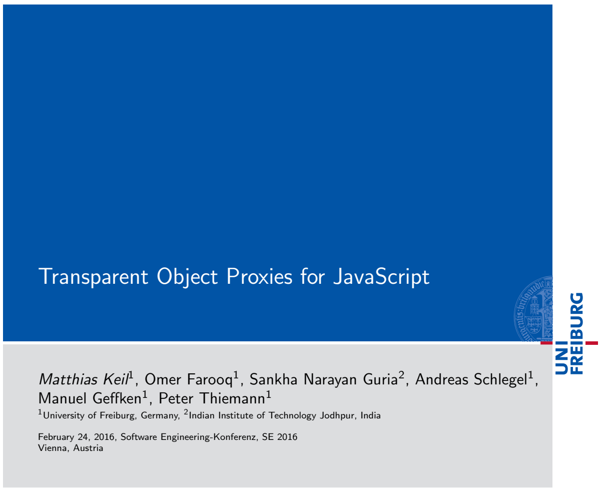


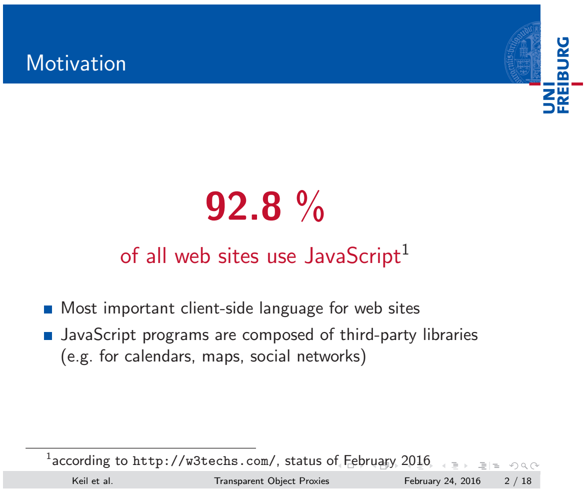
Transparent Object Proxies for JavaScript

Matthias Keil[1], Omer Farooq[1], Sankha Narayan Guria[2], Andreas Schlegel[1], Manuel Geffken[1], Peter Thiemann[1]

[1]University of Freiburg, Germany, [2]Indian Institute of Technology Jodhpur, India

February 24, 2016, Software Engineering-Konferenz, SE 2016
Vienna, Austria

Notizen



Motivation

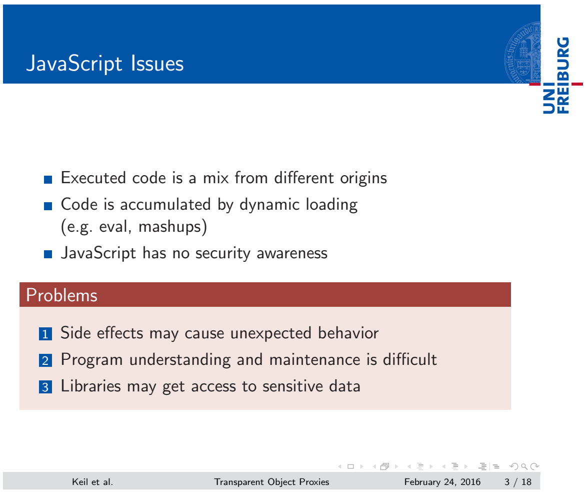
92.8 %

of all web sites use JavaScript[1]

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

[1] according to `http://w3techs.com/`, status of February 2016

Notizen



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

1. Side effects may cause unexpected behavior
2. Program understanding and maintenance is difficult
3. Libraries may get access to sensitive data

Notizen

Challanges of JavaScript Developers

- All-or-nothing choice when including code
- Some scripts must have access 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

Notizen

Possible Solutions

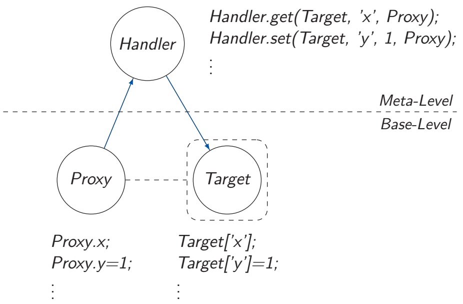
CONTRACTS with RUN-TIME MONITORING

- **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]

Notizen

Implementation of Contracts: JavaScript Proxies

[Keil & Thiemann 2015]



```
Handler.get(Target, 'x', Proxy);
Handler.set(Target, 'y', 1, Proxy);
⋮
```

```
Proxy.x;
Proxy.y=1;
⋮
```

```
Target['x'];
Target['y']=1;
⋮
```

Notizen

The Twist

A Maintenance Scenario

- A programmer adds contracts to sensitive objects (e.g. to 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?**

Notizen

Opaque Proxies

Proxies and Equality

```
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 behaviour** of well-behaved programs

Notizen

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: $M(t_1) \neq M(t_2) \Rightarrow t_1 \neq t_2$

Notizen

Classification of Proxy-Object Comparisons

Type I: $M_1(t_1) = t_2$ 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

Notizen

Numbers of Comparisons involving Proxies

		Type-I		Type-II	
Benchmark [2]	**Total**	**I-a**	**I-b**	**II-a**	**II-b**
DeltaBlue	144126	29228	1411	33789	79698
RayTrace	1075606	0	0	722703	352903
EarleyBoyer	87211	8651	6303	53389	18868
TypeScript	801436	599894	151297	20500	29745

Result

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

[2] The remaining benchmarks don't do any proxy-object comparisons.

Notizen

Transparent Proxies

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

Notizen

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**

Notizen

Scores

	Origin		**Transparent**	
Benchmark	**JIT**	**Interpreter**	**JIT**	**Interpreter**
DeltaBlue	453	82.5	466	79.6
RayTrace	462	182	462	174
EarleyBoyer	909	275	913	270
⋮	⋮	⋮	⋮	⋮
TypeScript	3708	1241	3666	1203
Total Score	**1594**	**456**	**1610**	**445**

Answer

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

Notizen

The User Level
Transparent Proxy

Just a new Proxy Constructor

```
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

Notizen

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*)

Identity Realm

```
var realm = TransparentProxy.createRealm();
var proxy = realm.Proxy (target, handler);
```

```
proxy === target; // evaluates to true
realm.equals(proxy, target); // evaluates to false
```

Notizen

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 guarantee projection contracts

Notizen

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)

Notizen

The User Level

Maps, Sets, and other Data Structures

- Data structures depending on object equality needs to handle transparent proxies
- If *obj1==obj2* then *map.get(obj1)==map.get(obj2)*

Normal Map

```
var realm = TransparentProxy.createRealm();
var tproxy1 = realm.Proxy (target, handler);
var tproxy2 = realm.Proxy (target, handler);

var map = new Map();
map.add(tproxy1, 1); // map : [#target −> (tproxy1, 1)]
map.add(tproxy2, 2); // map : [#target −> (tproxy2, 2)]
```

Notizen

The User Level

Maps, Sets, and other Data Structures

Realm-aware Map

```
var realm = TransparentProxy.createRealm();
var tproxy1 = realm.Proxy (target, handler);
var tproxy2 = realm.Proxy (target, handler);

var map = realm.Map();
map.add(tproxy1, 1); // map : [#tproxy1 −> (tproxy1, 1)]
map.add(tproxy2, 2); // map : [..., #tproxy2 −> (tproxy2, 2)]
```

Notizen

An Example

Proxies and Equality

- Let *x*, *f*, *g* be some global elements:

```
var x = { /∗ some object ∗/ };
var f = function (y) { return x===y }
var g = function (f, x) { return f(x) }
```

- Let *C*, *D* be two contracts implemented by proxies:

```
var h = g @ ([(C −> Any), D] −> Any)
```

- The execution ends up in:

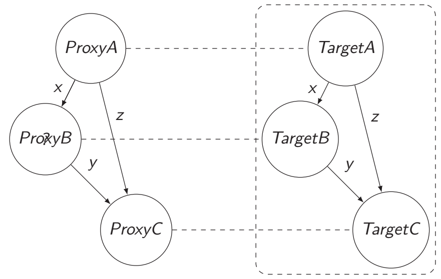
```
new Proxy(x, C_Handler)) === new Proxy(x, D_Handler))
```

- **Execution ends up in <u>false</u> instead of <u>true</u>!**

Notizen

Identity Preserving Membrane

[Van Cutsem & Miller 2010]



Notizen

Benchmark	Origin No-Ion	Origin No-JIT	Transparent No-Ion	Transparent No-JIT
Richards	505	64.8	509	64.3
DeltaBlue	453	82.5	466	79.6
Crypto	817	111	793	109
RayTrace	462	182	462	174
EarleyBoyer	909	275	913	270
RegExp	853	371	871	365
Splay	802	409	857	409
SplayLatency	1172	1336	1231	1338
NavierStokes	841	155	834	148
pdf.js	2759	704	2793	691
Mandreel	691	82.5	688	78.5
MandreelLatency	3803	526	3829	503
Gameboy Emulator	4275	556	4382	540
Code loading	9063	9439	9114	9502
Box2DWeb	1726	289	1736	282
zlib	28981	29052	28909	29108
TypeScript	3708	1241	3666	1203

Notizen

Notizen