Staged Information Flow for JavaScript

Ravi Chugh, Jeff Meister, Ranjit Jhala, Sorin Lerner

UC San Diego
wsj.com

<textbox id="SearchBox">
<button onclick="doSearch(SearchBox.value)">
</button>

<script type="javascript">
searchUrl = "wsj.com/search?";
doSearc
var u = searchUrl + s;
document.location = u;
}
z = get("a.com/ad.js");
eval(z);
</script>
<script type="javascript">
searchUrl = "wsj.com/search?";
doSearch = function(s) {
    var u = searchUrl + s;
document.location = u;
}
</script>
<script type="javascript">
searchUrl = "wsj.com/search?";
doSearch = function(s) {
    var u = searchUrl + s;
document.location = u;
}
</script>

displayAd = function() {
    ...
} displayAd();
searchUrl = "evil.com/";
evil.com

- Script navigates to malicious page
- Exploits browser vulnerability
The Problem, Part 1

- Third-party code may affect sensitive data
  - e.g. writing `doc.location`
  - e.g. reading `doc.cookie`

- Information flow policies
  - e.g. integrity of `doc.location`
  - e.g. confidentiality of `doc.cookie`

- JavaScript difficulties
  - dynamic typing
  - first-class functions
  - objects, but no classes
  - prototypes
The Problem, Part 2

- Entire code not available until runtime
- Arrives in stages

```javascript
var doc = ...;
doc.location = "evil";
steal(doc.cookie);
```
Our Staged Approach: Server

Information Flow Policies

Confidentiality policy:
  x should not be read

Integrity policy:
  x should not be written
Our Staged Approach: Server

- Summarizes how loaded code must behave
- Syntactically enforceable for speed
Our Staged Approach: Client

Browser

- JavaScript Engine
  - Residual Policy Checker
    - ✓
    - ✗

context

residual policy

hole
<script type="javascript">

searchUrl = "wsj.com/search?";

doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
};
</script>
`wsj.com`

```javascript
searchUrl = "wsj.com/search?";

doSearch = function(s) {
  var u = searchUrl + s;
  document.location = u;
}

</script>
```
`<script type="javascript">
searchUrl = "wsj.com/search?";
doSearch = function(s) {
  var u = searchUrl + s;
  document.location = u;
}
</script>`
wsj.com

<script type="javascript">

searchUrl = "wsj.com/search?";
doSearc
h = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
</script>
wsj.com

<script type="javascript">
searchUrl = "wsj.com/search?";
doSearch = function(s) {
  var u = searchUrl + s;
document.location = u;
}
</script>

<textbox id="SearchBox">
<button onclick="doSearch(SearchBox.value)">
<no-read a.com/ad3.js>
doSearch("foo");
</no-read>
</button>
Outline

• Overview
• JavaScript Static Analysis
• Computing Residual Policies
• Additional Challenges
• Evaluation
• Analysis tracks information flow in program
• Flow-insensitive, set constraint-based
• Graph representation:
  – program constants, variables, edges
    \[0 \rightarrow x\]
  – special nodes for function declarations and calls
    \[\text{Fun} \quad \text{Fun}\]
searchUrl = "wsj.com/search?";
doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad1.js */
displayAd = function() { ... };
displayAd();
searchUrl = "wsj.com/search?”;

doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}

doSearch(SearchBox.value);

/* a.com/ad1.js */
displayAd = function() { ... };
displayAd();
searchUrl = "wsj.com/search??";

```javascript
doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad1.js */
displayAd = function() { ... };
displayAd();
```
searchUrl = "wsj.com/search?";

```javascript
var u = searchUrl + s;
document.location = u;
}
doSearcSearchBox.value);
/* a.com/ad1.js */
displayAd = function() {
  ...
};
displayAd();
```
searchUrl = "wsj.com/search?";
doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad1.js */
displayAd = function() { ... };
displayAd();
searchUrl = "wsj.com/search?";
doSearch = function(s) {
  var u = searchUrl + s;
  document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad1.js */
displayAd = function() { ... };
displayAd();
searchUrl = "wsj.com/search?";
doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad1.js */
displayAd = function() { ... };
displayAd();
searchUrl = "wsj.com/search?";
doSearch = function(s) {
  var u = searchUrl + s;
  document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad1.js */
displayAd = function() { ... };
displayAd();
searchUrl = "wsj.com/search?";

doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}

doSearch(SearchBox.value);

/* a.com/ad1.js */
displayAd = function() { ... };

displayAd();
searchUrl = “wsj.com/search?”;
doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad1.js */
displayAd = function() { ... };
displayAd();
searchUrl = "wsj.com/search?";
doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad1.js */
displayAd = function() { ... }; displayAd();
searchUrl = "wsj.com/search?";
doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad2.js is */

searchUrl = "evil.com/";
searchUrl = "wsj.com/search?";
doSearch = function(s) {
  var u = searchUrl + s;
  document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad2.js */
searchUrl = "evil.com";
searchUrl = “wsj.com/search?”;
doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad3.js */
doSearch(“foo”);
searchUrl = "wsj.com/search??";
doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
doSearch(SearchBox.value);

/* a.com/ad3.js */
doSearch("foo");
Outline

• Overview
• JavaScript Static Analysis
• Computing Residual Policies
• Additional Challenges
• Evaluation
```javascript
searchUrl = "wsj.com/search?”;
doSearch = function(s) {
    var u = searchUrl + s;
    document.location = u;
}
doSearch(SearchBox.value);
```
Residual Policies

• Difficulties:
  – Aliasing
  – First-class functions
  – Don’t want flow analysis in browser

• Solution:
  – Conservatively taint functions
  – Conservatively taint fields
Tainted Functions

- Transfer taints from parameters to functions

- Transfer taints from return values to functions

```javascript
// hole redefines foo
foo = function(t) {
    // reads t, hence cookie
}
foo(document.cookie);
```
Aliasing and Tainted Fields

- Residual policy misses future aliasing
- Conservative approach:
  if field f is tainted for some object, f tainted for all

```javascript
tmp = document;

z = tmp.cookie;

// reads z
```
Outline

• Overview
• JavaScript Static Analysis
• Computing Residual Policies
• Additional Challenges
• Evaluation
Objects

• Used pervasively in JavaScript
• Hence, analysis must be field-sensitive
• Encode “setter” and “getter” for field $f$ using

\[
F_{ld_f}(x) = \begin{cases} 
\{ f:1 \} \\
\end{cases} 
\]

• Fields can be dynamically added
• Initially assume no fields
• Iteratively add constraints until fixpoint

```javascript
x = { f:1};
x.g = 2;
```
Prototypes

• JavaScript uses prototype-based inheritance
• Intuitively, each object \( x \)
  – has a link to its parent
  – inherits parent’s fields

• Ensures each object has fields of its ancestors
Indirect Flows

if (document.cookie == “foo”) {
  y = 1;
}

• Propagate taints along indirect flow edges
• But not program values
Outline

• Overview
• JavaScript Static Analysis
• Computing Residual Policies
• Additional Challenges
• Evaluation
Implementation

• Flow analysis and residual policy generator
  – parse JavaScript (JSure)
  – generate set constraints (6,000 lines of OCaml)
  – solve constraints (Banshee + 400 lines of C)

• Stand-alone residual policy checker
  – not yet incorporated into browser

• JavaScript collector
  – Firefox extension (500 lines of JavaScript)
Experimental Setup

- Collect JavaScript for Alexa top 100 web sites

Context:
all server code

Hole:
all third-party code

97/100 have JavaScript
63/97 have holes
Experimental Setup

• Information flow analysis on context + hole

• Compute residual policy, check it on hole
Scalability of Full Analysis

- Average running time: 9.9 seconds
- 80% run in <12 seconds
Average Running Times

Full Analysis

9.9 sec

Staged Analysis

14.0 sec
0.13 sec
Results of Analysis: Full

• Hole satisfies cookie policy?  ✓ 30  ✗ 32
Results of Analysis: Staged

- Hole satisfies cookie policy? ✓ 30 ✗ 32

Residual checker:
- 26/30 safe
- Imprecision:
  4 false positives
Future Work

- Context-sensitivity
- Dynamically-constructed field names
- Test more complicated policies
- Embed residual policy checker in browser
Related Work

• Information flow
  – type systems
  – dynamic instrumentation

• JavaScript analysis
  – types [Thiemann 05, Anderson et al. 05]
  – dynamic policies [Chander et al. 07]
  – static analysis [Guarnieri/Livshits 09]

• Browser security
  – finer-grained interaction between scripts [Howell et al. 07]
Summary

• JavaScript static analysis is scalable

• Residual checks are fast enough for client

• Residual policies precisely capture information flow
Thanks!
Extra Slides
Confidentiality of $x$: $x$ should not affect hole variables

directly or indirectly

holeVar = $x$;  
if (x) { holeVar = 1 };

Integrity of $x$: hole variables should not affect $x$

directly or indirectly

$x$ = holeVar;  
if (holeVar) { $x$ = 1 };
Fields
Running Times

<table>
<thead>
<tr>
<th></th>
<th>cookie policy</th>
<th>location policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow analysis on context + hole</td>
<td>9.9</td>
<td>10.7</td>
</tr>
<tr>
<td>Computing residual policy</td>
<td>14.0</td>
<td>28.4</td>
</tr>
<tr>
<td>Checking residual policy</td>
<td>0.13</td>
<td>0.12</td>
</tr>
</tbody>
</table>

- Full analysis too slow to run on client
- Quick to compute residual policy on server
- Small run-time overhead to check
  - running time includes parsing time
  - parser is not optimized for speed
### Results of Staged Analysis

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>26</td>
<td>49</td>
</tr>
<tr>
<td>✓</td>
<td>❌</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>❌</td>
<td>❌</td>
<td>32</td>
<td>5</td>
</tr>
<tr>
<td>❌</td>
<td>✓</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Residual policy usually agrees with full information flow analysis
- Imprecision from tainted functions/fields
- No false negatives