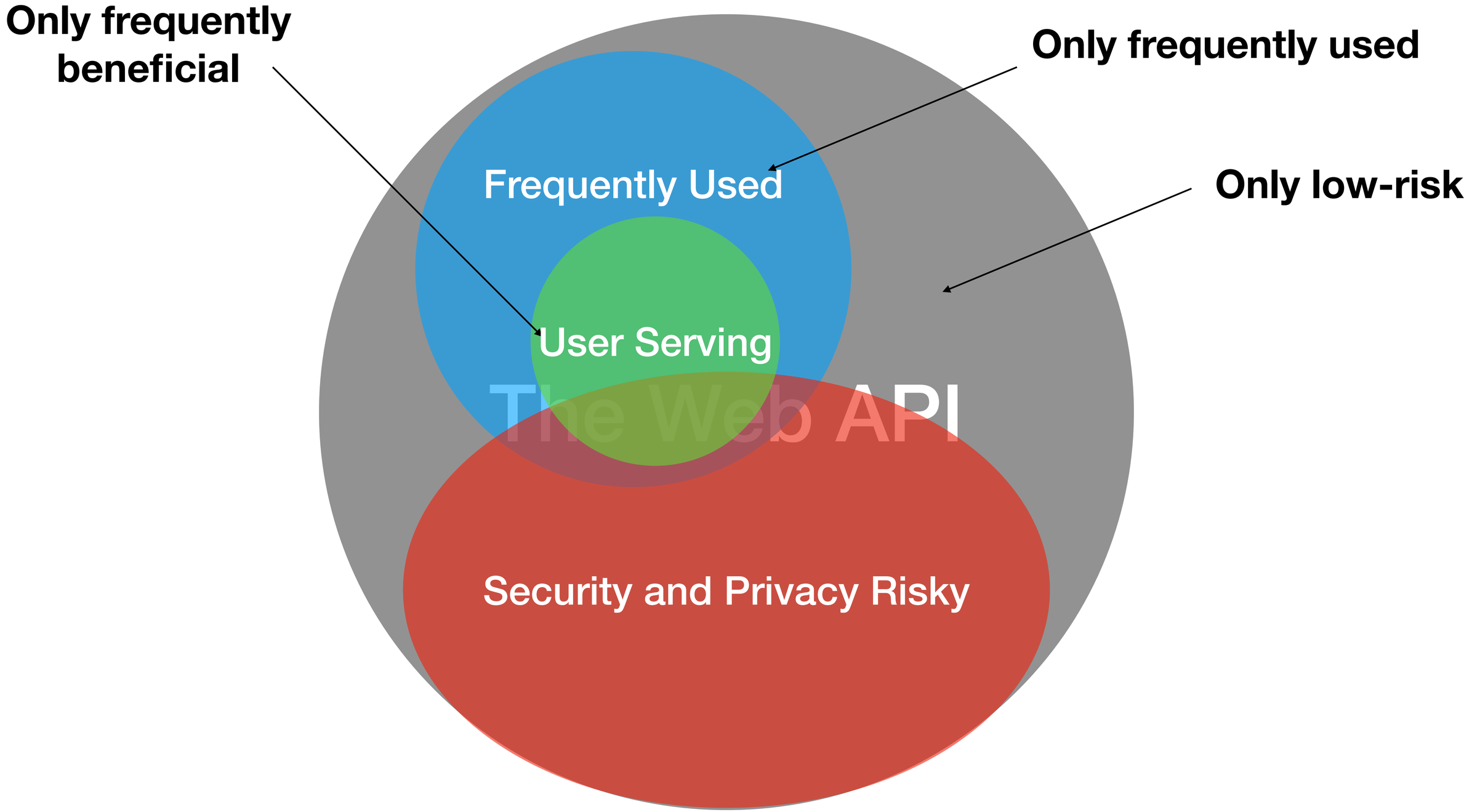
Most Websites Don’t Need to Vibrate: A Cost–Benefit Approach to Improving Browser Security

Peter Snyder – Cynthia Taylor – Chris Kanich


Only frequently beneficial
Only frequently used
Only low-risk
Frequently Used
User Serving
The Web API
Security and Privacy Risky

Outline

- Problem area
- Methodology and techniques
- Results and findings
- Proposed solution and evaluation

Outline

- Problem area
- Methodology and techniques
- Results and findings
- Proposed solution and evaluation

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.

Why the Web API Matters

- Privacy sensitive environment
- Permissive access control
- Frequent privacy and security violations

Research Questions

- Is providing so much capability to websites beneficial to users?
- Can we improve security and privacy imposing controls on what parts of the Web API pages can access?
- Problem area bounds:
  - Non-trust scenarios (e.g. non authenticated web)
  - Non-cutting edge Web API features

Outline

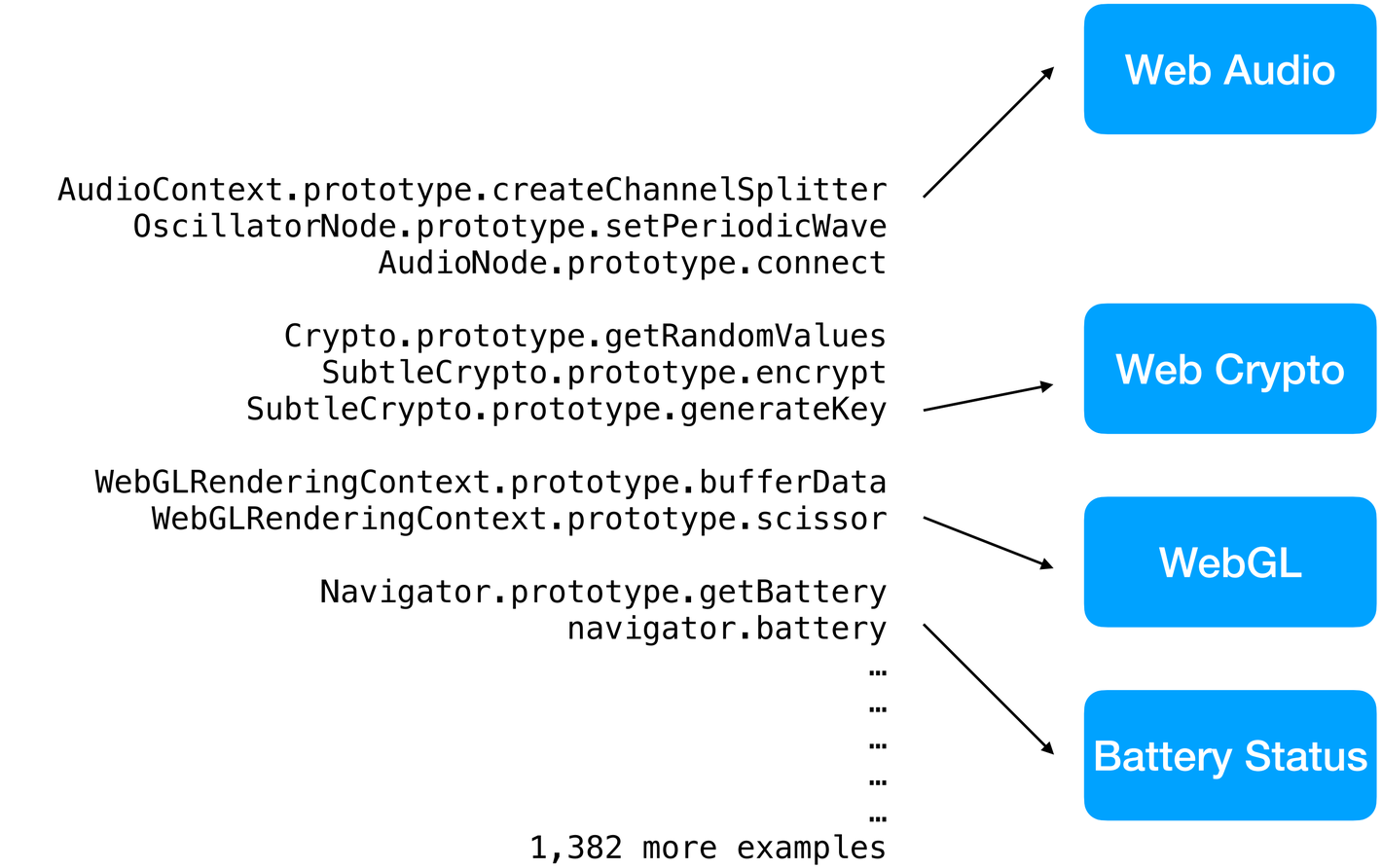
- Problem area
- Methodology and techniques
- Results and findings
- Proposed solution and evaluation

Methodology: Data Set

- Representative Browser
  - Firefox 43.0.1
  - Open source
  - Subject to relevant prior work
  - Standards focused

Determining Web API in Firefox

- JavaScript end points are defined through WebIDL
- 1,392 functions and properties defined in source
- Manually categorized into 74 standards and sub-standards
- Standards are the unit of measurement in this work



Per Standard Cost and Benefit

- 74 Standards in the browser
- Per standard benefit
  - Usefulness to users browsing the Web
- Per standard cost
  - Number of past vulnerabilities
  - Number of attacks in academic conferences
  - Complexity added to the code base

Determining Benefit: Strategy

- **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

Standard Benefit: Site Use

- Determine which sites in the Alexa 10k use each standard
  Snyder et al, Browser Feature Usage on the Modern Web, IMC 2016
  - Instrument a browser to record Web API use
  - Automate a browser to interact with websites automatically (repeated random interaction)
  - Every site for the Alexa 10k

Standard Benefit: Site Need

- Use ↛ need (advertising, tracking, analytics, etc.)
- For each standard
  - 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

Feature Removal Strategy

- Removing functions from the environment will break unrelated code paths
- Want to block page access to functionality, have other code run as normal
- Over count the affect of blocking a standard
- More fully described in the paper

```
var canvas = document.createElement("canvas");

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

document.getElementById("some-element);
```

```
WebGLRenderingContext.prototype.getShaderPrecisionFormat = null;
var canvas = document.createElement("canvas");

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

// Never Called
document.getElementById("some-element);
```

```
WebGLRenderingContext.prototype.getShaderPrecisionFormat = () => null;
var canvas = document.createElement("canvas");

var gl = canvas.getContext("webgl");
var 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(…);
var canvas = document.createElement("canvas");

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

// Code execution continues as expected
document.getElementById("some-element);
```

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: 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 proceedings at 10 top security conferences and journals:
- USENIX, S&P, NDSS, CCS, ESORICS, WOOT, ACSAC, Cryptology, etc

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

Methodology: Summary

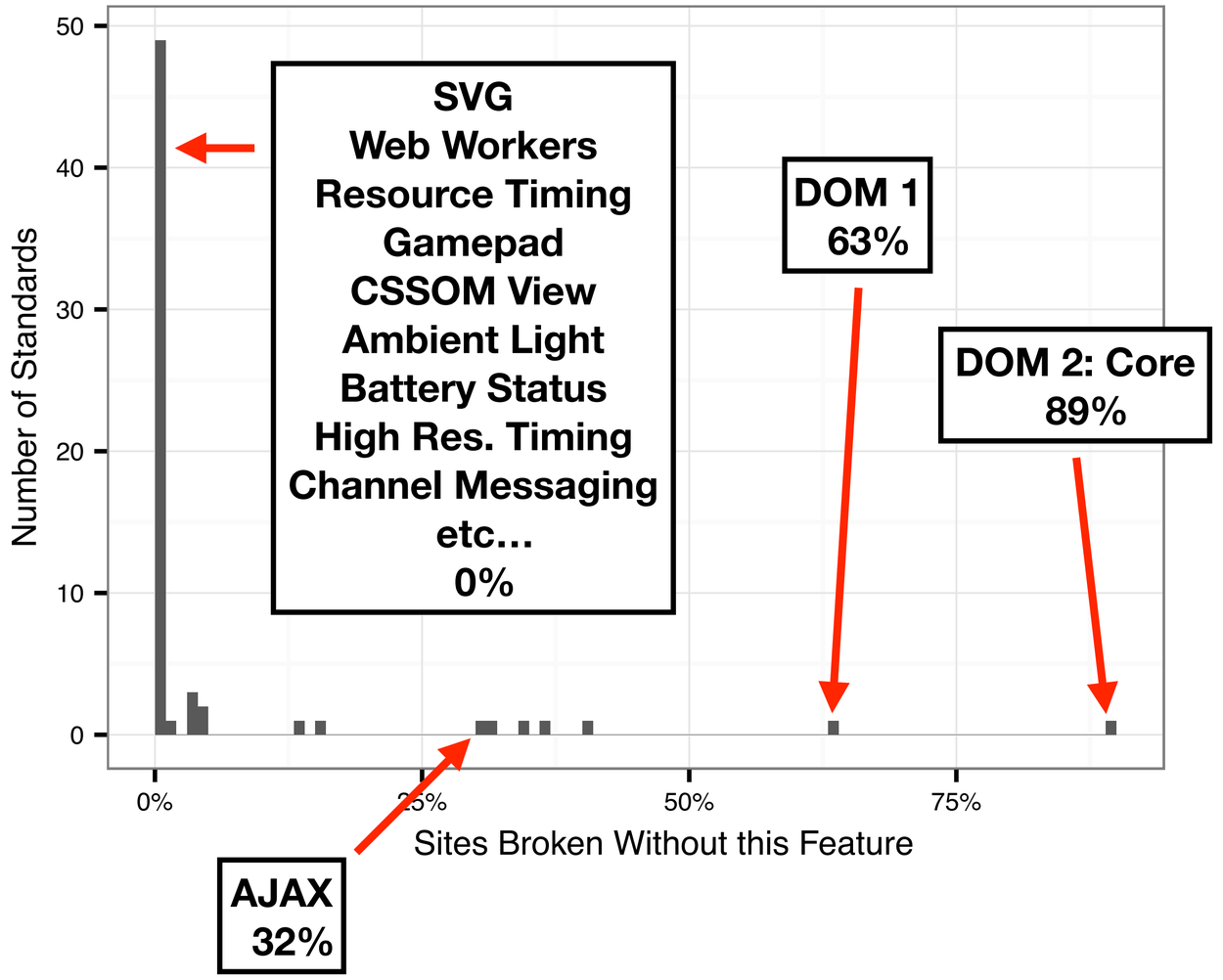
- Alexa 10k as representative of the internet
- Firefox 43.0.1 as representative of browsers
- One metric for measuring benefit
  - Site break rate
- Three metrics for measuring cost
  - CVEs, academic literature, lines of code

Outline

- Problem area
- Methodology and techniques
- Results and findings
- Proposed solution and evaluation

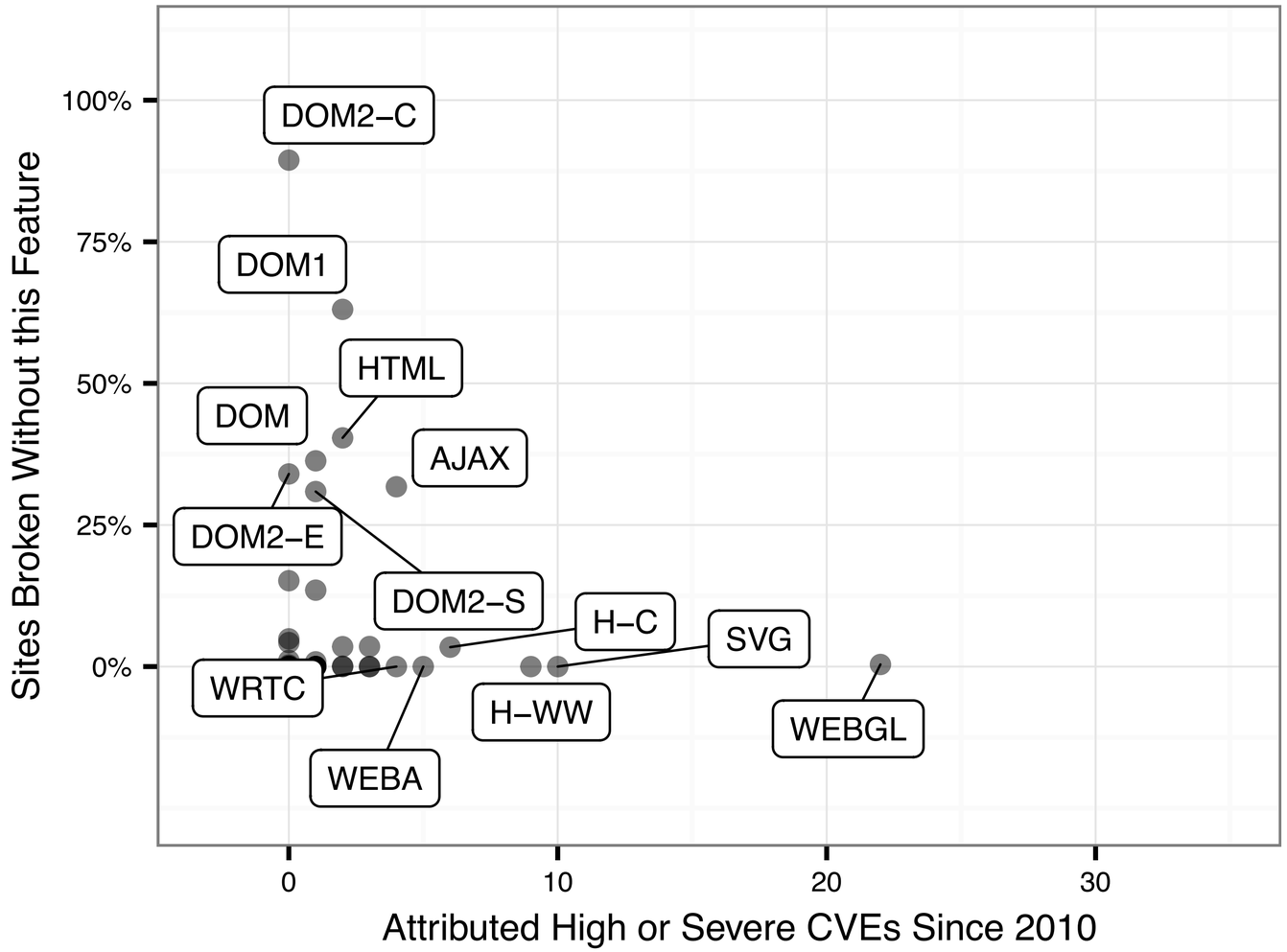
Standard Benefit

- 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: 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: Related Research (1/2)

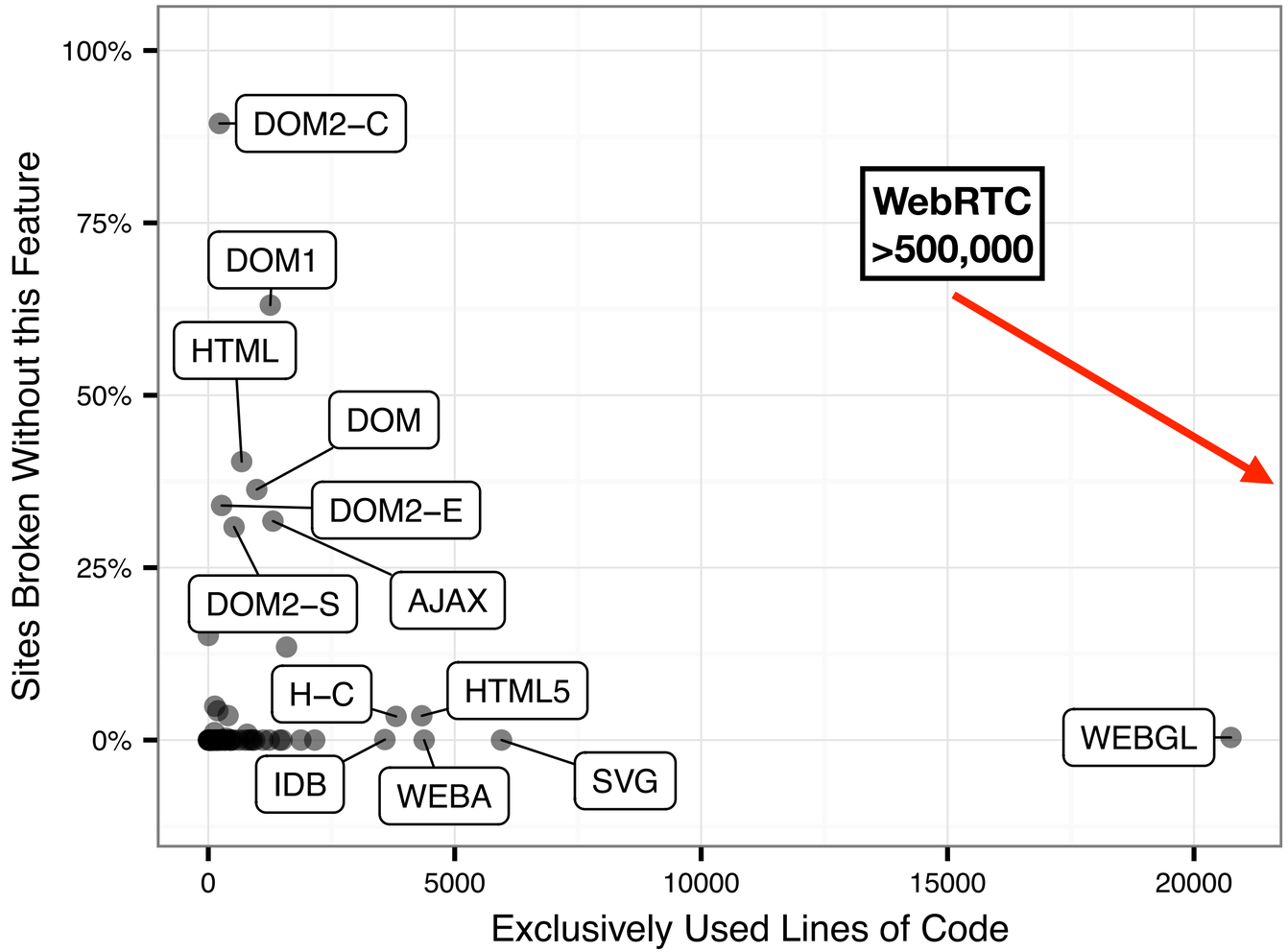
- 20 papers using 23 standards, 51 standards were never implicated
- Examples
  - **Breaking sandbox isolations with the High Resolution Timers API**
    EX: Andrysco, et al. "On subnormal floating point and abnormal timing." *S&P* 2015
  - **Fingerprinting and privacy attacks using Canvas API**
    Ex: Englehardt and Narayanan. "Online tracking: A 1-million-site measurement and analysis." *CCS* 2016
  - **Recovering length of cross origin responses using Fetch API**
    Ex: Van Goethem, et al. "Request and Conquer: Exposing Cross-Origin Resource Size." *USENIX* 2016.

Standard Cost: Related Research (2/2)

High Resolution Time Level 2	8	IEEE 2015, CCS 2015 (3), NDSS 2017, ESORICS 2015, WOOT 2014, CCS 2013
HTML: The Canvas Element	7	CCS 2014, ACSAC 2016, NDSS 2017, CCS 2016, WOOT 2014, CCS 2013, S&P 2016
Battery Status API	4	ACSAC 2016, CCS 2016, S&P 2013, Cryptology 2015
WebGL	4	ACSAC 2016, NDSS 2017, WOOT 2014, S&P 2016
Service Workers	3	CCS 2015 (2), USENIX 2016
Fetch	3	CCS 2015 (2), USENIX 2016
Web Storage	3	ACSAC 2016, WOOT 2014, CCS 2015

Standard Cost: Implementation Complexity

- 75,650 lines uniquely attributable
- Widely different costs between standards
- Undercounts because of:
  - third party libraries
  - shared code



Outline

- Problem area
- Methodology and techniques
- Results and findings
- Proposed solution and evaluation

Motivation from Results (1/2)

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)

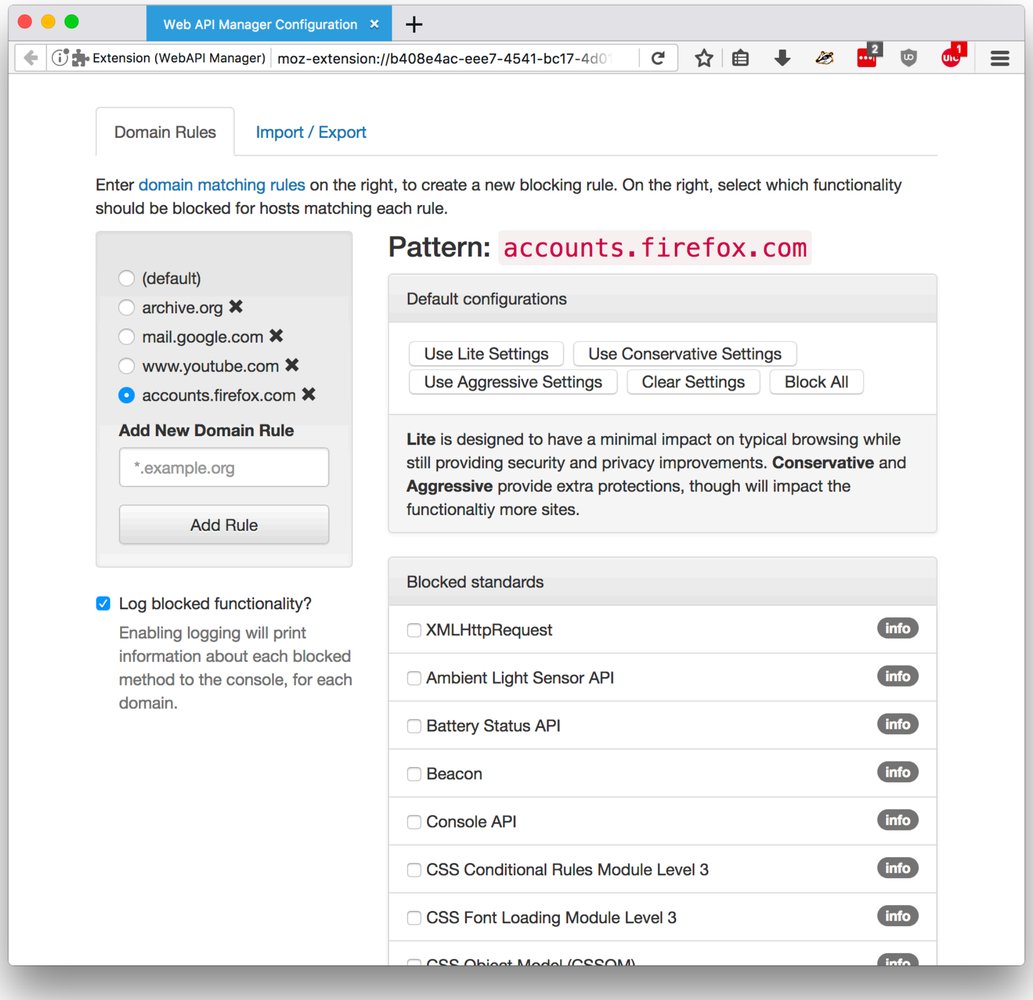
3. Users' privacy and security would be improved, at little cost, if non-trusted sites we're only given access to useful, safe features (by default).

Motivation from Results (2/2)

	Break Rate	# CVEs	# Attacks	% LOC
DOM2: Core	89%	0	0	0.29%
AJAX	32%	11	0	1.73%
Canvas	0%	13	7	5.03%
WebGL	<1%	31	4	27.43%

Proposed Solution

- Browser extension that imposes access controls on Web API
- Users can restrict site access to functionality only when trusted / needed.
- Default configurations, user configurable
- https://github.com/snyderp/web-api-manager



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

Standard	Conservative	Aggressive
Beacon	X	X
DOM Parsing	X	X
Full Screen	X	X
High Resolution Timer	X	X
Web Sockets	X	X
Channel Messaging	X	X
Web Workers	X	X
Index Database API	X	X
Performance Timeline	X	X
SVG 1.1	X	X
UI Events	X	X
Web Audio	X	X
WebGL	X	X
Ambient Light		X
Battery Status		X
31 more…		X

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

	Conservative	Aggressive
Standards Blocked	15	45
Previous CVEs Codepaths Avoided	89 (52.0%)	123 (71.9%)
LOC "Removed"	37,848 (50.00%)	37,848 (70.76%)
% Popular Sites Broken	7.14%	15.71%
% Less Popular Sites Broken	3.87%	11.61%

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

Discussion and Conclusions

Also In the Paper

- Specifics of our Web API blocking technique
- Numbers for standard use, break rates, CVE attributions, and academic attacks for all 74 standards
- Usability comparison with Tor Browser Bundle and NoScript
- Much more

Take Aways

- Large parts of the Web API are not needed for most websites.
- Many parts of rarely needed functionality carry high risks to user privacy and security.
- Data driven access controls can keep users safer with very small usability tradeoffs.
- We're working with browser vendors to integrate our findings.

Peter Snyder
psnyde2@uic.edu