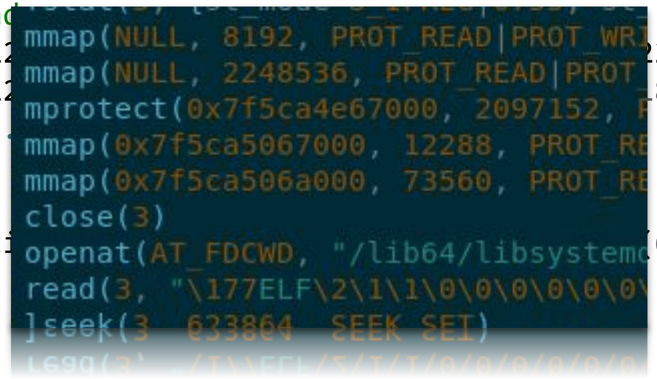
VisibleV8: In-browser Monitoring of JavaScript in the Wild

Jordan Jueckstock
North Carolina State University
jjuecks@ncsu.edu

Alexandros Kapravelos
North Carolina State University
akaprav@ncsu.edu

Seen in the wild...

```
/* ... following 100+ lines of obfuscated code defining u82222 ... */
(typeof window === "object" ? window : global).u82222 = u82222;
_ = wind
if (u822                                          2.e(0) +
    u822                                          8)]) ||
    /*
{}
else {
  locati                                          6) + u82222.e(17) + u82222.N(0), u82222.e(2))]();
}
```



```
mmap(NULL, 8192, PROT_READ|PROT_WRI
mmap(NULL, 2248536, PROT_READ|PROT
mprotect(0x7f5ca4e67000, 2097152, P
mmap(0x7f5ca5067000, 12288, PROT_RE
mmap(0x7f5ca506a000, 73560, PROT_RE
close(3)
openat(AT_FDCWD, "/lib64/libsystemd
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\
lseek(3, 633864, SEEK_SET)
```

How do we get `strace` for a browser?

- What does this code *do*?
- How do we *find out*?
- At *scale*?

Introducing VisibleV8

Standard Chromium + Instrumented V8 JS Engine

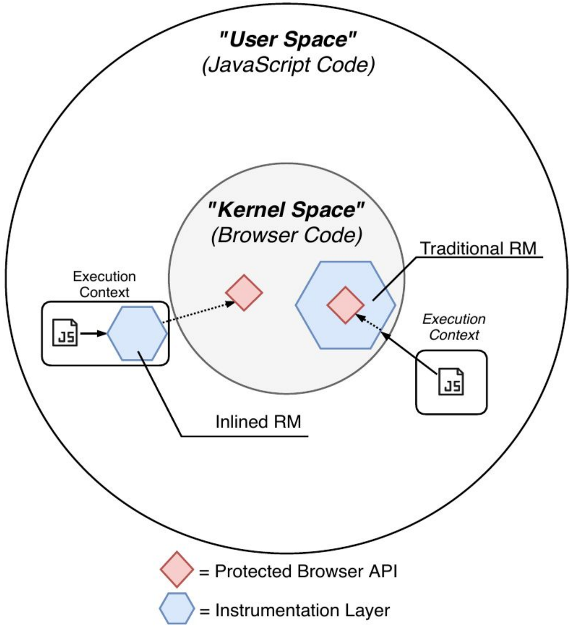
The Case Against In-Band JS Instrumentation

Background: Reference Monitors

Out-of-Band

vs.

In-Band

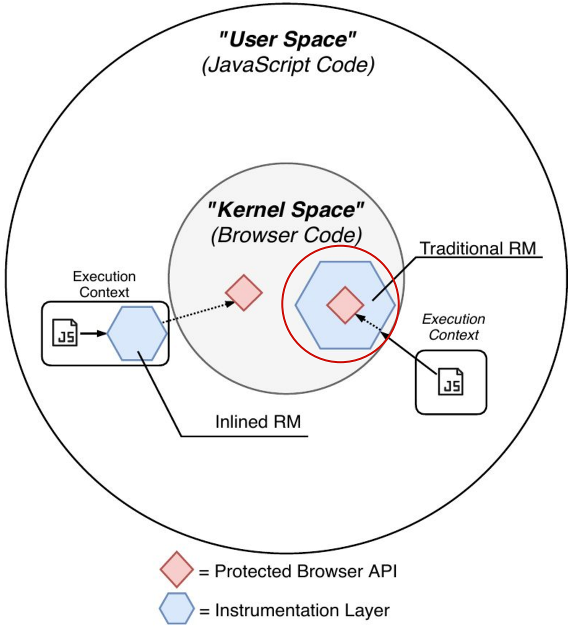


Background: Reference Monitors

Out-of-Band

vs.

In-Band

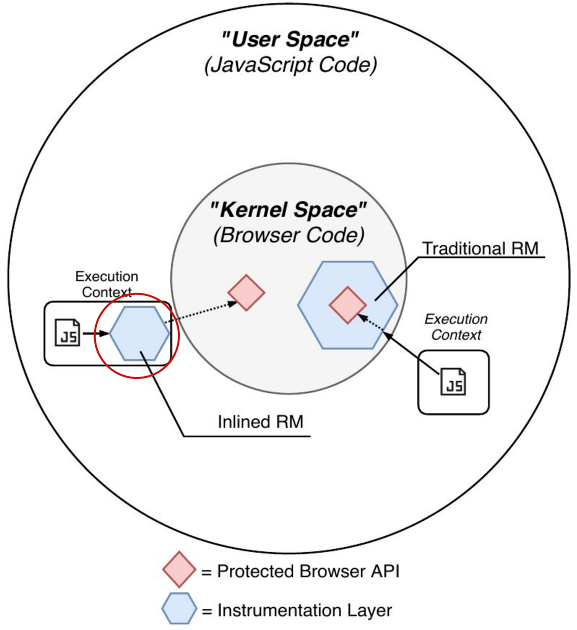


Background: Reference Monitors

Out-of-Band

vs.

In-Band



Related Research Tools

In-Band

Dynamic analysis
- OpenWPM [21,22,38]
- Snyder *et al.*, 2016 [49]
- FourthParty [35]
- TrackingObserver [45]

Policy Enforcement
- JavaScript Zero [47]
- Snyder *et al.*, 2017 [50]

Out-of-Band

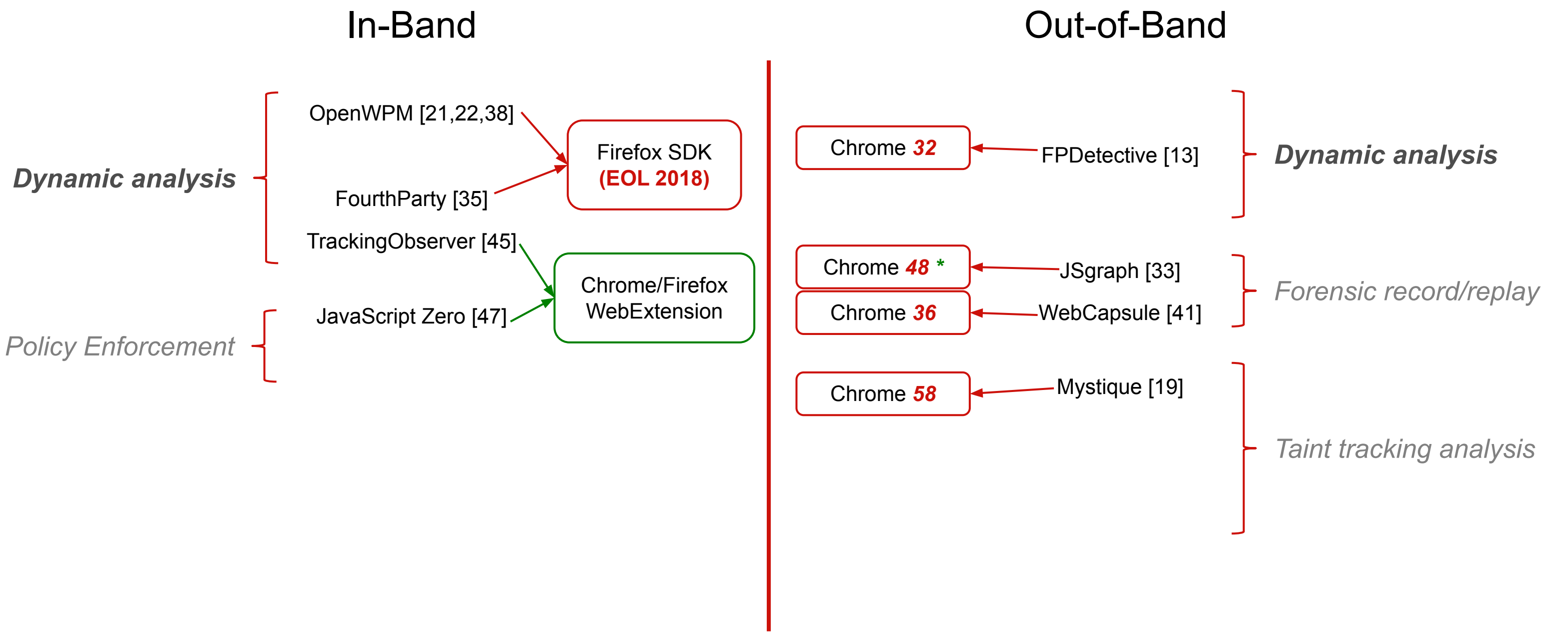
Dynamic analysis
- Li *et al.* [34]
- FPDetective [13]
- WebAnalyzer [48]

Forensic record/replay
- JSgraph [33]
- WebCapsule [41]

Taint tracking analysis
- Mystique [19]
- Lekies *et al.* [31,32]
- Stock *et al.* [51]
- Tran *et al.* [53]

NC STATE UNIVERSITY

Available Research Tools



In-Band vs. Out-of-Band

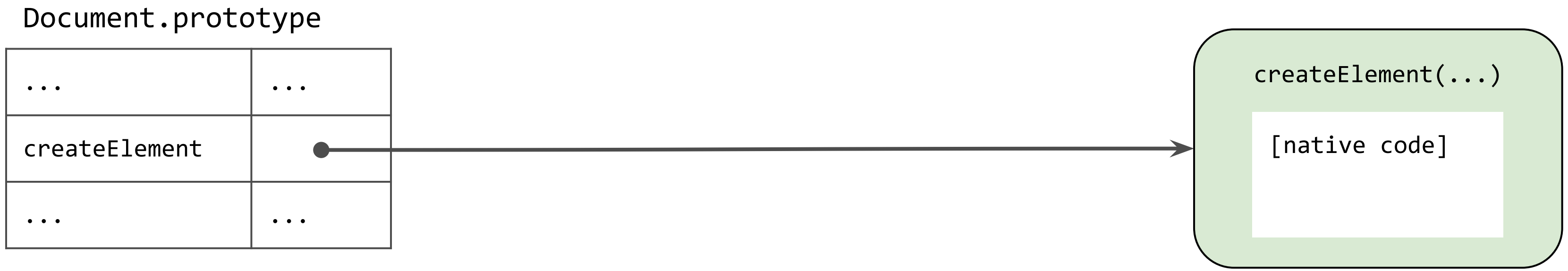
In-Band *(JS based)*

- a.k.a. “Monkey-patching” JS

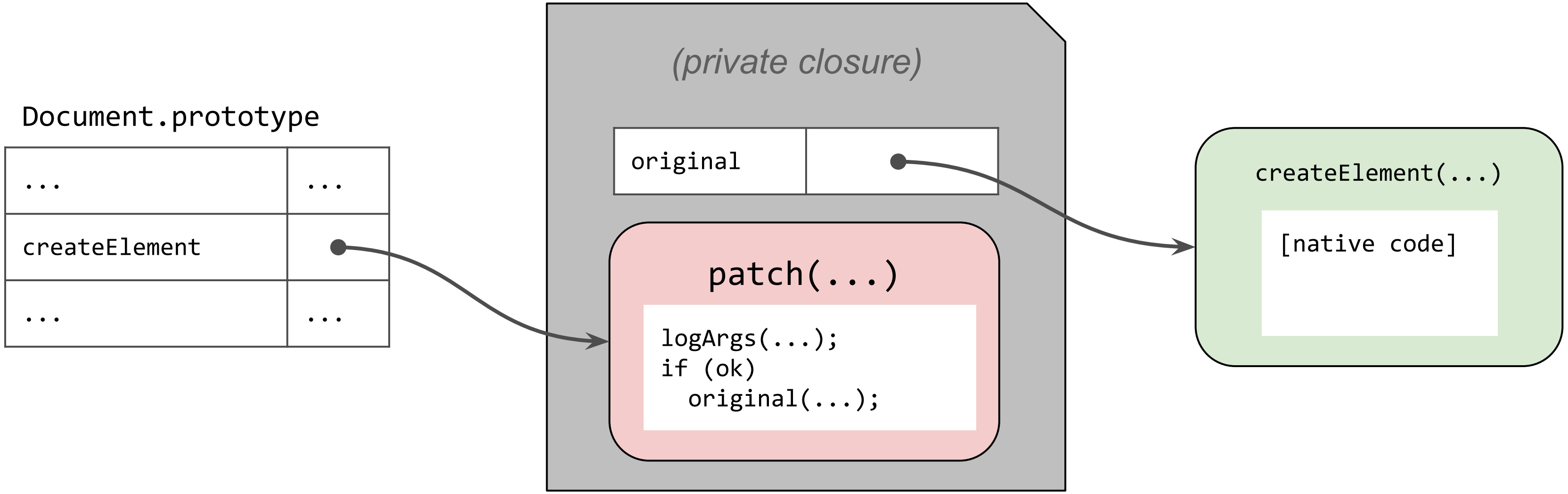
Out-of-Band *(browser based)*

- Modifying/adding C++ code

Background: Monkey Patching



Background: Monkey Patching



```
logArgs(...);
if (ok)
  original(...);
```

Advantage In-Band?

In-Band *(JS based)*

- a.k.a. “Monkey-patching” JS
- Pros:
  - Easy to construct
  - Easy to maintain
  - Portable across browsers

Out-of-Band *(browser based)*

- Modifying/adding C++ code
- Cons:
  - Hard to construct
  - Harder to maintain
  - Tied to one browser

But...

```
/* (all variable names original) */
var badWrite = !(document.write instanceof Function
                 && ~document.write.toString().indexOf('[native code]'));

/* (later on, among other logic checks) */
if (badWrite || o.append) {
    o.scriptLocation.parentNode.insertBefore(/* omitted for brevity */);
} else {
    document.write(div.outerHTML);
}
```

But… (cont'd)

```
function paranoidCreateElement(tag) {
  return document.createElement({
    toString: function() {
      var callers = new Error().stack.split('\n').slice(1);
      if (/at paranoidCreateElement /.test(callers[1])) {
        return tag; /* no patch */
      } else {
        throw new Error("evasive action!"); /* patched! */
      }
    },
  });
}
```

But… (cont'd)

```
/* (some names changed for clarity; cachedJSON is initially null) */
if (window.JSON && a.checkNativeCode(JSON.stringify) && a.checkNativeCode(JSON.parse))
    return window.JSON;

if (!cachedJSON) {
    var t = getInjectedIFrameElement();
    cachedJSON = t.contentWindow.JSON;
    var e = t.parentNode;
    e.parentNode.removeChild(e)
}

return cachedJSON;
```

All Things Considered

In-Band *(JS based)*

- a.k.a. “Monkey-patching” JS
- Pros:
  - Easy to construct
  - Easy to maintain
  - Portable across browsers
- Cons:
  - **Harder to hide**
  - **Race conditions**
  - **Unforgeable properties**
  - **Unproxiable objects**

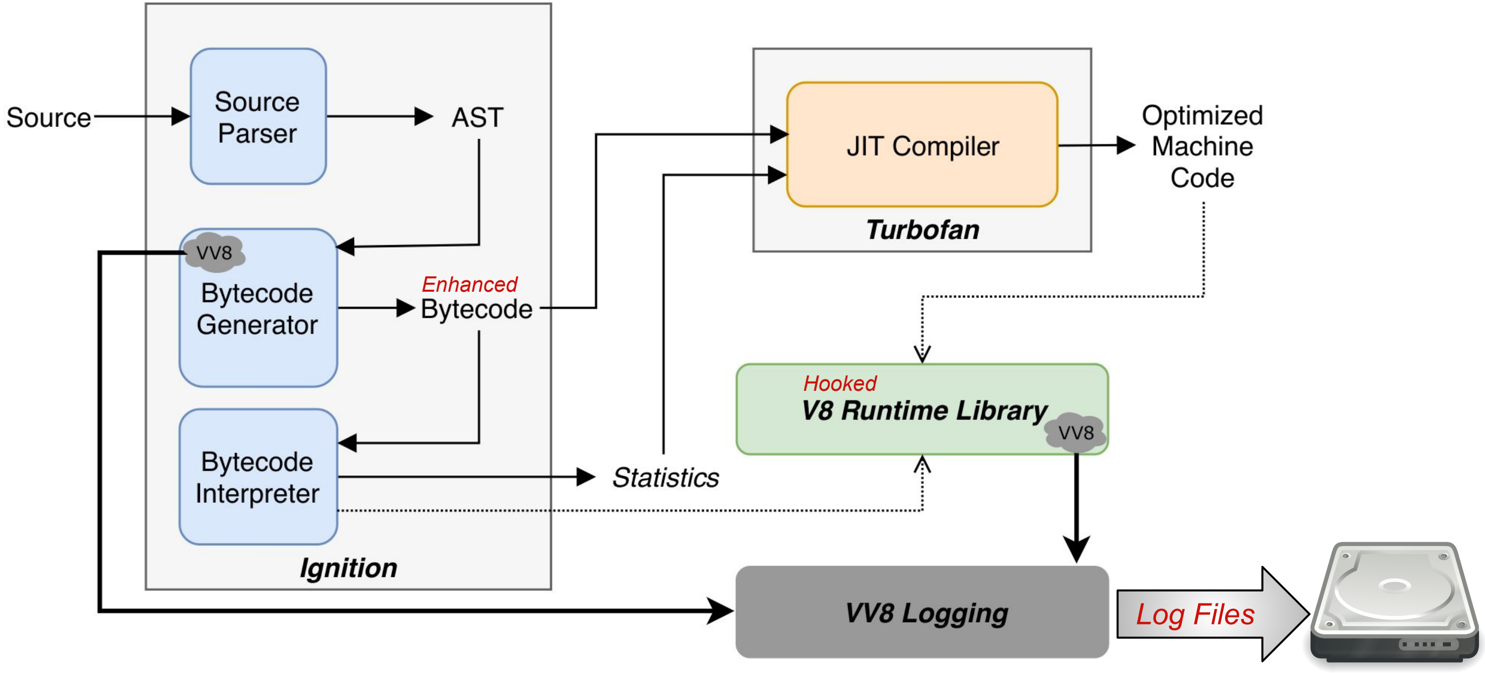
Out-of-Band *(browser based)*

- Modifying/adding C++ code
- Cons:
  - Hard to construct
  - Harder to maintain
  - Tied to one browser
- Pros:
  - **Hidden by default***
  - **Effectively no limitations**

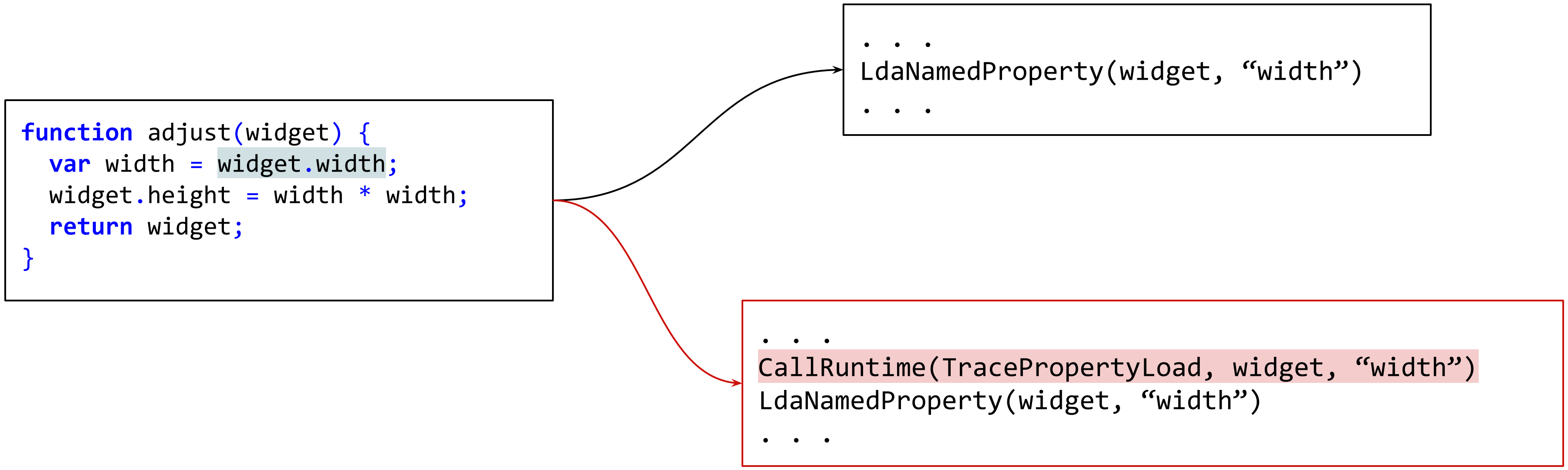
* *Modulo bugs and side channels*

Implementing VisibleV8

V8 Internals



Bytecode Injection

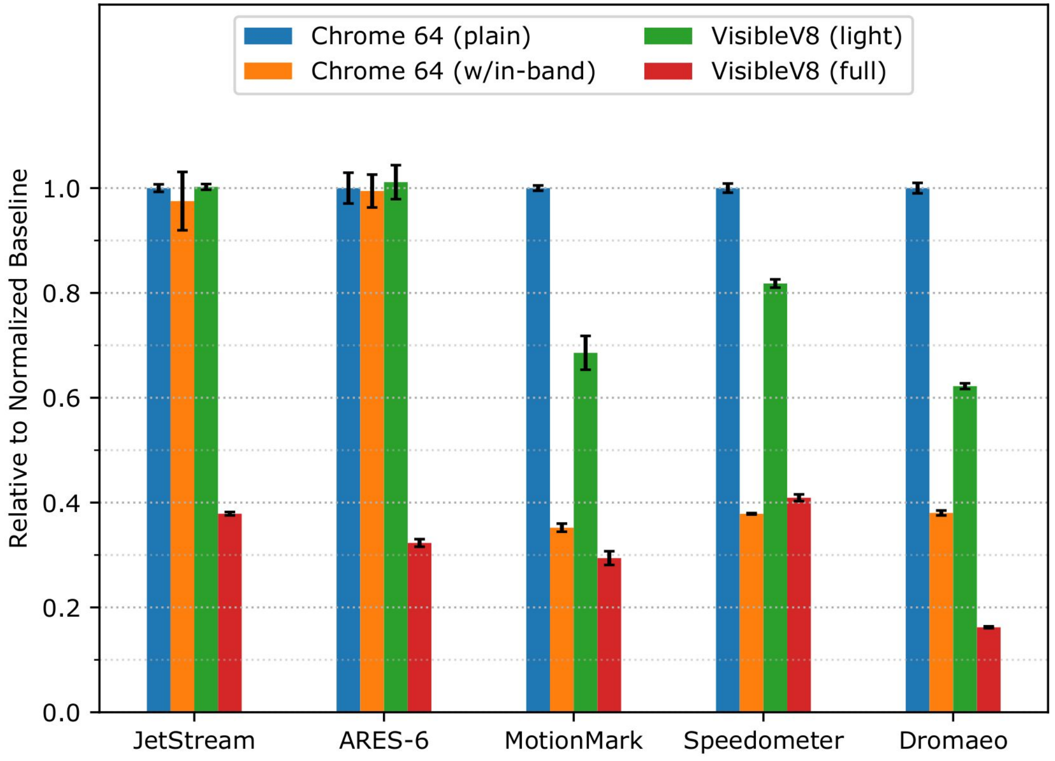


```
function adjust(widget) {
  var width = widget.width;
  widget.height = width * width;
  return widget;
}
```

```
. . .
LdaNamedProperty(widget, “width”)
. . .
```

```
. . .
CallRuntime(TracePropertyLoad, widget, “width”)
LdaNamedProperty(widget, “width”)
. . .
```

Performance Impact



- **"plain"**: baseline (no instrumentation)
- **"w/in-band"**: browser extension that hooks API function calls only
- **VV8 "light"**: VV8 build that hooks API function calls only
- **VV8 "full"**: VV8 build that hooks API function calls and property accesses

(higher is better)

Maintainability

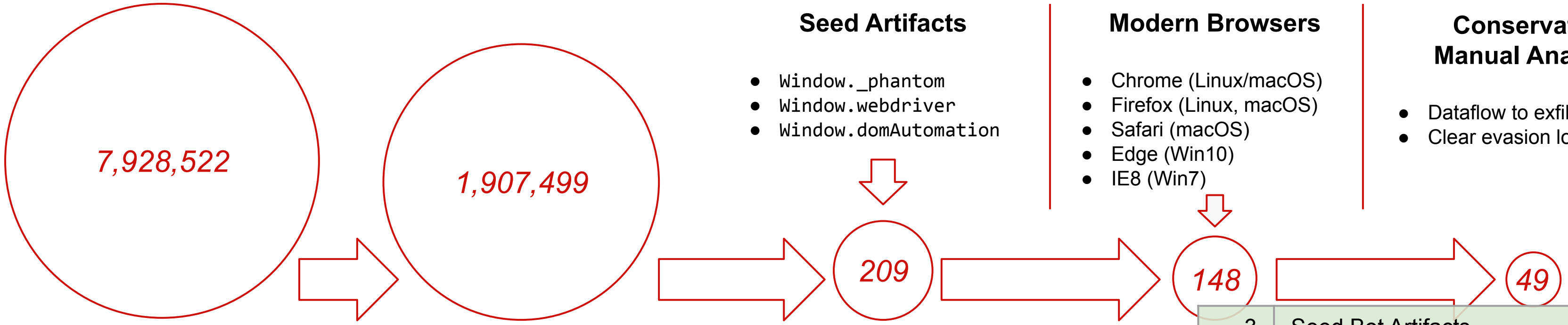
- Only minor revisions from Chrome 64 through 77
  - "Just in time" porting driven by project needs
  - Mostly syntactic trivialities caused by V8 refactors
- The Secret? Keep It Small & Simple! (KISS)
  - ~70 LoC changed/added in V8 proper
  - ~470 LoC added on the side for logging logic

Case Study: Bot-detection Artifact Discovery

“Bot Artifacts” Defined

```
detectExecEnv: function() {
  var e = "";
  return
    (window._phantom
      || /* more PhantomJS probes */)
      && (e += "phantomjs"),
    window.Buffer
      && (e += "nodejs"),
    window.emit
      && (e += "couchjs"),
    window.spawn
      && (e += "rhino"),
    window.webdriver
      && (e += "selenium"),
    (window.domAutomation || window.domAutomationController)
      && (e += "chromium-based-automation-driver"), e
},
```

Detection Workflow



Unique Non-IDL Properties Accessed

Mostly global variables accessed as properties of the `window` object

"Get-Only" Properties

Unique non-IDL properties that we never observed being "set"

Seed Artifacts

- `Window._phantom`
- `Window.webdriver`
- `Window.domAutomation`

Seed-Local Candidates

"Get-only" properties whose access sites are within a 1K-char radius of a seed-property access site

Modern Browsers

- Chrome (Linux/macOS)
- Firefox (Linux, macOS)
- Safari (macOS)
- Edge (Win10)
- IE8 (Win7)

Non-Browser Bot Artifact Candidates

Candidates not implemented by a major modern browser

Conservative Manual Analysis

- Dataflow to exfiltration
- Clear evasion logic

Confirmed Bot Artifacts

Properties used exclusively for probable bot detection/evasion

3	Seed Bot Artifacts
46	New Bot Artifact
10	Possible Bot Artifact
19	Device/Browser Fingerprint
46	Property Pollution/Iteration
11	Type Error/Misspelling
8	Missing Dependency
5	Other

The Rest of the Story

```
_ = window;
if (_["phantom"] || _["_phantom"] || _["callPhantom"] || _["__phantomas"]
    || _["Buffer"] || _["emit"] || _["spawn"]
    || _["webdriver"] || _["domAutomation"] || _["domAutomationController"]
) {}
else {
  location["reload"]();
}
```

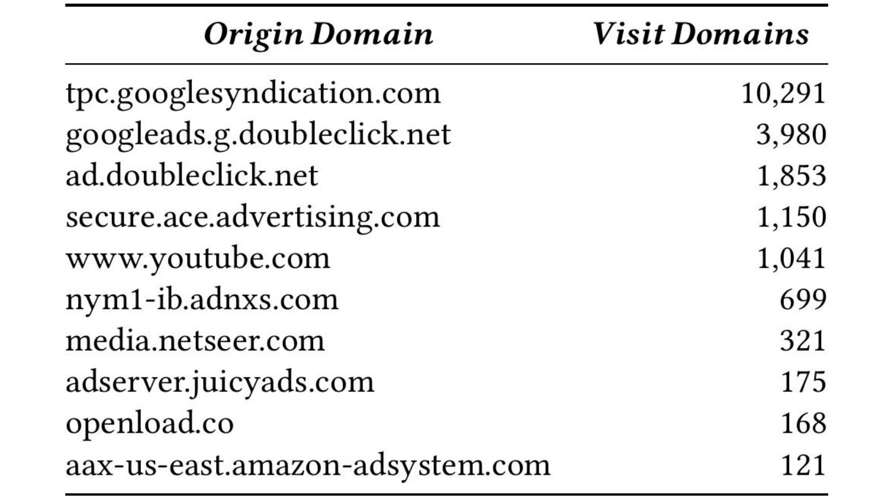
Reflections & Future Work

- Concurrent out-of-band success story: Brave’s AdGraph (IEEE S&P 2020)
- In development: integrating AdGraph and VV8
- Future work: detecting evasions at scale on a messy, chaotic Web
- **Maintenance commitment**: multiple projects at NCSU depend on VV8!

Takeaways

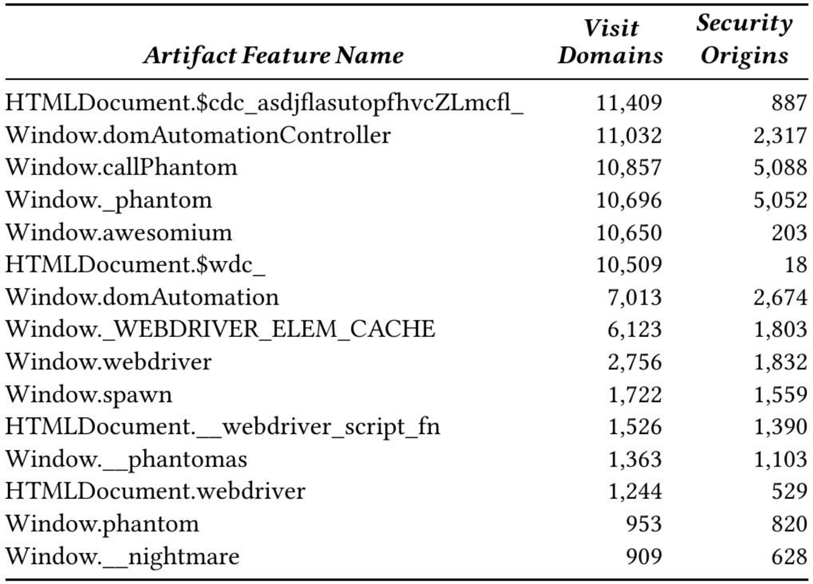
- Avoid in-band JS instrumentation: the limitations are serious
- Be aware that the Web may be measuring you back
- Check out VV8! Free (as in speech), maintained software available at https://kapravelos.com/projects/vv8

Result Summary



Origin Domain	*Visit Domains*
tpc.googlesyndication.com	10,291
googleads.g.doubleclick.net	3,980
ad.doubleclick.net	1,853
secure.ace.advertising.com	1,150
www.youtube.com	1,041
nym1-ib.adnxs.com	699
media.netseer.com	321
adserver.juicyads.com	175
openload.co	168
aax-us-east.amazon-adsystem.com	121

Table 6: Top security origin domains probing bot artifacts



Artifact Feature Name	*Visit Domains*	*Security Origins*
HTMLDocument.$cdc_asdjflasutopfhvcZLmcfl_	11,409	887
Window.domAutomationController	11,032	2,317
Window.callPhantom	10,857	5,088
Window._phantom	10,696	5,052
Window.awesomium	10,650	203
HTMLDocument.$wdc_	10,509	18
Window.domAutomation	7,013	2,674
Window._WEBDRIVER_ELEM_CACHE	6,123	1,803
Window.webdriver	2,756	1,832
Window.spawn	1,722	1,559
HTMLDocument.__webdriver_script_fn	1,526	1,390
Window.__phantomas	1,363	1,103
HTMLDocument.webdriver	1,244	529
Window.phantom	953	820
Window.__nightmare	909	628

Table 7: Most-probed bot artifacts