To Dream the Impossible Dream: Toward Security Analysis for JavaScript

Julian Dolby
IBM Thomas J. Watson Research Center
Seoul National University, Korea, October 16, 2014

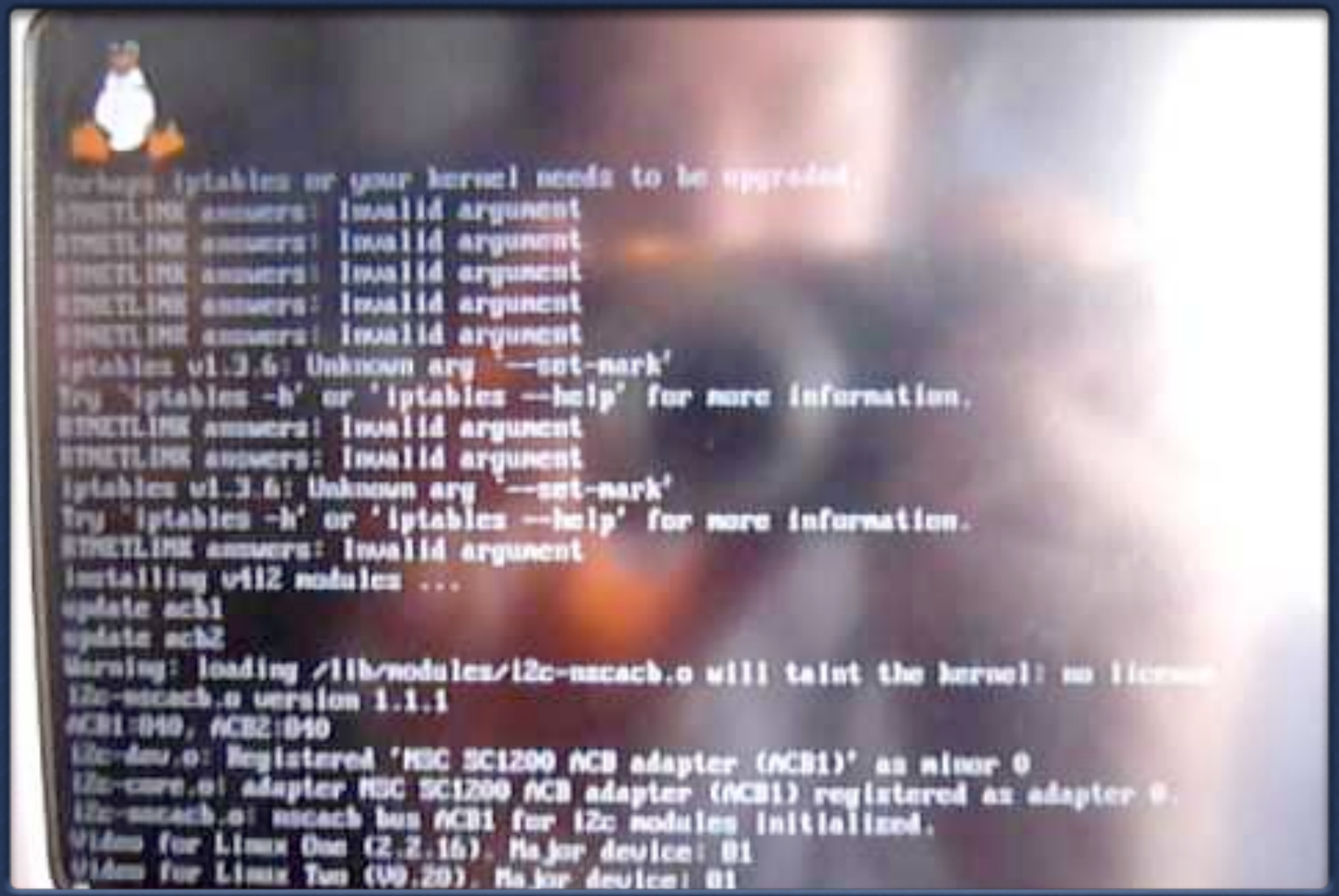
Based on [ECOOP12], [PLDI13], [ICSE13], ongoing work

Many Colleagues

Satish Chandra (Samsung)
Asger Feldhaus (Aarhus)
Salvatore Guarneri (Google)
Manu Sridharan (Samsung)
Max Schaefer (Semmle)
Frank Tip (Samsung)

Work done at IBM Thomas J. Watson Research Center

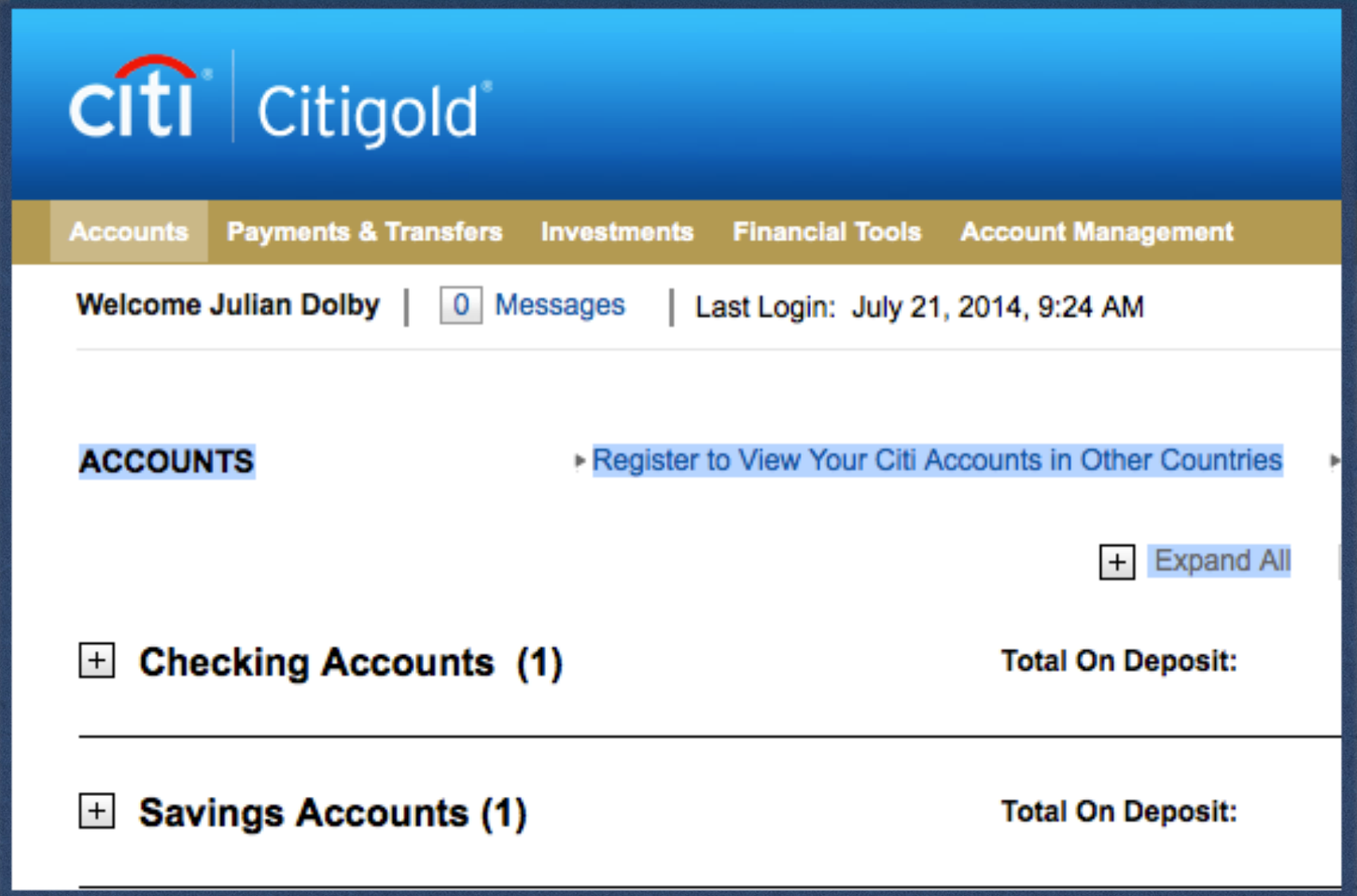
Some software works...



...and some does not

Two Classes of Software

- *"Important" software*
  - *Avionics, Antilock brakes, Medical, Financial*
    - *Deaths or major losses when it breaks*
- *Other software*
  - *Inflight entertainment, Social media*
  - *Mostly just causes annoyance*



"Important" software...

```
var f = function(){
          $jq(function($){
        if($jq("#InterdictionOverlayContent").jfpwidget() === undefined){
            $jq("#InterdictionOverlayContent").jfpwidget(new CJW.jfp.widget.Overlay(
{
        wrapperSet: "#InterdictionOverlayContent",
        hideTitleBar: false,
        dialog: true
    },
{
        draggable: false,
        resizable: false,
        position: ["center", 140],
        width: 550,
        height: 420,
        shadowOffset: 20,
        hideTitleBar: false,
        shadow:true,
        modal: true,
        beforeClose: checkStatusCsq
    },
{}));
        }
     }
      );
};
if (window.$RDY){
    $RDY('CJW.jfp.widget.Overlay',f);
}
else{
    f();
}
```

...has changed

JavaScript is "Important"

- *Web sites dominate life*
  - *banking, healthcare, shopping, communication...*
    - *important data and transactions*
- *Modern Web sites are implemented in JavaScript*
  - *rich client applications embody business logic*
  - *heavy use of rich frameworks, notably* `jQuery`

Improve Software Quality

program analysis based on call graphs

Traditional Call Graphs

- *Approximates targets at all call sites*
- *Type-based graphs simple but imprecise*

```
class Foo { void fun(); }
class Bar extends Foo { void fun(); }
Foo x = new Bar(); x.fun();
```
Foo.fun, Bar.fun

- *Dataflow-based graphs more precise, complex*

```
class Foo { void fun(); }
class Bar extends Foo { void fun(); }
Foo x = new Bar(); x.fun();
```
Bar.fun

Traditional Call Graphs

- *Dynamic languages complicate analysis*
- *Type-based graphs not applicable without static types*

```
x.fun = function foo {… }
var fun = x.f;
fun();
```

- *Dataflow-based graphs can track function objects*

```
x.fun = function foo {… }
var fun = x.f;
fun();
```

Traditional Call Graphs

- *JavaScript further challenges program analysis*
  - *heavy use of reflection, i.e. eval*
  - *first-class property names*

`x.s = y`

$$\frac{o \in pt(x)}{pt(y) \subseteq pt(o.\texttt{s})} \quad [\textsc{StoreField}]$$

`x[v] = y`

$$\frac{o \in pt(x) \qquad \texttt{s} \in pt(v)}{pt(y) \subseteq pt(o.\texttt{s})} \quad [\textsc{StoreField}]$$

- *Dynamism worsens asymptotic complexity*

Traditional Call Graphs

- *Much first-class property access in frameworks*

```
jQuery.fn = {
 extend: function ext(obj) {
  for(var p in obj)
    jQuery.fn[p] = obj[p];
 }
};
```

`p`: *fields of obj, e.g.* `"x","y", "z"`

`obj[p]`: *values of field, e.g.* `obj[x]=4,obj[y]="a"`

`jQuery.fn[p] = obj[p]` *assigns to all fields in* `p`

Correlation Tracking

- *Correlated first-class property reads and writes*

```
jQuery.fn = {
 extend: function ext(obj) {
  for(var p in obj)
    jQuery.fn[p] = obj[p];
 }
};
```

Same value per iteration

- *Need separate analysis per loop iteration*
  - *apply context sensitivity*
  - *extract loop body into separate function*

Correlation Tracking

```
jQuery.fn = {
 extend: function ext(obj) {
  for(var p in obj)
   jQuery.fn[p] = obj[p];
 }
};
```

```
if (p == "x") {
 jQuery["x"] = obj["x"]
} else if (p == "y" ) {
 jQuery["x"] = obj["x"]
}
```

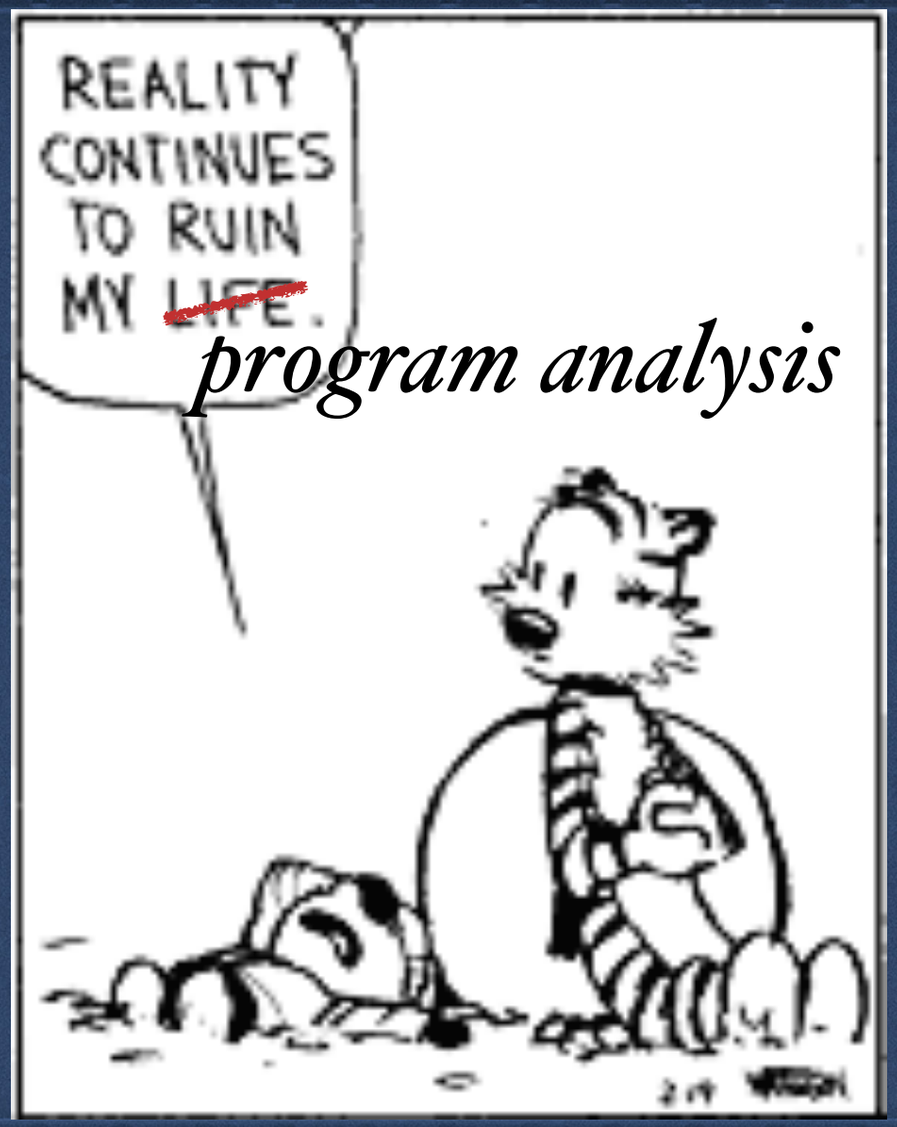
Implemented using context-sensitivity

Traditional Call Graphs

Framework	Baseline$^-$	Baseline$^+$	Correlations$^-$	Correlations$^+$
dojo	* (*)	* (*)	3.1 (30.4)	6.7 (*)
jquery	*	*	78.5	*
mootools	0.7	*	3.1	*
prototype.js	*	*	4.4	4.5
yui	*	*	2.2	2.1

+/- means with/without modeling of call and apply

- *Correlation tracking greatly aids analysis*
- *Other JavaScript dynamism still problematic*
- *Framework-based code beyond state of the art*



program analysis

Dynamic Determinacy

Runtime information to aid static analysis

Actual JQuery Code

```
var e = ("blur,focus,load,resize,scroll,unload," +
 "click,dblclick,mousedown,mouseup,mousemove," +
"mouseover,mouseout,change,reset,select,submit," +
“keydown,keypress,keyup,error").split(",");

for(var i=0;i<e.length;i++) new function(){
      var o = e[i];
      jQuery.fn[o] = function(f){
            return f ? this.bind(o, f) : this.trigger(o); };
      jQuery.fn["un"+o] = function(f){
            return this.unbind(o, f); };
      jQuery.fn["one"+o] = function(f){
            return this.each(function(){
                  var count = 0;
                  jQuery.event.add(this, o, function(e){
                        if(count++) return;
                        return f.apply(this, [e]); }); }); }; };
```

Dynamic Behavior

```
jQuery.fn.blur = function () { ... };
jQuery.fn.unblur = function () { ... };
jQuery.fn.oneblur = function () { ... };
jQuery.fn.focus = function () { ... };
jQuery.fn.unfocus = function () { ... };
jQuery.fn.onefocus = function () { ... };
jQuery.fn.load = function () { ... };
jQuery.fn.unload = function () { ... };
jQuery.fn.oneload = function () { ... };
```

- *Dynamic code is completely constant at runtime*
- *Values must be the same in every execution*

Determinacy Facts

- *Record values known at runtime to be constant*
- *Record as determinacy facts*

$$[\![e]\!]_{f \to g \to \ldots \to k} = c$$

expression

context

value

Determinacy Examples

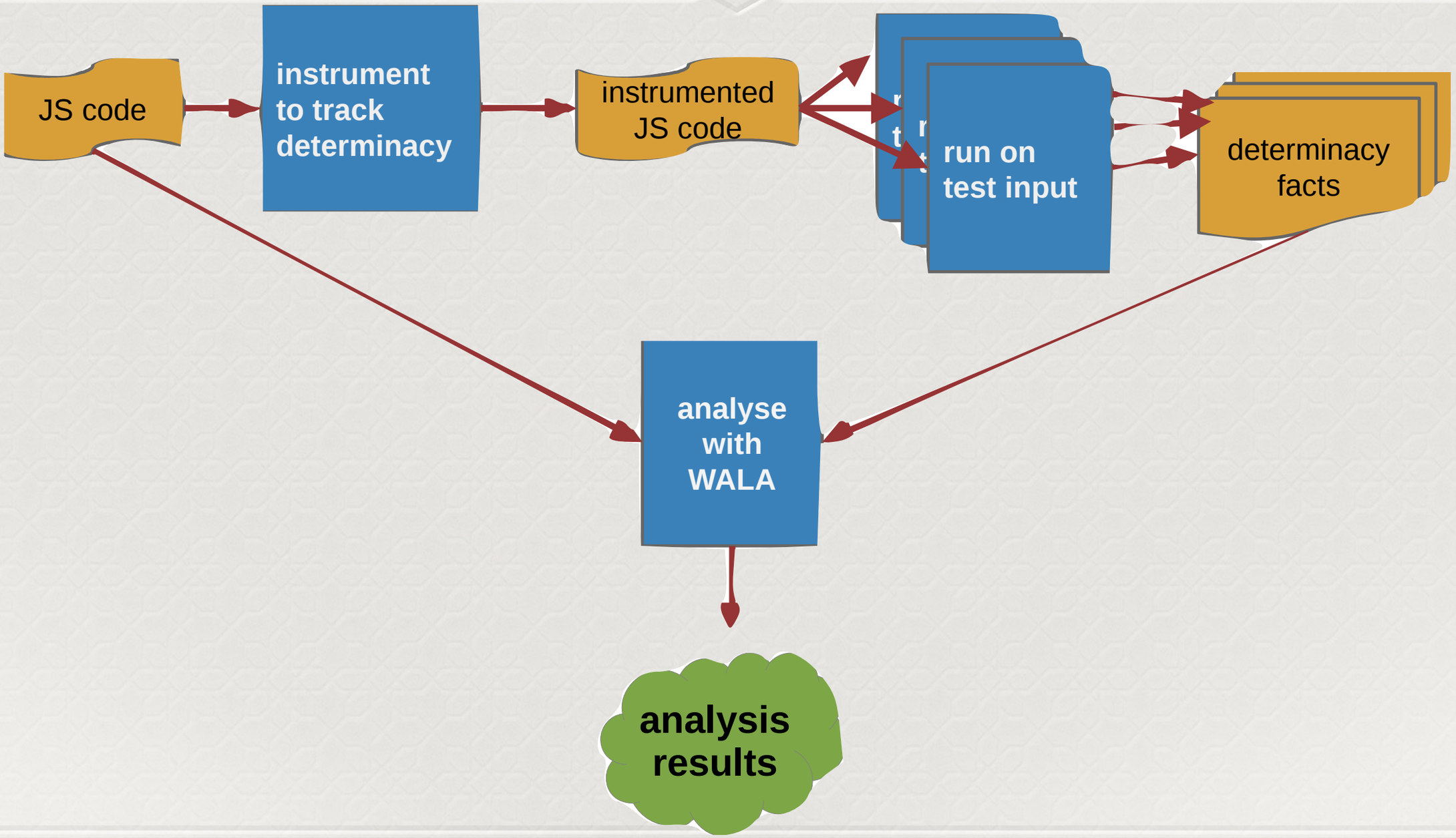
```
var e = ("blur,focus,load,resize,scroll,unload," +
 "click,dblclick,mousedown,mouseup,mousemove," +
"mouseover,mouseout,change,reset,select,submit," +
“keydown,keypress,keyup,error").split(",");

for(var i=0;i<e.length;i++) new function(){
    var o = e[i];
     …
    jQuery.fn["un"+o] = …
```

$$[\![\ \texttt{e[i]}\ ]\!]_{\perp \to 5} = \text{“}unload\text{”}$$

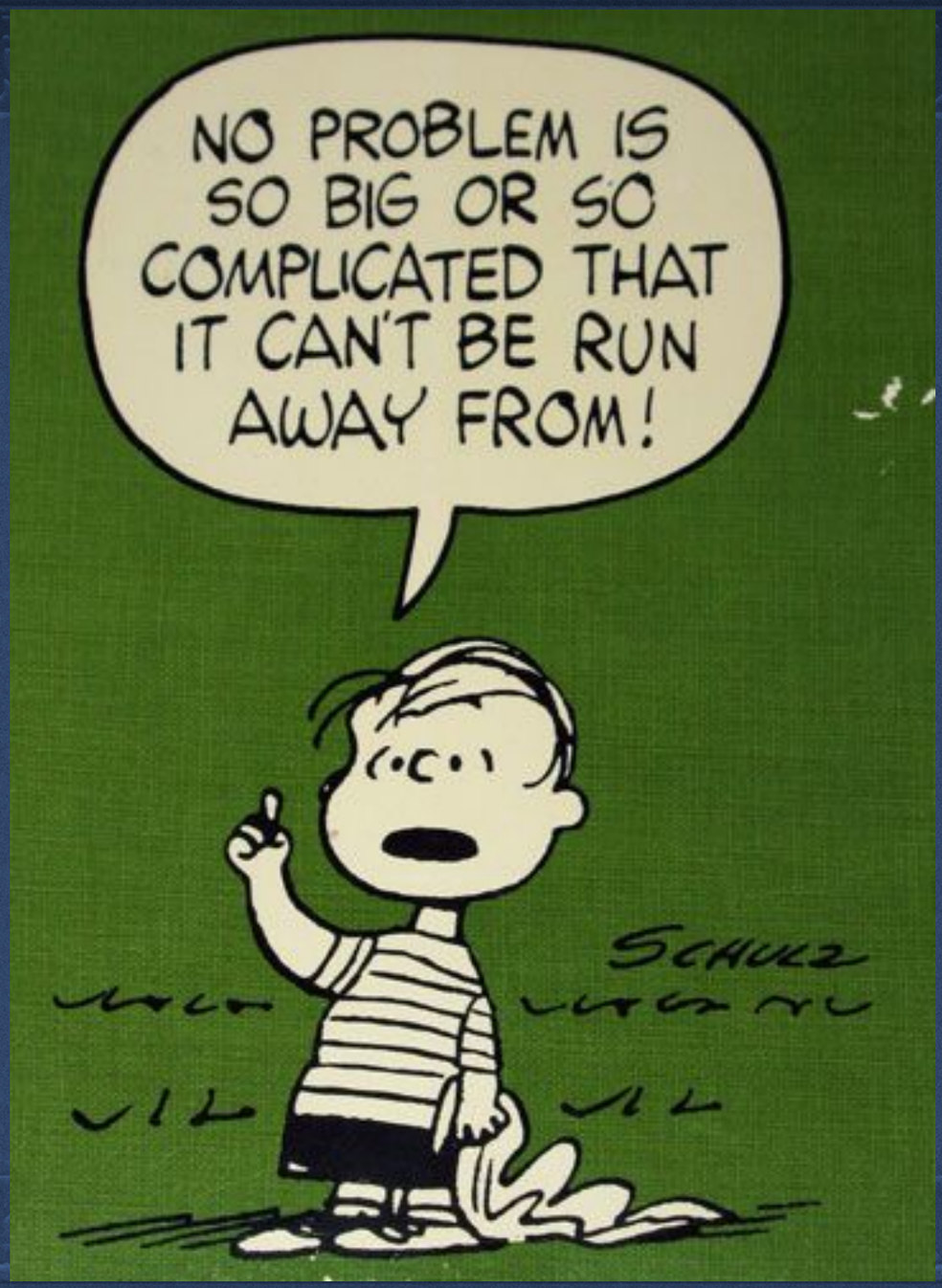
$$[\![\ \text{“}\texttt{un}\text{”} + \texttt{o}\ ]\!]_{\perp \to 5} = \text{“}ununload\text{”}$$

Implementation



Does it work?

- *Determinacy greatly aids static analysis*
- *See OOPSLA13 paper from Aarhus for a static version*
- *Allows analysis of more jQuery versions*
- *Still not sufficient for real jQuery applications*



Approximate Callgraphs

Let's not solve the impossible problems

Field-Sensitive Approach

```
jQuery.fn = {
 extend: function ext(obj) {
  for(var p in obj)
   jQuery.fn[p] = obj[p];
 }
};
```

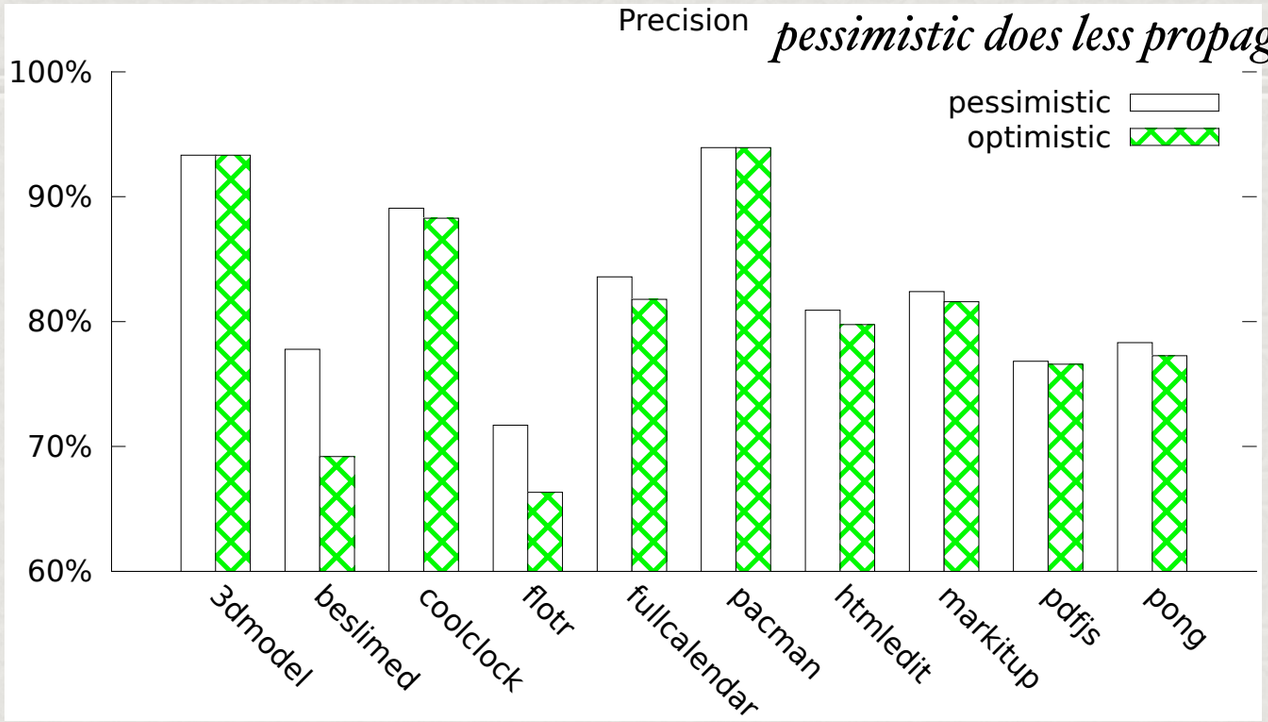
- *Field-based means one location per property*
  - *imprecisely merge properties for each name*
  - *makes copying fields a no-op to analysis*
- *No full pointer analysis*

Unsoundness

```
arr = ["Width","Height"];
for (var i=0;i<arr.length;++i)
 $.fn["outer"+arr[i]] = function() { ... };
$.fn.outerWidth();
```

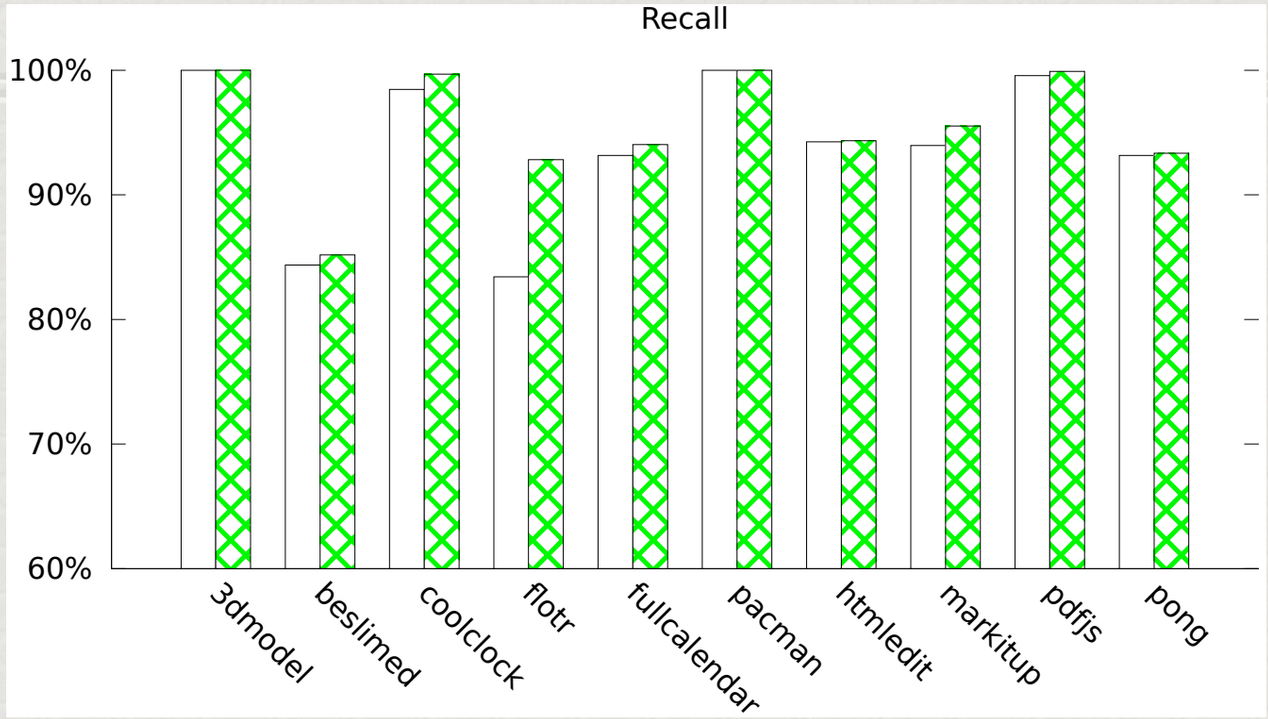
- *Ignoring dynamic accesses can be unsound*
  - *some functions only written in such accesses*
- *Field based analysis imprecise for some idioms*
  - *inheritance and overriding not distinguished*
  - *less common in JavaScript than Java*

Call Graph Precision



- *Fraction of static edges found in dynamic call graph*
- *Variable noise, like most static call graphs*
  - *>= 90% real edges in a few cases, always > 50%*
  - *Pessimistic reduces extraneous propagation*

Call Graph Recall



- *Fraction of dynamic edges found in static call graph*
- *Unsoundness limited*
  - *>= 80% of the call graph usually found*
  - *Optimistic helps callback-heavy code*

Taint Analysis Client

Adapt taint analysis to approximate call graphs

Taint Example

```
<head>
<TITLE>Welcome!</TITLE>
<script src="script.js">
</script>
</head>
<body>
<br>
Welcome to our system
<form>
  <input type="button"
         onClick="doit();"/>
  <br>
  <input type="textarea"
         id="fd"
         value="enter text"/>
</form>
</body>
</HTML>
```

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

taint rule: document.getElementById.value > window.open

Taint Example

```
<head>
<TITLE>Welcome!</TITLE>
<script src="script.js">
</script>
</head>
<body>
<br>
Welcome to our system
<form>
  <input type="button"
         onClick="doit();"/>
  <br>
  <input type="textarea"
         id="fd"
         value="enter text"/>
</form>
</body>
</HTML>
```

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

how to identify tainted source?

Taint Example

```
<head>
<TITLE>Welcome!</TITLE>
<script src="script.js">
</script>
</head>
<body>
<br>
Welcome to our system
<form>
  <input type="button"
         onClick="doit();"/>
  <br>
  <input type="textarea"
         id="fd"
         value="enter text"/>
</form>
</body>
</HTML>
```

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

don't taint all 'url' properties

Taint Example

```
<head>
<TITLE>Welcome!</TITLE>
<script src="script.js">
</script>
</head>
<body>
<br>
Welcome to our system
<form>
  <input type="button"
         onClick="doit();"/>
  <br>
  <input type="textarea"
         id="fd"
         value="enter text"/>
</form>
</body>
</HTML>
```

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

first-class property accesses

Taint Example

```
<head>
<TITLE>Welcome!</TITLE>
<script src="script.js">
</script>
</head>
<body>
<br>
Welcome to our system
<form>
  <input type="button"
         onClick="doit();"/>
  <br>
  <input type="textarea"
         id="fd"
         value="enter text"/>
</form>
</body>
</HTML>
```

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

only open after copy is bad

Taint Example

```
<head>
<TITLE>Welcome!</TITLE>
<script src="script.js">
</script>
</head>
<body>
<br>
Welcome to our system
<form>
  <input type="button"
         onClick="doit();"/>
  <br>
  <input type="textarea"
         id="fd"
         value="enter text"/>
</form>
</body>
</HTML>
```

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

only some 'url' tainted

Flow-Sensitive Propagation

- *No pointer analysis*
  - *No explicit aliasing*
    - *Rely on local queries*
- *Use access paths*
  - *Globals, e.g. document*
  - *But must handle methods*

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

Identifying Tainted Sources

```
var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };
```

variable	path
document	document

taint rule: document.getElementById.value > window.open

Identifying Tainted Sources

```
var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };
```

variable	path
document	document
elt	document.getElementById

taint rule: document.getElementById.value > window.open

Identifying Tainted Sources

```
var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };
```

variable	path
document	document
elt	document.getElementById
_	document.getElementById.value

taint rule: document.getElementById.value > window.open

Distinguishing Objects

variable	path
obj	url

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

taint rule: document.getElementById.value > window.open

Distinguishing Objects

variable	path
obj	url
x	url

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

taint rule: document.getElementById.value > window.open

First-Class Properties

variable	path
obj	url
x	url
a	url
b	*

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

taint rule: document.getElementById.value > window.open

First-Class Properties

variable	path
obj	url
x	url
a	url
b	*
obj	*
_	_

```
function copy(a, b) {
  for(t in a) {
    b[t] = a[t]; } }

var Facade = {
  open: function open(obj) {
    window.open(obj.url); },
  url: function url() {
    var obj = { };
    var elt = document.getElementById("fd");
    obj.url = elt.value;
    return obj;
} };

function doit() {
  var x = Facade.url();
  var y = { url: "http://cnn.com/" };
  Facade.open(y);
  copy(x,y);
  Facade.open(y); }
```

taint rule: document.getElementById.value > window.open

Taint Analysis

- *Flow-sensitive approach promising*
  - *tolerates imprecisions from field-based analysis*
  - *initial scalability results promising*
- *Evaluation so far against existing analyses*
  - *better scalability of underlying analysis*
  - *equally precise results*

The Impossible Dream

- *Approximate analysis seems inevitable*
  - *traditional approaches do not scale yet*
  - *dynamism of JavaScript challenging*
- *Approximate security analysis promising*
  - *shows good coverage compared to existing approaches*
  - *scalability benefits of approximate techniques*