Same issue as with XSS: is there anything accidentally not checked / escaped?

Prepared Statements

```
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, quantity, order_day "
                + "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParamenter("month")));
ResultSet res = ps.executeQuery();
```

- Bind variables: placeholders guaranteed to be data (not code)
- Query is parsed without data parameters
- Bind variables are typed (int, string, …)

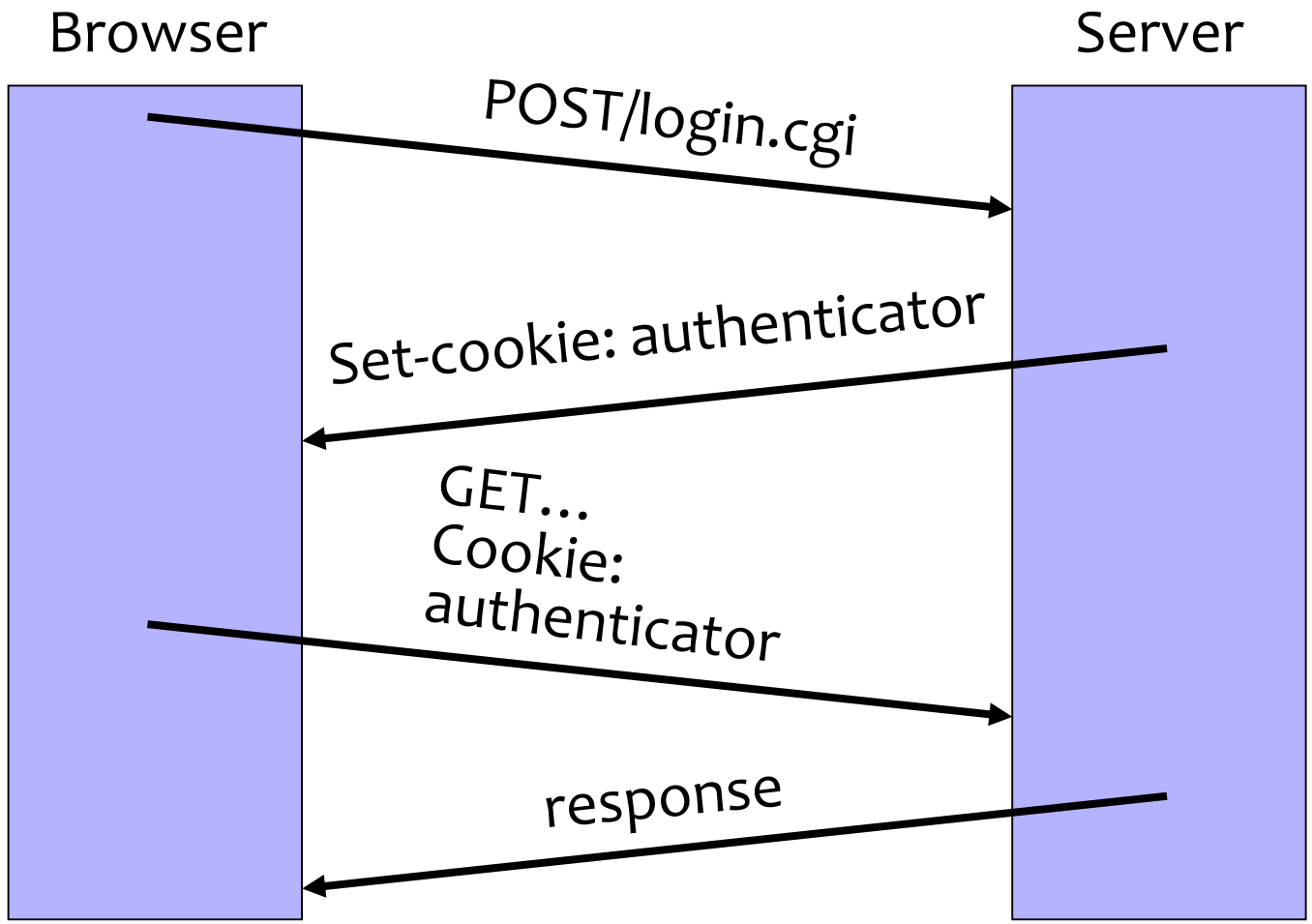
http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html

Core Issue: Data-As-Code

- XSS
- SQL Injection
- (Like buffer overflows)

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

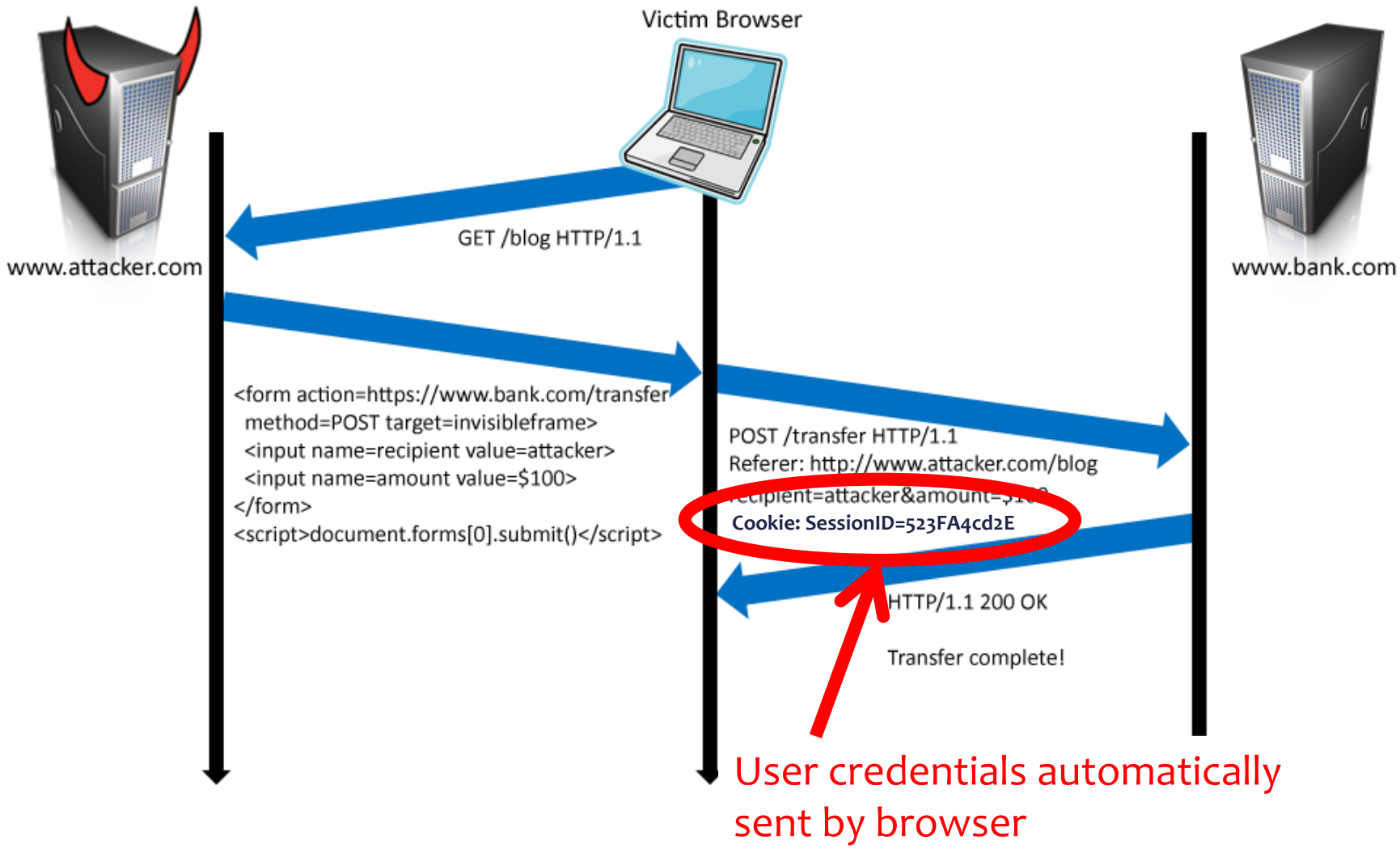
- Users logs into bank.com, forgets to sign off
  - Session cookie remains in browser state
- User then visits a malicious website containing

```
<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



Sending a Cross-Domain POST

```
<form method="POST" action=http://othersite.com/action >
...
</form>
<script>document.forms[0].submit()</script>
```

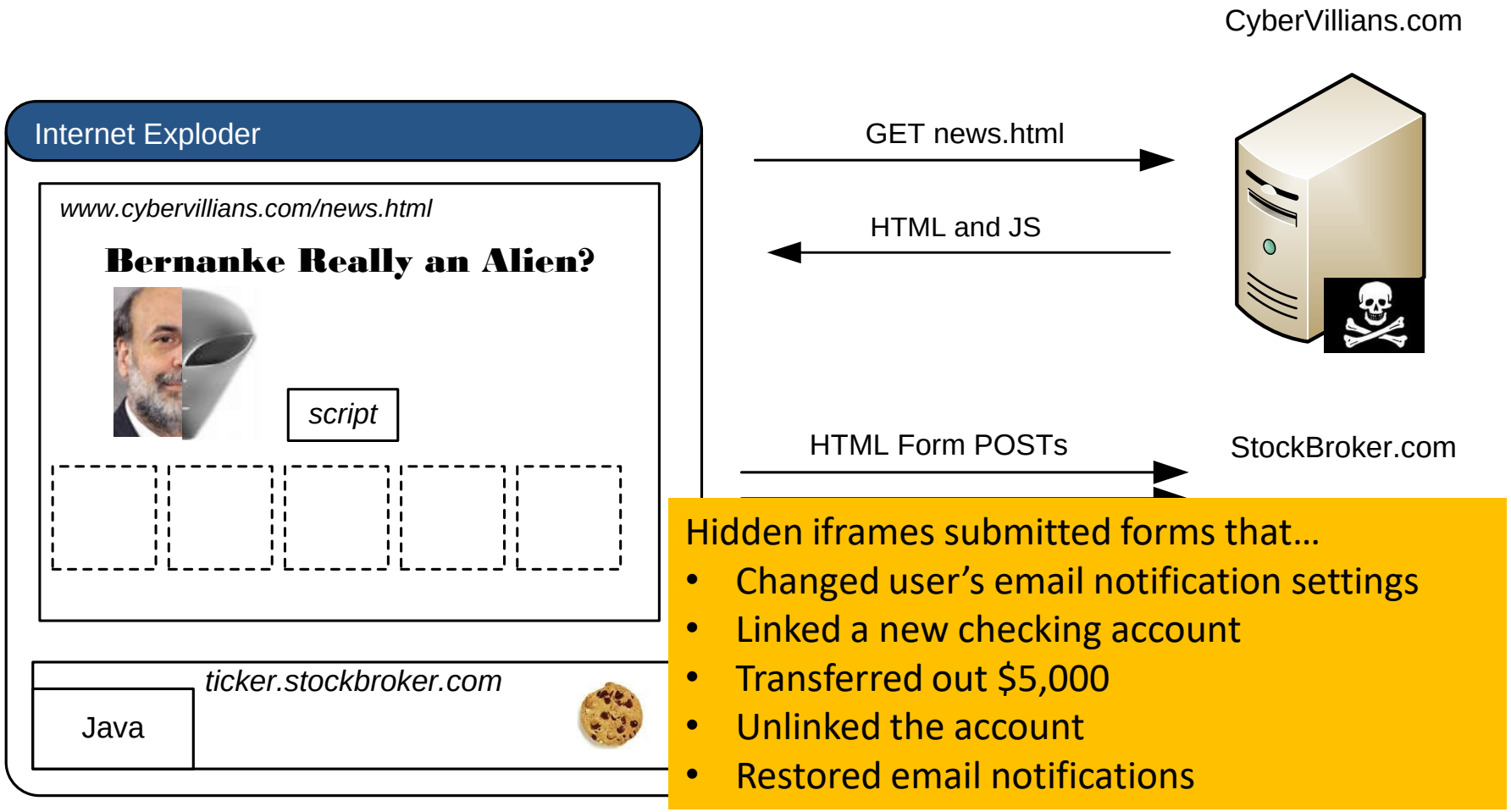
← submit post

- Hidden iframe can do this in the background
- User visits a malicious page, browser submits form on behalf of user
  - Hijack any ongoing session (**if no protection**)
    - Netflix: change account settings, Gmail: steal contacts, Amazon: one-click purchase
  - Reprogram the user's home router
  - Many other attacks possible

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?

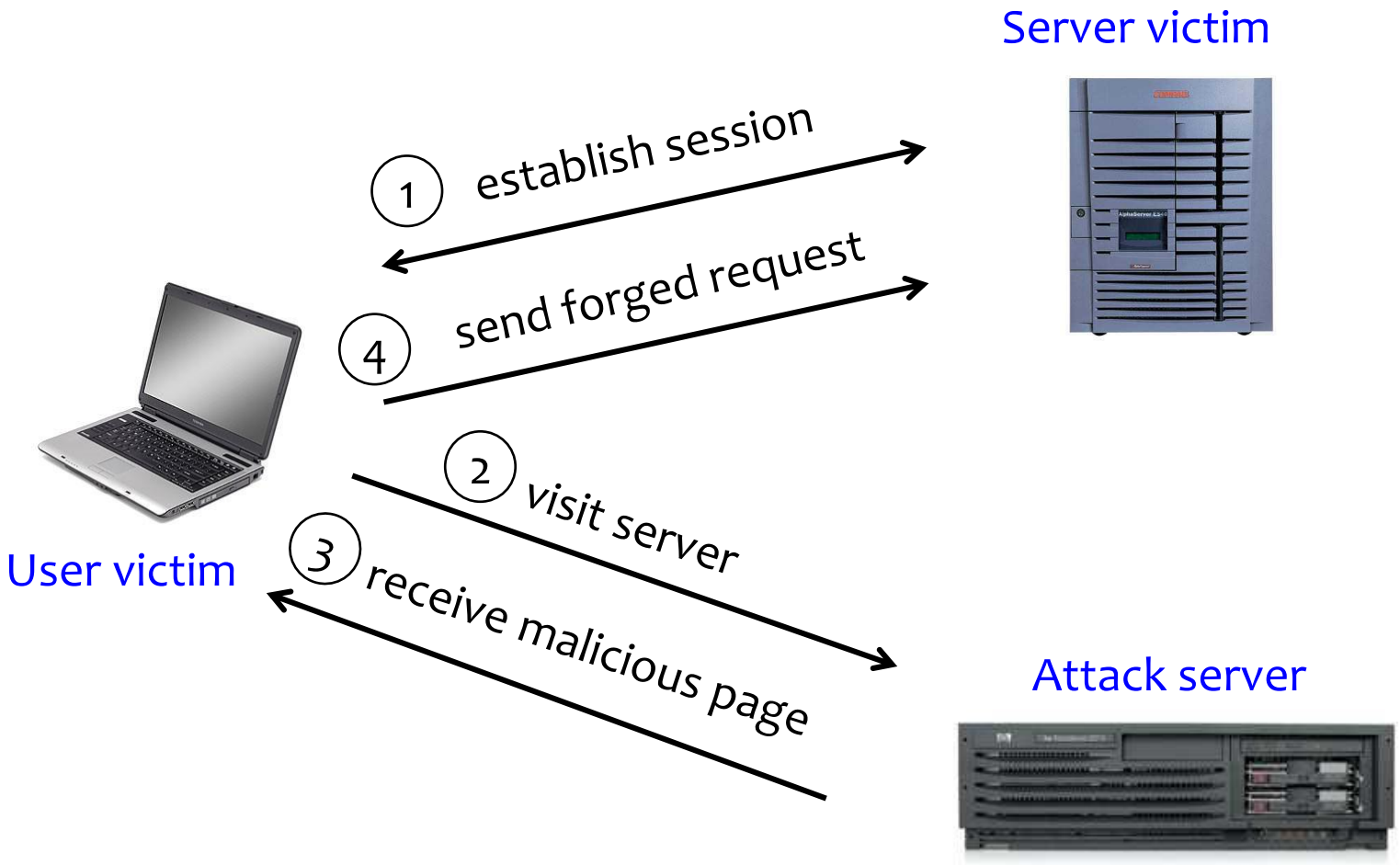
XSRF True Story [Alex Stamos]



Hidden iframes submitted forms that…

- Changed user's email notification settings
- Linked a new checking account
- Transferred out $5,000
- Unlinked the account
- Restored email notifications

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 initiates 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

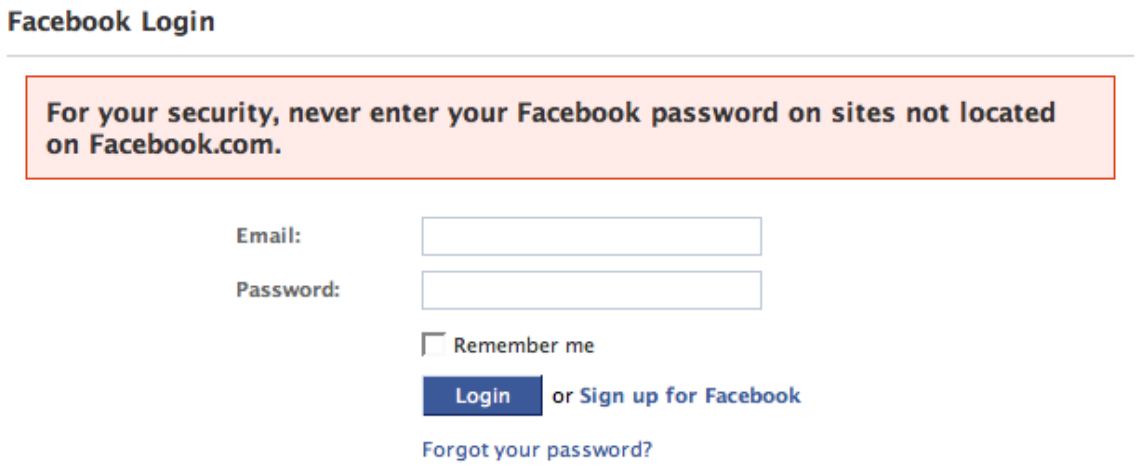
- Secret validation token


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

- Referer validation


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

Referer Validation



```
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

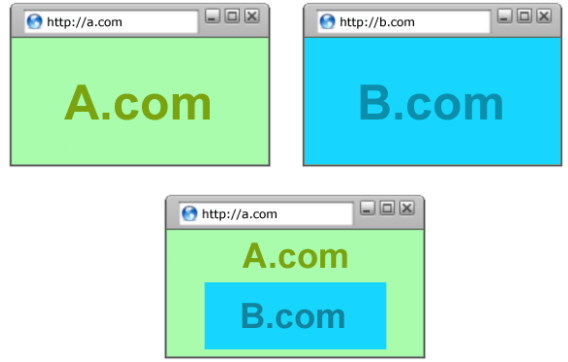
Review: Browser Security Model

Goal 1: Protect local system from web attacker

→ Browser Sandbox

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

→ Same Origin Policy



Browser Sandbox

Goals: (1) Protect local system from web attacker;
(2) 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

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

"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

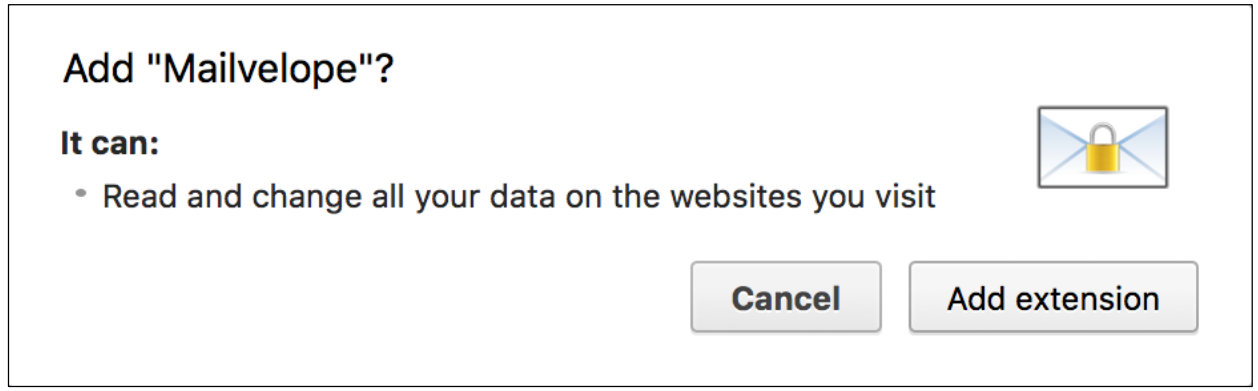
What about Browser Extensions?

- 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