ADsafety
Type-based Verification of JavaScript Sandboxing

Joe Gibbs Politz
Spiridon Aristides Eliopoulos
Arjun Guha
Shriram Krishnamurthi
OUTLOOK: NEGATIVE

U.S. Credit Downgrade Stokes Recession Fears
5 of your friends are already here!
Clare Patterson
Kate MG
Jennifer B. Beinart
Ian Wolfcat Atha

Invite your friends to HuffPost
Add up to 40 of your friends by clicking on their pictures below.

Find Friends: Start Typing a Name

Filter Friends: All Selected (0)

Aaron T Myers
Brown

Aditya Dhandhania
JWU Prov...

Aheli Purkayastha
Universi...

Adam Khalil Grinnell

Aditya Pai
Georgia ...

Akili Thomas
Grinnell

Adityarup Chakraborty
Grinnell

Albert Varma
UVA

Send HuffingtonPost Invitation
<table>
<thead>
<tr>
<th>Name</th>
<th>Path</th>
<th>Method</th>
<th>Status Text</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-6fTutip1SMLM2.js</td>
<td>pixel.quantserve.com/seg</td>
<td>GET</td>
<td>304 Not Modif</td>
<td>application/javascript</td>
<td>54.18KB</td>
</tr>
<tr>
<td>tw_dfp_adsonar.js</td>
<td>js.adsonar.com/js</td>
<td>GET</td>
<td>200 OK</td>
<td>application/x-javascript</td>
<td>6.12KB</td>
</tr>
<tr>
<td>beacon.js</td>
<td>b.scorecardresearch.com</td>
<td>GET</td>
<td>200 OK</td>
<td>application/x-javascript</td>
<td>3.37KB</td>
</tr>
<tr>
<td>quant.js</td>
<td>edge.quantserve.com</td>
<td>GET</td>
<td>200 OK</td>
<td>text/html</td>
<td>5.14KB</td>
</tr>
<tr>
<td>ad_html_wh.php</td>
<td>s.huffpost.com/assets</td>
<td>GET</td>
<td>200 OK</td>
<td>application/x-javascript</td>
<td>2.10KB</td>
</tr>
<tr>
<td>aceUAC.js</td>
<td>uac.advertising.com/wrapper</td>
<td>GET</td>
<td>200 OK</td>
<td>application/x-javascript</td>
<td>14.81KB</td>
</tr>
<tr>
<td>dref=https%25A%252F%252F%252Fwwwr1-ads.ace.advertising.com/site=80</td>
<td>GET</td>
<td>304 Not Modif</td>
<td>application/x-javascript</td>
<td>39.59KB</td>
<td>225B</td>
</tr>
<tr>
<td>imb8Z50C5TH.js</td>
<td>s-static.ak.facebook.com/rsr.php/v</td>
<td>GET</td>
<td>304 Not Modif</td>
<td>application/x-javascript</td>
<td>5KB</td>
</tr>
</tbody>
</table>

Who is running code in your browser?
Who is running code in your browser?
Who is running code in your browser?
the host
you visit
the host you visit
the host you visit

same JavaScript context

the ad server
the host you visit

the ad server

<iframe>
the host you visit

the ad server

<iframe>

top.location.href
All are defining safe sub-languages
eval
wrap(e)

“filters”

“rewriters,“

wrap

— Maffeis, Mitchell, and Taly, ESORICS 2009
untrusted widget

eval

ADSAFE.get(obj, x)
• 1,800 LOC adsafe.js library
• 50 calls to three kinds of assertions
• 40 type-tests
• 5 regular-expression based checks
• 60 privileged DOM method calls
• 1,800 LOC adsafe.js library
• 50 calls to three kinds of assertions
• 40 type-tests
• 5 regular-expression based checks
• 60 privileged DOM method calls
Type-based Verification of ADsafe
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;

2. Widgets cannot obtain direct references to DOM nodes;
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;
2. Widgets cannot obtain direct references to DOM nodes;
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;
2. Widgets cannot obtain direct references to DOM nodes;
3. Widgets cannot affect the DOM outside of their subtree; and
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;

2. Widgets cannot obtain direct references to DOM nodes;

3. Widgets cannot affect the DOM outside of their subtree; and
**Definition 1 (ADsafety):** If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;

2. Widgets cannot obtain direct references to DOM nodes;

3. Widgets cannot affect the DOM outside of their subtree; and

4. Multiple widgets on the same page cannot communicate.
Goal: Verify ADsafe

ADSAFE.get

ADSAFE.get(obj, x)
Goal: Verify ADsafe

ADSAFE.get

ADSAFE.get(obj, x)

untrusted, but passes JSLint
Goal: Verify ADsafe

ADSAFE.get

Goal: model JSLint

untrusted, but passes JSLint

ADSAFE.get(obj, x)
JSLint ensures:

no DOM references

"Widgets cannot obtain direct references to DOM nodes."
ADsafe ensures: only “safe” methods on bunches

bunch = {
  __nodes__ : array of nodes,
  append: function ..., 
  getText: function ..., 
  ... 20 functions
}

JSLint ensures: no DOM references

“Widgets cannot obtain direct references to DOM nodes.”
bunch = {
  __nodes__: array of nodes,
  append: function ..., 
  getText: function ..., 
  ... 20 functions
}
bunch = {
  __nodes__: array of nodes,
  append: function ...,
  getText: function ...,
  ... 20 functions
}

ADsafe ensures:
only "safe" methods on bunches

JSLint ensures:
__nodes__ is "private"

"Widgets cannot obtain direct references to DOM nodes."
bunch = {
  __nodes__ : array of nodes,
  append: function ...,  
  getText: function ...,  
  ... 20 functions
}

ADsafe ensures: only “safe” methods on bunches

JSLint ensures: __nodes__ is “private”

bunch.append(...)

“Widgets cannot obtain direct references to DOM nodes.”
JSLint ensures:
- no DOM references

ADsafe ensures:
- only “safe” methods on bunches
- __nodes__ is “private”

ADsafe ensures:
- DOM nodes are not returned

bunch = {
  __nodes__: array of nodes,
  append: function ...
  getText: function ...
  ... 20 functions
}

"Widgets cannot obtain direct references to DOM nodes.”
Goal 1: model JSLint

Goal 2: Verify ADsafe

untrusted, but passes JSLint

ADSAFE.get

ADSAFE.get(obj, x)
var n = 6
var s = "a string"
var b = true
var n = 6
var s = "a string"
var b = true

Widget := Number + String + Boolean + Undefined + Null +
Widget := Number + String + Boolean + Undefined + Null +
Widget := Number + String + Boolean + Undefined + Null +

{ x: 6, b: "car" }
Widget := Number + String + Boolean + Undefined + Null + ★: Widget

{ x: 6, b: "car" }
{ nested: { y: 10, b: false } }
Widget := Number + String + Boolean + Undefined + Null +
★: Widget
__nodes__: Array<Node>
caller: 🕽
prototype: 🕽
...

{ x: 6, b: "car" }
{ nested: { y: 10, b: false } }
{ __nodes__: 90 }
myObj.prototype = { };
{ x: 6, b: "car" }
{ nested: { y: 10, b: false } }
{ __nodes__: 90 }
myObj.prototype = { };
function foo(x) { return x + 1; }
foo(900)
foo.w = "functions are objects"
["array", "of", "strings"]
/regular[ \t]*expressions/

Widget := Number + String + Boolean + Undefined + Null +
★: Widget
__nodes__: Array<Node>
caller: 껑
prototype: 껑
...
  code : Widget × ... → Widget
__proto__: Object + Function + Array + ...
JSLint

widgets that pass JSLint

Widget type-checker

typable widgets
JSLint

Claim:
evidence: 1,100 LOC of tests

or, passing JSLint \(\Rightarrow\) Widget-typable
Claim:
evidence: 1,100 LOC of tests

or, passing JSLint $\Rightarrow$ Widget-typable
Goal 1: model JSLint

Goal 2: Verify ADsafe

untrusted, but passes JSLint
window.setTimeout(callback, delay);
window.setTimeout(callback, delay);

ADSAFE.later = function(callback, delay) {
    if (typeof callback !== "function") {
        throw "expected function";
    }
    window.setTimeout(callback, delay);
};

/*: Widget × Widget → Widget */

window : {
    eval: ☠,
    setTimeout : (Widget × ... → Widget) × Widget → Undefined,
    ...
  }

 ADSAFE.later = function(callback, delay) {
    if (typeof callback !== "function") {
        throw "expected function";
    }
    window.setTimeout(callback, delay);
}
window : {
  eval: ☠,
  setTimeout : (Widget × ... → Widget) × Widget → Undefined,
}

 ADSAFE.later = function(callback, delay) {
  if (typeof callback !== "function") {
    throw "expected function";
  }
  window.setTimeout(callback, delay);
}
window : {
    eval: ☠,
    setTimeout : (Widget × ... → Widget) × Widget → Undefined,
    ...
}

ADSAFE.later = function(callback, delay) {
    if (typeof callback !== "function") {
        throw "expected function";
    }
    window.setTimeout(callback, delay);
}

/*: Widget × Widget → Widget */
window : {
  eval: ☠,
  setTimeout : (Widget × ... → Widget) × Widget → Undefined,
  ...
}

/*: Widget × Widget → Widget */
ADSAFE.later = function(callback, delay) {
  if (typeof callback !== "function") {
    throw "expected function";
  }
  window.setTimeout(callback, delay);
}

This is just one kind of if-split we handle.

adsafe.js

JSLinted widget
adsafe.js

JSLinted widget
JSLint model + Typable widgets

= ... Type-checked ADsafe
var fakeNode = {
  tagName: "div",
  appendChild: function(elt) {
    var win = elt.ownerDocument.defaultView;
    win.eval("alert('hacked')");
  }
};
var fakeNode = {
  tagName: "div",
  appendChild: function(elt) {
    var win = elt.ownerDocument.defaultView;
    win.eval("alert('hacked')");
  }
};

var fakeBunch = { __nodes__: [fakeNode] };
var fakeNode = {
  tagName: "div",
  appendChild: function(elt) {
    var win = elt.ownerDocument.defaultView;
    win.eval("alert('hacked')");
  }
};

var fakeBunch = { __nodes__: [fakeNode] };  // Accepted by JSLint

var fakeBunch = { '__nodes__': [fakeNode] };  // Rejected by JSLint

type error: expected Array<HTML>, received Array<Widget>
WrappedElt.prototype.style = function(name, val) {
    var regexp = new RegExp("url");
    if (regexp.test(val)) {
        return error();
    }
    ... this.__node__.style[name] = val ...
}
WrappedElt.prototype.style = function(name, val) {
  var regexp = new RegExp("url");
  if (regexp.test(val)) {
    return error();
  }
  this.__node__.style[name] = val ...
}
```javascript
var firstCall = true;
var badName = {
  toString: function() {
    if (firstCall) {
      firstCall = false;
      return "font";
    } else {
      return "url('/evil.xml')";
    }
  }
};
```
/** Widget × Widget → Widget */
WrappedElt.prototype.style = function(name, val) {
  var regexp = new RegExp("url");
  if (regexp.test(val)) {
    return error();
  }
  ... this.__node__.style[name] = val ...
}

var firstCall = true;
var badName = {
  toString: function() {
    if (firstCall) {
      firstCall = false;
      return "font";
    } else {
      return "url('/evil.xml')";
    }
  }
};

Fix:
check_string assertion inserted here, and in 16 other places

Passed safety check
returns bad value
**Definition 1 (ADsafety):** If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;

2. Widgets cannot obtain direct references to DOM nodes;

3. Widgets cannot affect the DOM outside of their subtree; and

4. Multiple widgets on the same page cannot communicate.
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;

2. Widgets cannot obtain direct references to DOM nodes;

3. Widgets cannot affect the DOM outside of their subtree; and

4. Multiple widgets on the same page cannot communicate.
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;

2. Widgets cannot obtain direct references to DOM nodes;

3. Widgets cannot affect the DOM outside of their subtree; and

4. Multiple widgets on the same page cannot communicate.
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf; 

2. Widgets cannot obtain direct references to DOM nodes; 

3. Widgets cannot affect the DOM outside of their subtree; and 

4. Multiple widgets on the same page cannot communicate.
Definition 1 (ADsafety): If all embedded widgets pass JSLint, then:

1. Widgets cannot load new code at runtime, or cause ADsafe to load new code on their behalf;

2. Widgets cannot obtain direct references to DOM nodes;

3. Widgets cannot affect the DOM outside of their subtree; and

4. Multiple widgets on the same page cannot communicate.
Caveats:

- 11 LOC unverified
- subtree property unverified
Proofs for JavaScript?

JavaScript program \( \xrightarrow{\text{desugar}} \) \( \lambda_{\text{JS}} \) program

Proofs for \( \lambda_{\text{JS}} \).
Proofs for JavaScript?

JavaScript program

SpiderMonkey, V8, Rhino

“their answer”

dsugar

identical for Mozilla JS test suite*

λ program

100 LOC interpreter

“our answer”

Proofs for \( \lambda_{JS} \).

banned = {
    'arguments': true,
    'callee': true,
    'caller': true,
    'eval': true,
    'prototype': true,
    'stack': true,
    'unwatch': true,
    'valueOf': true,
    'watch': true
}

function reject_global(that) {
    if (that.window) {
        error();
    }
}

if (/url/i.test(string_check(value[i]))) {
    error('ADsafe error. ');
} and other patterns...
banned = {
    'arguments' : true,
    callee      : true,
    caller      : true,
    constructor : true,
    'eval'      : true,
    prototype   : true,
    stack       : true,
    unwatch     : true,
    valueOf     : true,
    watch       : true
}

function reject_global(that) {
    if (that.window) {
        error();
    }
}

if (/url/i.test(string_check(value[i]))) {
    error('ADsafe error. ');
}

... can be succinctly expressed with types

Widget := Number + String + Boolean + Undefined + Null +

★: Widget
__nodes__: Array<Node>
 caller: ★
 prototype: ★
...
 code : Widget × ... → Widget
__proto__: Object + Function + Array + ...
Conclusion

1. Model sandbox as a type system

2. Object types for JavaScript (★ and ☠)

3. Proofs over tractable JavaScript semantics
Extra Slides
function F() {};

ADSAFE.create = typeof Object.create === 'function' ? Object.create : function (o) {
    F.prototype = typeof o === 'object' && o ? o : Object.prototype;
    return new F();
};

/*@: (banned → True) & (not_banned → False) */
function reject_name(name) {
    return banned[name] ||
        (typeof name !== 'number' || name < 0) &&
        (typeof name !== 'string' || name.charAt(0) === '_' ||
            name.slice(-1) === '_' || name.charAt(0) === '-');
}
Theorems

Lemma 1 (Type Preservation)  If, for an expression e, type T, environment Γ and abstract heap Σ,
1. \( \Sigma \vdash \sigma \),
2. \( \Sigma; \Gamma \vdash e : T \), and
3. \( \sigma e \rightarrow \sigma' e' \);
then there exists a \( \Sigma' \) with \( \Sigma' \vdash \sigma' \) and \( \Sigma'; \Gamma \vdash e' : T \).

Theorem 1 (ADsafety)  For all widgets p, if
1. all subexpressions of p are Widget-typable,
2. \( \text{adsafe.js} \) is typable,
3. \( \text{adsafe.js} \) runs before p, and
4. \( \sigma p \rightarrow \sigma' p' \) (single-step reduction),
then at every step \( p' , \ p' \) also has the type Widget.
Full Widget Type

$$\text{Widget} = \mu \alpha.$$

$$\text{Full Widget Type}$$

$$\text{proto : } \text{UBunch} \cup \text{Array} \cup \text{RegExp}$$

$$\cup \text{String} \cup \text{Number} \cup \text{Boolean},$$

$$\ast : \alpha,$$

$$\text{code : } [\text{Global} \cup \alpha] \alpha \cdots \to \alpha,,$$

$$\text{__nodes__} : \text{Array}(\text{HTML}) \cup \text{Undefined},$$

$$\text{__star__} : \text{Boolean} \cup \text{Undefined},$$

$$\text{caller} : \alpha, \text{callee} : \alpha,$$

$$\text{eval} : \alpha, \text{prototype} : \alpha,$$

$$\text{watch} : \alpha, \text{constructor} : \alpha,$$

$$\text{__proto__} : \alpha, \text{unwatch} : \alpha,$$

$$\text{arguments} : \alpha, \text{valueOf} : \text{Absent},$$

$$\text{toString} : \text{Absent}.$$