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...

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**
- OpenWPM [21,22,38]
- Snyder et al., 2016 [49]
- FourthParty [35]
- TrackingObserver [45]
- JavaScript Zero [47]
- Snyder et al., 2017 [50]

**Out-of-Band**
- Li et al. [34]
- FPDetective [13]
- WebAnalyzer [48]
- JSgraph [33]
- WebCapsule [41]
- Mystique [19]
- Lekies et al. [31,32]
- Stock et al. [51]
- Tran et al. [53]

**Dynamic analysis**

**Forensic record/replay**

**Taint tracking analysis**
Available Research Tools

**In-Band**
- OpenWPM [21,22,38]
- FourthParty [35]
- TrackingObserver [45]
- JavaScript Zero [47]
  - Firefox SDK (EOL 2018)

**Out-of-Band**
- Chrome 32
- FPDetector [13]
- Chrome 48 *
- JSgraph [33]
- Chrome 36
- WebCapsule [41]
- Chrome 58
- Mystique [19]
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

Document.prototype

<table>
<thead>
<tr>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>createElement</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Background: Monkey Patching

```
Document.prototype
... ...
createElement ...
... ...
```

```
original

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

(createElement(...)
[native code]

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

Source → Source Parser → AST

VV8 Bytecode Generator

Enhanced Bytecode

Bytecode Interpreter

Ignition → JIT Compiler → Turbofan

Optimized Machine Code

Hooked V8 Runtime Library

VV8 Logging

Log Files
Bytecode Injection

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

7,928,522

“Get-Only” Properties

Unique non-IDL properties that we never observed being “set”

1,907,499

Seed Artifacts

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

209

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)

148

Conservative Manual Analysis

- Dataflow to exfiltration
- Clear evasion logic

49

Non-Bot Artifact Candidates

19

Device/Browser Fingerprint

46

Property Pollution/Iteration

11

Type Error/Misspelling

8

Missing Dependency

5

Other
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

<table>
<thead>
<tr>
<th>Origin Domain</th>
<th>Visit Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpc.goglesyndication.com</td>
<td>10,291</td>
</tr>
<tr>
<td>googleads.g.doubleclick.net</td>
<td>3,980</td>
</tr>
<tr>
<td>ad.doubleclick.net</td>
<td>1,853</td>
</tr>
<tr>
<td>secure.ace.advertising.com</td>
<td>1,150</td>
</tr>
<tr>
<td><a href="http://www.youtube.com">www.youtube.com</a></td>
<td>1,041</td>
</tr>
<tr>
<td>nym1-ib.adnxs.com</td>
<td>699</td>
</tr>
<tr>
<td>media.netseer.com</td>
<td>321</td>
</tr>
<tr>
<td>adserver.juicyads.com</td>
<td>175</td>
</tr>
<tr>
<td>openload.co</td>
<td>168</td>
</tr>
<tr>
<td>aax-us-east.amazon-adsystem.com</td>
<td>121</td>
</tr>
</tbody>
</table>

Table 6: Top security origin domains probing bot artifacts

<table>
<thead>
<tr>
<th>Artifact Feature Name</th>
<th>Visit Domains</th>
<th>Security Origins</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTMLDocument.$cdc_3dijfiasuopfhlvZLmcl_</td>
<td>11,409</td>
<td>887</td>
</tr>
<tr>
<td>Window.domAutomationController</td>
<td>11,032</td>
<td>2,317</td>
</tr>
<tr>
<td>Window.callPhantom</td>
<td>10,857</td>
<td>5,088</td>
</tr>
<tr>
<td>Window.phantom</td>
<td>10,696</td>
<td>5,052</td>
</tr>
<tr>
<td>Window.awesomium</td>
<td>10,650</td>
<td>203</td>
</tr>
<tr>
<td>HTMLDocument.$wdc_</td>
<td>10,509</td>
<td>18</td>
</tr>
<tr>
<td>Window.domAutomation</td>
<td>7,013</td>
<td>2,674</td>
</tr>
<tr>
<td>Window._WEBDRIVER_ELEM_CACHE</td>
<td>6,123</td>
<td>1,803</td>
</tr>
<tr>
<td>Window.webdriver</td>
<td>2,756</td>
<td>1,832</td>
</tr>
<tr>
<td>Window.spawn</td>
<td>1,722</td>
<td>1,559</td>
</tr>
<tr>
<td>HTMLDocument.__webdriver_script_fn</td>
<td>1,526</td>
<td>1,390</td>
</tr>
<tr>
<td>Window.__phantomas</td>
<td>1,363</td>
<td>1,103</td>
</tr>
<tr>
<td>HTMLDocument.webdriver</td>
<td>1,244</td>
<td>529</td>
</tr>
<tr>
<td>Window.phantom</td>
<td>953</td>
<td>820</td>
</tr>
<tr>
<td>Window.__nightmare</td>
<td>909</td>
<td>628</td>
</tr>
</tbody>
</table>

Table 7: Most-probed bot artifacts