Improving Web Privacy And Security with a Cost-Benefit Analysis of the Web API

Pete Snyder
Thesis Defense

Committee: Christopher Kanich
Venkat Venkatakrishnan
Jakob Eriksson
Stephen Checkoway
Damon McCoy
The Web API

Only frequently beneficial

Only frequently used

Only low-risk

Frequently Used

User Serving

Security and Privacy Risky
Thesis Questions

• Q1: Can we quantitatively distinguish between high and low benefit, and high and low cost, browser features?

• Q2: Can we use this information to improve privacy and security for web users?
Outline

• Background
• Measuring use
• Measuring cost vs. benefit
• Applying findings to “current web”
• Applying findings to “future web”
Outline

• Background
• Measuring use
• Measuring cost vs. benefit
• Applying findings to “current web”
• Applying findings to “future web”
What is the Web API?

• Browser implemented functionality

• Provided to websites as JavaScript methods, events, structures

• Sites authors use these browser capabilities to create interactive sites

• Cross browser (mostly)
What the Web API Is Not

- Internals (networking stack, TLS, etc.)
- Browser interface
- Extensions
- Plugins
- Static documents
- (generally) anything browser specific
What is In the Web API?

- Document manipulation
- AJAX / server requests
- Cookies
- Browser navigation
- Complex graphics animations
- WebGL
- Cryptographic operations
- Parallel operations
- Font operations
- Styling / presentation
- Ambient light sensing
- Peer-to-peer networking
- Audio synthesis
- “Beacons”
- Geolocation
- Gamepads
- Vibration
- High resolution timers
- DRM
- SVG animations
- Speech synthesis
- Battery status
- Virtual reality support
- Selection events
- Fetch API
- Shared memory
- ResourceStats API
- Gesture support
- Pause Frame API
- CSS Paint API
- WebUSB
- Device Memory
- Server Timing
- etc.
Who Defines Web Standards?

• De Jure Standardzation
  • W3C, WHATWG, Khronos Group
  • E.g. WebGL, Web USB, WebVR

• De Facto Standardization
  • Browser competition, retroactive de facto standardization
  • E.g. innerHTML, WebExtension, early DOM standards
Terms

• **Feature**
  A single, JS accessible, function, data structure or event

• **Standard**
  A set of “features”, defined in a standards document (or subsection of a standards document), designed to accomplish a similar set of goals

• **Web API**
  Set of every feature in every standard, or union of all “standards”
Outline

• Background
• Measuring use
• Measuring cost vs. benefit
• Applying findings to “current web”
• Applying findings to “future web”
Measuring Feature Use

Snyder, Peter, Lara Ansari, Cynthia Taylor, and Chris Kanich. "Browser feature usage on the modern web." IMC 2016
Available Functionality

Used Functionality

Used Functionality without Advertising + Tracking

User Beneficial Functionality
Standard Use vs Benefit

Available Functionality

Used Functionality

Used Functionality without Advertising + Tracking

User Beneficial Functionality
Measuring Feature Use

Methodology
Standard Use vs Benefit

Available Functionality

Used Functionality

Used Functionality without Advertising + Tracking

User Beneficial Functionality
Available Functionality: Data Set

- Representative Browser
  - Firefox 43.0.1
  - Open source
  - Standards focused
Available Functionality: Data Set

• Firefox WebIDL

• 1,392 features

• 74 standards and sub-standards
Standard Use vs Benefit

Available Functionality

Used Functionality

Used Functionality without Advertising + Tracking

User Beneficial Functionality
Problem Area Bounds

- Anonymous and {no, low} trust environments
- Only consider costs and benefits to users
- Site and developer interests left out
Used Functionality: Methodology

- Alexa 10k to represent the web
- Too much for a manual review
- Javascript makes static analysis difficult
- Automation with extension-based measurement
Measuring Code Injection

1. Each browser requests the selected page
2. Proxy injects hooks at beginning of `<head>`
3. Each browser records every feature use

vanilla, example.com, Crypto.getRandomValues(), 1
vanilla, example.com, Node.cloneNode(), 10
ghostery, example.com, Node.cloneNode(), 10
abp, example.com, Crypto.getRandomValues(), 1
abp, example.com, Node.cloneNode(), 4
Measuring Code Injection

example.org

60 sec
Measuring Code Injection
Measuring Code Injection

60 sec  
example.org

example.org/path-1  example.org/path-2  example.org/path-3

60 60 60 60 60 60 60 60 60 60 60 60 60 60 60
Thirty days has September, April, June and November. 
Three score and ten days hath December; All the rest have thirty-one. 
April, June and November.

February’s all done, all the rest have thirty-one. Now You know.
Automated Measurement

- 5 times per domain
- Every site in Alexa 10k
- 4 browser configurations

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domains measured</td>
<td>9,733</td>
</tr>
<tr>
<td>Total website interaction time</td>
<td>480 days</td>
</tr>
<tr>
<td>Web pages visited</td>
<td>2,240,484</td>
</tr>
<tr>
<td>Feature invocations recorded</td>
<td>21,511,926,733</td>
</tr>
</tbody>
</table>
Standard Use vs Benefit

Available Functionality

Used Functionality

Used Functionality without Advertising + Tracking

User Beneficial Functionality
Removing Tracking and Advertising

ABP

Ghost icon
Measuring Code Injection

ABP + ABP
Measuring Feature Use

Results
Standard Popularity
Standard vs Site Popularity

Portion of all website visits

Portion of all websites

HTML: History

Timing Control
Standard Popularity by Date
Standard Popularity vs Blocking

Figure 6: Popularity of standards versus their block rate, on a log scale.

Figure 7: Comparison of block rates of standards using advertising vs. tracking blocking extensions.

5.7 Site Complexity

We also evaluated sites based on their complexity. We define complexity as the number of standards used on a given website. As Figure 8 shows, most sites use many standards: between 14 and 32 of the 74 available in the browser. No site used more than 41 standards, and a second mode exists around the zero mark, showing that a small but measurable number of sites use little to no JavaScript at all.
Standard Use vs Benefit

Available Functionality

Used Functionality

Used Functionality - (Advertising + Tracking)

User Beneficial Functionality
Outline

• Background

• Measuring use

• Measuring cost vs. benefit

• Applying findings to “current web”

• Applying findings to “future web”
Standard Use vs Benefit

- **Available Functionality**
- **Used Functionality**
- **Used Functionality without Advertising + Tracking**
- **User Beneficial Functionality**
Feature Cost vs. Benefit
Methodology
Feature Cost vs. Benefit: Methodology

- Measuring Benefit
  - Number of sites that “need” a feature

- Measuring Cost
  - References in peer-reviewed literature
  - Recent related vulnerabilities
  - Additional code complexity
Measuring Feature Benefit (1/3)

- **Intuition**: Web API standards that are less frequently needed to accomplish user-serving tasks are less beneficial to users.

- **Metric**: What % of websites break when a standard is removed from the browser?
  - ↑ means more beneficial, ↓ means less beneficial

- Only considers benefit to browser users (not site owners)

- Only considering the anonymous / no-trust case
For each of the 74 standards in the browser:

- Randomly select 40 sites using the standard
- Have two students independently visit the site for 60 seconds
- Remove the standard from the browser, revisit site for 60 seconds
- Record if they were able to accomplish "the site's main purpose"
- 96.74% agreement between testers
Measuring Feature Benefit (3/3)

- Ranking System
  - 1: No visible difference
  - 2: Some difference, but didn't affect core functionality
  - 3: Core functionality affected
- 96.74% agreement between testers (1 & 2 vs 3)
Feature Removal Strategy

• Problem
  • Removing functions from the environment will break unrelated code
  • Lead to over count in site need

• Goal
  • Want to block page access to functionality
  • Have other code run as normal
const canvas = document.createElement("canvas");

const gl = canvas.getContext("webgl");
const format = gl.getShaderPrecisionFormat(
    gl.VERTEX_SHADER,
    gl.MEDIUM_FLOAT
);
console.log(format.precision); // Finger printing

document.getElementById("some-element");
WebGLRenderingContext.prototype.getShaderPrecisionFormat = null;
const canvas = document.createElement("canvas");

const gl = canvas.getContext("webgl");
const format = gl.getShaderPrecisionFormat( // Throws
  gl.VERTEX_SHADER,
  gl.MEDIUM_FLOAT
);
console.log(format.precision); // Finger printing

// Never Called
document.getElementById("some-element");
WebGLRenderingContext.prototype.getShaderPrecisionFormat = () => null;
const canvas = document.createElement("canvas");

const gl = canvas.getContext("webgl");
const format = gl.getShaderPrecisionFormat(
    gl.VERTEX_SHADER,
    gl.MEDIUM_FLOAT
);
console.log(format.precision); // Throws

// Never Called
document.getElementById("some-element");
WebGLRenderingContext.prototype.getShaderPrecisionFormat = new Proxy(...);
const canvas = document.createElement("canvas");

const gl = canvas.getContext("webgl");
const format = gl.getShaderPrecisionFormat( // Proxied "call" operation
  gl.VERTEX_SHADER,
  gl.MEDIUM_FLOAT
);
console.log(format.precision); // Proxied "get" operation

// Code execution continues as expected
document.getElementById("some-element");
const blockingProxy = new Proxy(function () {}, {
  get: function (ignore, property) {
    if (property === Symbol.toPrimitive) {
      return toPrimitiveFunc;
    }
    if (property === "valueOf") {
      return toPrimitiveFunc;
    }
    return blockingProxy;
  },
  set: function () {
    return blockingProxy;
  },
  apply: function () {
    return blockingProxy;
  },
  ownKeys: function () {
    return unconfigurablePropNames;
  },
  has: function (ignore, property) {
    return (unconfigurablePropNames.indexOf(property) > -1);
  }
});
```javascript
const canvas = document.createElement("canvas");
const gl = canvas.getContext("webgl");
const format = gl.getShaderPrecisionFormat(
  // Proxied "call" operation
  gl.VERTEX_SHADER,
  gl.MEDIUM_FLOAT
);
format.get("these").things[3].thatDo().not.exist;

// Code still continues as expected
document.getElementById("some-element");
```
Measuring Benefit Summary

- Only a subset of the standards in Web API is used
- Users only notice when a subset of those standards are removed
- If users don't noticed when they're not available -> not useful
Per-Standard Cost

- Published attacks using the standard
- Past vulnerabilities associated with the standard
- Code complexity added by the standard
Standard Cost: Related Research

• **Intuition:** Functionality frequently leveraged in attacks in academic publications poses a greater cost to S&P.

• **Metric:** How many papers in top research conferences use a standard in their attack?

• Past 5 years of 10 top security conferences and journals (2010-2015)

• USENIX, S&P, NDSS, CCS, ESORICS, WOOT, ACSAC, Cryptology, etc
Standard Cost: Past Vulnerabilities

- **Intuition**: Functionality that has harmed security and privacy in the past should be treated with greater caution.

- **Metric**: How many CVEs have been filed against a standard's implementation in Firefox

- Look for all CVEs against Firefox since 2010

- Where possible, attribute to a standard

- 1,554 CVEs in general, 175 attributable to a standard

- Distinguish CVEs associated with a standard and other parts of the browser
# Standard Cost: Past Vulnerabilities

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Quantity</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE named Web API Standard</td>
<td>117</td>
<td>66.9%</td>
</tr>
<tr>
<td>CVE named JS method unique to a Web API Standard</td>
<td>32</td>
<td>18.3%</td>
</tr>
<tr>
<td>CVE named C++ method uniquely tied to a Web API Standard</td>
<td>21</td>
<td>12%</td>
</tr>
<tr>
<td>CVE named functionality uniquely associated with Web API Standard</td>
<td>5</td>
<td>2.8%</td>
</tr>
</tbody>
</table>

**Total** 175 100%
Standard Cost: Code Complexity

• **Intuition:** Functionality that adds greater complexity to the browser code base poses a greater cost to S&P.

• **Metric:** How many lines of code are uniquely in the browser to support each browser standard?

• Static analysis of C++ implementation code in Firefox
Standard Cost: Code Complexity

1. Build call-graph using Clang and Mozilla's DXR tools

2. Identify entry point into call graph for each JS end point in the standard

3. Remove those entry points and identify newly orphaned nodes

4. Attribute LOC in orphaned nodes as being code uniquely attributable to the standard

5. Remove newly orphaned nodes, GOTO 4
Interface `BatteryManager`:

```javascript
interface BatteryManager {
    readonly charging;
    readonly chargingTime;
    readonly dischargingTime;
}
```

Automatically generated binding functions:

```javascript
mozilla::dom::BatteryManagerBinding::charging
mozilla::dom::BatteryManagerBinding::chargingTime
mozilla::dom::BatteryManagerBinding::dischargingTime
```

Functions used exclusively for implementing the Battery API:

```javascript
mozilla::dom::BatteryManager::Charging
mozilla::dom::BatteryManager::ChargingTime
mozilla::dom::BatteryManager::DischargingTime
```
Standard Cost: Code Complexity

- Caveats and short comings
  - Does not include third party code
  - Does not include code shared between standards
- Metric: # lines of code unique to standard in Web API
Methodology Summary

- Alexa 10k as representative of the internet
- Firefox 43.0.1 as representative of browsers
- One metric for measuring benefit
  - Site break rate
- Three metrics for measuring cost
  - CVEs, academic literature, lines of code
Feature Cost vs. Benefit

Results
Standard Benefit

- Most standards provide very little benefit to browser users
- For 60% of standards, no measurable impact on browsing when they're removed
- Sometimes because the standard was never used (e.g. WebVTT)
- Sometimes because the standard is intended to not be visible (e.g. Beacon)
Standard Cost: Related Research (1/2)

- 20 papers using 23 standards, 51 standards were never implicated

- Examples

  - **Breaking sandbox isolations with the High Resolution Timers API**

  - **Fingerprinting and privacy attacks using Canvas API**

  - **Recovering length of cross origin responses using Fetch API**
### Standard Cost: Related Research (2/2)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Count</th>
<th>conferences/venues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Workers</td>
<td>3</td>
<td>CCS 2015 (2), USENIX 2016</td>
</tr>
<tr>
<td>Fetch</td>
<td>3</td>
<td>CCS 2015 (2), USENIX 2016</td>
</tr>
<tr>
<td>Web Storage</td>
<td>3</td>
<td>ACSAC 2016, WOOT 2014, CCS 2015</td>
</tr>
</tbody>
</table>
Standard Cost: CVEs

- CVEs are distributed unevenly
- A small number of Web API standards account for most CVEs since 2010
- Many frequently implicated standards are rarely used / needed
- Suggests areas for S&P benefit
Standard Cost: Implementation Complexity

- 75,650 lines uniquely attributable
- Widely different costs between standards
- Undercounts because of:
  - third party libraries
  - shared code
<table>
<thead>
<tr>
<th>Standard Name</th>
<th>Abbreviation</th>
<th># Alexa 10k Using</th>
<th>Site Break Rate</th>
<th>Agree</th>
<th># CVEs</th>
<th># High or Severe</th>
<th>% ELoC</th>
<th>Enabled attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebGL</td>
<td>WEBGL</td>
<td>852</td>
<td>&lt;1%</td>
<td>93%</td>
<td>31</td>
<td>22</td>
<td>27.43</td>
<td>[15, 21, 34, 40]</td>
</tr>
<tr>
<td>HTML: Web Workers</td>
<td>H-WW</td>
<td>856</td>
<td>0%</td>
<td>100%</td>
<td>16</td>
<td>9</td>
<td>1.63</td>
<td>[30, 34]</td>
</tr>
<tr>
<td>WebRTC</td>
<td>WRTC</td>
<td>24</td>
<td>0%</td>
<td>93%</td>
<td>15</td>
<td>4</td>
<td>2.48</td>
<td>[15, 26]</td>
</tr>
<tr>
<td>HTML: The canvas element</td>
<td>H-C</td>
<td>6935</td>
<td>0%</td>
<td>100%</td>
<td>14</td>
<td>6</td>
<td>5.03</td>
<td>[12, 15, 21, 26, 34, 38, 40]</td>
</tr>
<tr>
<td>Scalable Vector Graphics</td>
<td>SVG</td>
<td>1516</td>
<td>0%</td>
<td>98%</td>
<td>13</td>
<td>10</td>
<td>7.86</td>
<td></td>
</tr>
<tr>
<td>Web Audio API</td>
<td>WEBA</td>
<td>148</td>
<td>0%</td>
<td>100%</td>
<td>10</td>
<td>5</td>
<td>5.79</td>
<td>[15, 26]</td>
</tr>
<tr>
<td>XMLHttpRequest</td>
<td>AJAX</td>
<td>7806</td>
<td>32%</td>
<td>82%</td>
<td>11</td>
<td>4</td>
<td>1.73</td>
<td></td>
</tr>
<tr>
<td>HTML</td>
<td>HTML</td>
<td>8939</td>
<td>40%</td>
<td>85%</td>
<td>6</td>
<td>2</td>
<td>0.89</td>
<td>[13, 46]</td>
</tr>
<tr>
<td>HTML 5</td>
<td>HTML5</td>
<td>6882</td>
<td>4%</td>
<td>97%</td>
<td>5</td>
<td>2</td>
<td>5.72</td>
<td></td>
</tr>
<tr>
<td>Service Workers</td>
<td>SW</td>
<td>0</td>
<td>0%</td>
<td>-</td>
<td>5</td>
<td>0</td>
<td>2.84</td>
<td>[28, 59, 60]</td>
</tr>
<tr>
<td>HTML: Web Sockets</td>
<td>H-WS</td>
<td>514</td>
<td>0%</td>
<td>95%</td>
<td>5</td>
<td>3</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td>HTML: History Interface</td>
<td>H-HI</td>
<td>1481</td>
<td>1%</td>
<td>96%</td>
<td>5</td>
<td>1</td>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>Indexed Database API</td>
<td>IDB</td>
<td>288</td>
<td>&lt;1%</td>
<td>100%</td>
<td>4</td>
<td>2</td>
<td>4.73</td>
<td>[12, 15]</td>
</tr>
<tr>
<td>Web Cryptography API</td>
<td>WCR</td>
<td>7048</td>
<td>4%</td>
<td>90%</td>
<td>4</td>
<td>3</td>
<td>0.52</td>
<td></td>
</tr>
<tr>
<td>Media Capture and Streams</td>
<td>MCS</td>
<td>49</td>
<td>0%</td>
<td>95%</td>
<td>4</td>
<td>3</td>
<td>1.08</td>
<td>[57]</td>
</tr>
<tr>
<td>DOM Level 2: HTML</td>
<td>DOM2-H</td>
<td>8956</td>
<td>13%</td>
<td>89%</td>
<td>3</td>
<td>1</td>
<td>2.09</td>
<td></td>
</tr>
<tr>
<td>DOM Level 2: Traversal and Range</td>
<td>DOM2-T</td>
<td>4406</td>
<td>0%</td>
<td>100%</td>
<td>3</td>
<td>2</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>HTML 5.1</td>
<td>HTML51</td>
<td>2</td>
<td>0%</td>
<td>100%</td>
<td>3</td>
<td>1</td>
<td>1.18</td>
<td></td>
</tr>
<tr>
<td>Resource Timing</td>
<td>RT</td>
<td>433</td>
<td>0%</td>
<td>98%</td>
<td>3</td>
<td>0</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>Fullscreen API</td>
<td>FULL</td>
<td>229</td>
<td>0%</td>
<td>95%</td>
<td>3</td>
<td>1</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>Beacon</td>
<td>BE</td>
<td>2302</td>
<td>0%</td>
<td>100%</td>
<td>2</td>
<td>0</td>
<td>0.23</td>
<td></td>
</tr>
<tr>
<td>DOM Level 1</td>
<td>DOM1</td>
<td>9113</td>
<td>63%</td>
<td>96%</td>
<td>2</td>
<td>2</td>
<td>1.66</td>
<td></td>
</tr>
<tr>
<td>DOM Parsing and Serialization</td>
<td>DOM-PS</td>
<td>2814</td>
<td>0%</td>
<td>83%</td>
<td>2</td>
<td>1</td>
<td>0.31</td>
<td></td>
</tr>
<tr>
<td>DOM Level 2: Events</td>
<td>DOM2-E</td>
<td>9038</td>
<td>34%</td>
<td>96%</td>
<td>2</td>
<td>0</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>DOM Level 2: Style</td>
<td>DOM2-S</td>
<td>8773</td>
<td>31%</td>
<td>93%</td>
<td>2</td>
<td>1</td>
<td>0.69</td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>F</td>
<td>63</td>
<td>&lt;1%</td>
<td>90%</td>
<td>2</td>
<td>0</td>
<td>1.14</td>
<td>[28, 59, 60]</td>
</tr>
<tr>
<td>CSS Object Model</td>
<td>CSS-OM</td>
<td>8094</td>
<td>5%</td>
<td>94%</td>
<td>1</td>
<td>0</td>
<td>0.17</td>
<td>[46]</td>
</tr>
</tbody>
</table>
Standard Use vs Benefit

Available Functionality

Used Functionality

Used Functionality without Advertising + Tracking

User Beneficial Functionality
Outline

• Background
• Measuring use
• Measuring cost vs. benefit
• Applying findings to “current web”
• Applying findings to “future web”
Measuring Feature Cost vs. Benefit


...along with significant work conducted after publication.
1. Web API standards differ hugely in the benefit and cost they provide browser users.

2. All standards are equally available to web sites (with rare exceptions)

3. Users' privacy and security would be improved, at little cost, if non-trusted sites were only given access to useful, safe features (by default).
## Motivation from Results (2/2)

<table>
<thead>
<tr>
<th></th>
<th>Break Rate</th>
<th># CVEs</th>
<th># Attacks</th>
<th>% LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DOM2: Core</strong></td>
<td>89%</td>
<td>0</td>
<td>0</td>
<td>0.29%</td>
</tr>
<tr>
<td><strong>AJAX</strong></td>
<td>32%</td>
<td>11</td>
<td>0</td>
<td>1.73%</td>
</tr>
<tr>
<td><strong>Canvas</strong></td>
<td>0%</td>
<td>13</td>
<td>7</td>
<td>5.03%</td>
</tr>
<tr>
<td><strong>WebGL</strong></td>
<td>&lt;1%</td>
<td>31</td>
<td>4</td>
<td>27.43%</td>
</tr>
</tbody>
</table>
WebAPI Access Controls

- Browser extension that imposes access controls on Web API
- Users can restrict site access to functionality only when trusted / needed.
- Default configurations, user configurable
Usability Evaluation

- Interesting idea, but is it feasible (would anyone use it)
- Subjective measurements needed
- Impossible to evaluate $74^2$ possible configurations, on all websites
- Create plausible extension configurations
Evaluated Configurations

- Two tested, realistic, configurations

- **Conservative**: Block default access to 15 rarely needed standards

- **Aggressive**: Block 45 rarely needed and/or high-risk standards

<table>
<thead>
<tr>
<th>Standard</th>
<th>Conservative</th>
<th>Aggressive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beacon</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DOM Parsing</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Full Screen</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>High Resolution Timer</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Web Sockets</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Channel Messaging</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Web Workers</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Index Database API</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Performance Timeline</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>SVG 1.1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>UI Events</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Web Audio</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WebGL</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Ambient Light</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Battery Status</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>31 more…</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Evaluation Methodology

1. Select Representative sites
   - **Popular**: Non-pornographic, English sites in Alexa 200 (175 sites)
   - **Less Popular**: Random sampling of the rest of the Alexa 10k (155 sites)

2. Have two students visit each site for 60 seconds in default browser

3. Repeat visit in browser modified with `conservative` blocking configuration

4. Repeat visit in browser modified with `aggressive` blocking configuration

5. Compared break rates, both numerically and textually
Evaluation Findings

- Significant privacy and security benefits to blocking certain standards
- Tradeoff between S&P and functionality
- Testers agreed 97.6%-98.3% of the time

<table>
<thead>
<tr>
<th></th>
<th>Conservative</th>
<th>Aggressive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standards Blocked</td>
<td>15</td>
<td>45</td>
</tr>
<tr>
<td>Previous CVEs</td>
<td>89 (52.0%)</td>
<td>123 (71.9%)</td>
</tr>
<tr>
<td>Codepaths Avoided</td>
<td>123 (52.0%)</td>
<td>123 (71.9%)</td>
</tr>
<tr>
<td>LOC &quot;Removed&quot;</td>
<td>37,848 (50.00%)</td>
<td>37,848 (70.76%)</td>
</tr>
<tr>
<td>% Popular Sites Broken</td>
<td>7.14%</td>
<td>15.71%</td>
</tr>
<tr>
<td>% Less Popular Sites Broken</td>
<td>3.87%</td>
<td>11.61%</td>
</tr>
</tbody>
</table>
Usability Comparison

- How realistic are these tradeoffs?
- Repeat measurement using other popular browser privacy techniques
- Techniques compose, are not replacements

<table>
<thead>
<tr>
<th></th>
<th>% Popular Sites Broken</th>
<th>% Unpopular Sites Broken</th>
<th>Sites Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conservative Blocking</td>
<td>7.14%</td>
<td>3.87%</td>
<td>330</td>
</tr>
<tr>
<td>Aggressive Blocking</td>
<td>15.71%</td>
<td>11.61%</td>
<td>330</td>
</tr>
<tr>
<td>Tor Browser Bundle</td>
<td>16.28%</td>
<td>7.50%</td>
<td>100</td>
</tr>
<tr>
<td>No Script</td>
<td>40.86%</td>
<td>43.87%</td>
<td>300</td>
</tr>
</tbody>
</table>
Improving Usability

- Moved from fixed blocking configurations to dynamic
  - Trust context aware (HTTPS, logged in, privacy modes, etc.)
  - Crowd sourced / trusted rule lists (EasyList model)
  - Third party vs. first party code
  - Dwell time
- Single purpose applications
Lessons from Deployment

- > 1k users
- Actual, real world contributors!
- Publicity among privacy and security enthusiasts / activists
- Firefox and Chrome (and related...)
- https://github.com/snyderp/web-api-manager
Lessons Learned from Deployment

• Standards may be sub-optimal level of granularity
  • Often its just one feature (apple) that ruins the barrel

• Standards change fast
  • WebVR (2 versions!), Speech Synthesis, WebUSB, Payments API etc

• Common vulnerability in DOM modifying browser extensions
WebExtension Model
WebExtension Model

- Frame Created
- DOM Created
- Initial Script Executes
- New Script Fetched
- document_start
- document_end
- document_idle
WebExtension Model

- Frame Created
- DOM Created
- Initial Script Executes
- New Script Fetched
- document_start
- document_end
- document_idle
WebExtension Model

Parent Frame

Child Frame

document_start
WebExtension Vulnerability

Parent Frame

Child Frame

Frame Create
DOM Created
Initial Script
New Script

Frame Create
DOM Created
Initial Script
New Script
WebExtension Vulnerability

• Reported to Firefox, Chrome, Brave, EFF, uBlock Origin, etc
  • Fixed in Brave
  • Acknowledged by Firefox, EFF (Privacy Badger), and uBlock Origin
  • Still waiting in Chromium bug queue

• Possible fixes
  • Freeze parent frames while child frame is being set up
  • Move blocking into core browser functionality (TBB did, Brave now does)
Outline

• Background
• Measuring use
• Measuring cost vs. benefit
• Applying findings to “current web”
• Applying findings to “future web”

…along with significant work conducted after publication.
Findings to Build On

• Most sites don’t need most functionality

• Small amount of functionality gets users most of the benefits of the web

• JavaScript it difficult to predict benign from safe behavior

• Users and developers really like web application model
  • Decentralized, open, well understood application model
Goals for New Web Systems

- Improve privacy for non-technical users
  - Predictable, constrained information flow
- Improved security
  - Reduced attack surface, well tested code paths
- Predictable execution
  - Statically determinable execution effects
# CDF Approach

<table>
<thead>
<tr>
<th>Approach</th>
<th>Goal / Purpose</th>
</tr>
</thead>
</table>
| Declarative syntax | - Statically predictable behavior  
- Easy to write and check  
- Easy(er) to constant behavior |
| Trusted base interactive additions | - Write the tricky parts once, heavily-vet them  
- Allow sites to declare invocation parameters |
| Constrain information flow through syntax | - Disallow sending client-held information to 3rd party  
- Force server in the middle of third party communication |
| Proxy and compiler trusted based additions | - Compile statically-checked CDF into HTML+JS  
- Build on existing browser engineering |
CDF System

1. Client Request

2. CDF File

3. CDF → HTML+JS

4. HTML+JS

5. Trusted JS

6. “Safe” Assets
CDF Example

```javascript
{  
  "t": "button",
  "c": [{"text": "click me"}],
  "e": [
    {  
      "t": "click",
      "b": {  
        "t": "states",
        "s": {  
          "stateId": "text-change",
          "wrap": true,
          "states": [[["button", {  
            "t": "replace-sub",
            "c": {  
              "text": "click on"
            }}, [["button", {  
              "t": "replace-sub",
              "c": {  
              "text": "click off"
            }]}]]], [["button", {  
              "t": "replace-sub",
              "c": {  
              "text": "click on"
            }]}]]}, [["button", {  
              "t": "replace-sub",
              "c": {  
              "text": "click off"
            }]}]]}
  }
}

<button>click me</button>

<script>
let buttons = document.getElementsByTagName("button");
let stateIndex = 0;
let textStates = ["click on", "click off"];
buttons[0].addEventListener("click", function (event) {
  let newTextIndex = stateIndex++ % textStates.length;
  let newText = textStates[newTextIndex];
  event.target.innerHTML = newText;
});
</script>
```
# CDF Structure

<table>
<thead>
<tr>
<th>CDF Type</th>
<th>Purpose in System</th>
<th>Type Examples</th>
<th>Current Analogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>Define static document structure</td>
<td>List, List Element, Image, Video</td>
<td>HTML tags</td>
</tr>
<tr>
<td>Event</td>
<td>Define timer and user event to respond to</td>
<td>“timer trigger”, “mouse over”</td>
<td>DOM events</td>
</tr>
<tr>
<td>Behavior</td>
<td>Define what to do when an Event occurs</td>
<td>“state transition”, “remove subtree”, “change attribute”</td>
<td>Javascript event handlers</td>
</tr>
<tr>
<td>Delta</td>
<td>Define changes to the current document</td>
<td>“cdf sub-document”, “attribute”, “remove event”</td>
<td>AJAX response, WebSocket Response</td>
</tr>
</tbody>
</table>
System Evaluation

• Popular blog
  http://www.vogue.com/

• Online-banking
  https://www.bankofamerica.com/

• Social media
  https://twitter.com/

• Collaborative web application
  HotCRP
CDF Take Aways

- Existing system: https://github.com/bitslab/cdf
- Most of the “power” of the HTML+JS isn’t needed for most of the web
- Most of the risk of the WebAPI isn’t worth the corresponding benefit
Outline

• Background

• Measuring use

• Measuring cost vs. benefit

• Applying findings to “current web”

• Applying findings to “future web”
Conclusions

• Web is an enormously popular application system

• Web an enormously complicated system

• It's possible to evaluate the cost and benefit of discrete parts of a complicated system

• Doing so has tangible security and privacy benefits on existing systems

• Those improvements can be used to guide the design of future systems
Thank you!

Especially committee members and BITSLab comrades

(…but especially committee members)

(…but especially Chris :)