CSE 484 / CSE M 584
Computer Security:
Web Security

TA: Franzi Roesner
franzi@cs.washington.edu
Logistics

• Homework #2 (crypto) due 2/22 5pm.
• Lab #2 (web security) due 2/27 5pm.

• Lab #1 looks AWESOME! 😊
# Same-Origin Policy

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 thanks to Wikipedia.]
Same-Origin Policy (DOM)

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

  - **www.example.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).
Same-Origin Policy (Cookies)

• **For cookies:** Only code from same origin can read/write cookies associated with an origin.
  – Can be set via Javascript (`document.cookie=...`) or via `Set-Cookie` header in HTTP response.
  – Can narrow to subdomain/path (e.g., `http://example.com` can set cookie scoped to `http://account.example.com/login`.)
  – **Secure cookie:** send only via HTTPS.
  – **HttpOnly cookie:** can’t access using JavaScript.
Same-Origin Policy (Cookies)

• Browsers **automatically include cookies** with HTTP requests.

• **First-party cookie**: belongs to top-level domain.

• **Third-party cookie**: belongs to domain of embedded content.
Same-Origin Policy (Scripts)

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

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


• If code in the script sets a cookie, under what origin will it be set?
XSS: Cross-Site Scripting

- **Idea:** Place *user-provided data* in the page.
  - Makes page more interactive and personal.

- **Threat:** Improperly used data can be interpreted as code.

- **Demo...**

- **Solutions?**
  - Sanitize/validate input. *(e.g., htmlspecialchars())*
  - Browser detection/prevention.
**XSSI: Cross-Site Script Inclusion**

- **Idea:** Include scripts (e.g., libraries) to run in context of current domain.

  **Example:**
  
  ```html
  <head> <script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script> </head>
  ```

- **Threat:** Attacker provides malicious library, can execute code in your domain’s context.

- **Solution:** Make sure included code comes from trusted site.
XSRF: Cross-Site Request Forgery

• **Idea:** Protect sensitive actions (e.g., Amazon purchase) by authenticating users w/ cookies.

• **Threat:** Attacker tricks user’s browser into visiting sensitive URL. For example:


• **Why does this work?**
  
  – Browsers automatically attach cookies to requests.
XSRF Defense

Include XSRF token (e.g., based on user session):

```html
<form action="purchase.php" method="post">
  <input type="hidden" name="csrf" value="<?php echo $key; ?>" />
  <input type="submit" value="One-Click Purchase">
</form>
```

Why does this work?

Attacker can’t read token due to same-origin policy.
Lab #2 Explained

abstract.cs.washington.edu/~you/evil.php

TAs

authToken=aet38f

You

authToken=34gae8

(1) Click this:
search?v=x&q=evil

(2) Okay!
search?v=x&q=evil

(3) Evil page renders:
search?v=x&q=evil

(4) Cookie steal

(5) Use cookie to change grade
Lab #2 Guidelines

• Email me your group members, group name, and desired password.
• Your script must run on abstract.cs.washington.edu.
• Some versions of some browsers provide XSS protections, so testing might fail. (Try Firefox.)
• Make sure exploits work locally before submitting links to y.um.my.
• See lab FAQ for links to add-ons to modify cookies.
• Extra credit is hard/unexpected, based on real bug from previous TAs (don’t waste your time).
Clickjacking

• **Trick users** into interacting with sensitive user interfaces in another domain.
  – Using invisible iframes:

```
Click here to win!!!
```

– Exploit predictable user timing:

```http://lcamtuf.coredump.cx/ffgeo2/```