Motivation

92.8% of all web sites use JavaScript¹

- Most important client-side language for web sites
- JavaScript programs are composed of third-party libraries (e.g. for calendars, maps, social networks)

¹According to http://w3techs.com/, status of February 2016

JavaScript Issues

- Executed code is a mix from different origins
- Code is accumulated by dynamic loading (e.g. eval, mashups)
- JavaScript has no security awareness

Problems

- Side effects may cause unexpected behavior
- Program understanding and maintenance is difficult
- Libraries may get access to sensitive data
Challenges of JavaScript Developers

- All-or-nothing choice when including code
- Some scripts must have access to the application state or are allowed to change it
- Some JavaScript fragments are ill-behaved

Key Challenges

1. Manage untrusted JavaScript Code
2. Control the use of data by included scripts
3. Reason about effects of included scripts

Possible Solutions

CONTRACTS with RUN-TIME MONITORING

- Behavioral Contracts
  Assertions, pre-/postconditions, higher-order contracts
  [Findler & Felleisen 2002] [Keil & Thiemann 2015]
- Access Permission Contracts
  Monitor and control access paths
  [Keil & Thiemann 2013]
- Security Policies
  Monitor and enforce object behavior
  [Agten et al. 2012]
- Preserve Integrity
  Membranes, Revocable References
  [Van Cutsem & Miller 2010]

Implementation of Contracts: JavaScript Proxies

[Keil & Thiemann 2015]
The Twist

A Maintenance Scenario
- A programmer adds contracts to sensitive objects (e.g., the arguments of a function)
- Program execution ends up in a mix of objects with and without contract
- The same object may appear with and without contract
- The original object may be compared with its contracted counterpart (e.g., by using ===)
- What happens?

Opaque Proxies

Proxies and Equality
```javascript
var target = { /* some object */ };
var contracted = new Proxy(target, contractHandler);
// ...
target === contracted // evaluates to false
```

Consequence
If a program uses object equality, then adding contracts may change the behavior of well-behaved programs.

Does this happen in practice?

Research Question 1
Does the contract implementation based on opaque proxies affect the meaning of realistic programs?

The Experiment
- Instrument the JavaScript engine to count and classify proxy-object comparisons
- Subject programs are taken from the Google Octane 2.0 Benchmark Suite
- Recursive object wrapper simulates a simple contract system by wrapping the arguments of a function
- Identity preserving membrane $M$ maintains aliasing: $M(t_1) \neq M(t_2) \Rightarrow t_1 \neq t_2$
Classification of Proxy-Object Comparisons

Type I: \( M_1(t_1) = t_2 \text{ or } M_1(t_1) = M_2(t_2) \)

I-a. If \( t_1 \neq t_2 \), then result should be false.
   Same result for all implementations.
I-b. If \( t_1 = t_2 \), then result should be true.
   False with JS proxies

Type II: \( M(t_1) = M(t_2) \)

II-a. If \( t_1 \neq t_2 \), then result should be false.
   Same result for all implementations.
II-b. If \( t_1 = t_2 \), then result should be true.
   May be false with JS proxies if membrane not identity preserving

Numbers of Comparisons involving Proxies

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Total</th>
<th>Type-I</th>
<th>Type-II</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeltaBlue</td>
<td>144126</td>
<td>29228</td>
<td>43398</td>
</tr>
<tr>
<td>RayTrace</td>
<td>107560</td>
<td>0</td>
<td>82299</td>
</tr>
<tr>
<td>EarleyBoyer</td>
<td>87211</td>
<td>8651</td>
<td>6303</td>
</tr>
<tr>
<td>TypeScript</td>
<td>801436</td>
<td>598984</td>
<td>151297</td>
</tr>
</tbody>
</table>

Result

Yes, it happens! A significant number of object comparisons fail when mixing opaque proxies and their target objects.

Transparent Proxies

When comparing two objects for equality, a transparent proxy is (recursively) replaced by its target object.

Suggested Use

- Implement projections, e.g., projection contracts
- Contracts become invisible
Performance

Research Question 2
Does the introduction of the transparent proxies affect the performance of non-proxy code?

The Testing Procedure
- Google Octane 2.0 Benchmark Suite
- IonMonkey turned off / baseline JIT turned off
- One run in each configuration
- Scores: Bigger is better

Scores

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Origin JIT</th>
<th>Origin Interpreter</th>
<th>Transparent JIT</th>
<th>Transparent Interpreter</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeltaBlue</td>
<td>453</td>
<td>82.5</td>
<td>466</td>
<td>79.5</td>
</tr>
<tr>
<td>RayTrace</td>
<td>462</td>
<td>182</td>
<td>462</td>
<td>174</td>
</tr>
<tr>
<td>EarleyBoyer</td>
<td>909</td>
<td>275</td>
<td>913</td>
<td>270</td>
</tr>
<tr>
<td>TypeScript</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Total Score</td>
<td>1584</td>
<td>456</td>
<td>1610</td>
<td>445</td>
</tr>
</tbody>
</table>

Answer
There is no measurable difference. The difference is within the range of measurement accuracy.

The User Level
Transparent Proxy

Just a new Proxy Constructor
```javascript
var proxy = new TransparentProxy(target, handler);
proxy === target // evaluates to true
```

Caveat
- Transparent proxies are slippery!
- Library code may want to break the transparency (e.g. for efficiency reasons)
- Hard to manipulate because they have no identity
The User Level
Identity Realms

- Consists of a constructor for transparent proxies
- Provides an equals function revealing proxies of that realm
- Provides constructors for realm-aware data structures (e.g., `Map`, `Set`, `WeakMap`, `WeakSet`)

```javascript
var realm = TransparentProxy.createRealm();
var proxy = realm.Proxy(target, handler);
proxy === target; // evaluates to true
realm.equals(proxy, target); // evaluates to false
```

In the Paper

- Discussion of different use cases of proxies with respect to the requirements on proxy transparency
- Discussion of the programmer’s expectations from an equality operator
- Discussion of alternative designs to obtain transparency
- Two different APIs for creating transparent proxies
- Draft implementation of an observer proxy that guarantees projection contracts

Conclusion

- A significant number of object comparisons fail when mixing opaque proxies and their target objects
- Implementing contract systems with opaque proxies changes the semantics of contract-abiding programs
- Transparent proxies are a viable alternative
- Neither the transparent nor the opaque implementation is appropriate for all use cases
- To preserve programmer expectations, transparent proxies should be used as observer proxies (cf. Chaperones vs. Impersonators in Racket)
Data structures depending on object equality need to handle transparent proxies.
If \( \text{obj1} == \text{obj2} \) then \( \text{map.get(obj1)} == \text{map.get(obj2)} \)

**Normal Map**
1. var realm = TransparentProxy.createRealm();
2. var tproxy1 = realm.Proxy(target, handler);
3. var tproxy2 = realm.Proxy(target, handler);
4. var map = new Map();
5. map.add(tproxy1, 1); // map : \{#target \rightarrow (tproxy1, 1)\}
6. map.add(tproxy2, 2); // map : \{#target \rightarrow (tproxy2, 2)\}

**Realm-aware Map**
1. var realm = TransparentProxy.createRealm();
2. var tproxy1 = realm.Proxy(target, handler);
3. var tproxy2 = realm.Proxy(target, handler);
4. var map = realm.Map();
5. map.add(tproxy1, 1); // map : \{#tproxy1 \rightarrow (tproxy1, 1)\}
6. map.add(tproxy2, 2); // map : \{#tproxy2 \rightarrow (tproxy2, 2)\}

**An Example**

Proxies and Equality
- Let \( x, f, g \) be some global elements:
  1. var x = \{ /\ some object \}/;
  2. var f = function (y) { return x == y }
  3. var g = function (x, s) { return f(s) }
- Let \( C, D \) be two contracts implemented by proxies:
  1. var h = g @ ([[C -> Any], D] -> Any)
- The execution ends up in:
  1. new Proxy(x, C.Handler)) == new Proxy(x, D.Handler))
- Execution ends in false instead of true!
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Origin</th>
<th>Transparent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No-Ion</td>
<td>No-JIT</td>
</tr>
<tr>
<td>Richards</td>
<td>505</td>
<td>64.8</td>
</tr>
<tr>
<td>DeltaBlue</td>
<td>453</td>
<td>82.5</td>
</tr>
<tr>
<td>Crypto</td>
<td>817</td>
<td>121</td>
</tr>
<tr>
<td>RayTrace</td>
<td>462</td>
<td>182</td>
</tr>
<tr>
<td>EarleyBoyer</td>
<td>909</td>
<td>275</td>
</tr>
<tr>
<td>RegExp</td>
<td>853</td>
<td>371</td>
</tr>
<tr>
<td>Splay</td>
<td>802</td>
<td>409</td>
</tr>
<tr>
<td>SplayLatency</td>
<td>1172</td>
<td>1336</td>
</tr>
<tr>
<td>NavierStokes</td>
<td>841</td>
<td>155</td>
</tr>
<tr>
<td>pdf.js</td>
<td>2759</td>
<td>704</td>
</tr>
<tr>
<td>Mandreel</td>
<td>691</td>
<td>82.5</td>
</tr>
<tr>
<td>MandreelLatency</td>
<td>3803</td>
<td>526</td>
</tr>
<tr>
<td>Gameboy Emulator</td>
<td>4275</td>
<td>556</td>
</tr>
<tr>
<td>Code loading</td>
<td>9063</td>
<td>943</td>
</tr>
<tr>
<td>Box2DWeb</td>
<td>1726</td>
<td>289</td>
</tr>
<tr>
<td>zlib</td>
<td>28981</td>
<td>29052</td>
</tr>
<tr>
<td>TypeScript</td>
<td>3708</td>
<td>1241</td>
</tr>
</tbody>
</table>