Efficient Dynamic Access Analysis Using JavaScript Proxies
DLS’13

Matthias Keil, Peter Thiemann
Institute for Computer Science
University of Freiburg
Freiburg, Germany

October 28, 2013, Indianapolis, Indiana, USA.
92 %

of all web sites use JavaScript
Motivation

92% of all web sites use JavaScript

- Most important client-side language for web sites
- Web-developers rely on third-party libraries
  - e.g. for calendars, maps, social networks
Situation of a Web-developer
Situation of a Web-developer

Matthias Keil, Peter Thiemann

Dynamic Access Analysis

October 28, 2013
Situation of a Web-developer

Matthias Keil, Peter Thiemann

Dynamic Access Analysis

October 28, 2013
Italienische Regierungskrise

Berlusconi's Abgeordnete wollen Letta Vertrauen aussprechen

In der italienischen Regierungskrise gibt es eine überraschende Wende: Senatoren und Abgeordnete der Partei von Silvio Berlusconi wollen Premier Enrico Letta nun doch das Vertrauen aussprechen - damit könnte die Regierung weiterarbeiten. mehr... [Forum]

Staatskrise in Italien: Der Geiselnehmer

Haushaltsstreit

Obama bittet Shutdown-Opfer um Geduld

US-Präsident Obama hat sich wenige Stunden nach Beginn des Haushaltsstreiks an die betroffenen Staatsbediensteten gewandt. Er werde sich dafür einsetzen, dass die Misere bald beendet sei - doch das kann dauern. mehr... [Video | Forum]

Republikaner im US-Haushaltsdrama: Die Kamikaze-Partei

Shutdown: Börsen sitzen auf schnelle Lösung im US-Haushaltsstreit

Bundeshaushalt: Warum in Deutschland ein Shutdown unmöglich ist

#Shutdown auf Twitter: "Die Bücherei ist zu. Darf ich stattdessen dich auschecken?"
Situation of a Web-developer

Matthias Keil, Peter Thiemann
Dynamic Access Analysis
October 28, 2013
JavaScript issues

- Dynamic programming language
  - Code is accumulated by dynamic loading
  - e.g. eval, mashups
JavaScript issues

- Dynamic programming language
  - Code is accumulated by dynamic loading
  - e.g. eval, mashups

- JavaScript has no security awareness
  - No namespace or encapsulation management
  - Global scope for variables/ functions
  - All scripts have the same authority
JavaScript issues

- Dynamic programming language
  - Code is accumulated by dynamic loading
  - e.g. eval, mashups
- JavaScript has no security awareness
  - No namespace or encapsulation management
  - Global scope for variables/functions
  - All scripts have the same authority

Problems

1. Side effects may cause unexpected behavior
2. Program understanding and maintenance is difficult
3. Libraries may get access to sensitive data
   - User code may be prone to injection attacks
JSConTest, a tool for effect monitoring and inference
JSConTest
Type and effect contracts with run-time checking

- JSConTest, a tool for effect monitoring and inference

Type and effect contracts

- Type contracts
  
  ```
  1 function (x, y) /*c (int, int) \rightarrow bool */ { ... }
  ```
JSCoTnTest
Type and effect contracts with run-time checking

- JSCoTnTest, a tool for effect monitoring and inference

**Type and effect contracts**

- **Type contracts**
  
  ```
  function (x, y) /*c (int, int) -> bool */ { ... }
  ```

- **Effect contracts specifying access paths**
  
  ```
  js: tree() -> int with [this./left | right/*.bal]
  ```
JSConTest
Type and effect contracts with run-time checking

- JSConTest, a tool for effect monitoring and inference

### Type and effect contracts

- **Type contracts**
  ```
  function (x, y) /*c (int, int) -> bool */ { ... }
  ```

- **Effect contracts specifying access paths**
  ```
  js:tree(). -> int with [this./left | right /*.bal]
  ```

- Investigate effects of unfamiliar function
  - Monitoring its execution
  - Summarizing the observed traces to compact descriptions
Shortcomings of JSConTest

- Implemented by an offline code transformation
  - Partial interposition (dynamic code, `eval`, . . .)
  - Tied to a particular version of JavaScript
  - Transformation hard to maintain
Shortcomings of JSConTest

- Implemented by an offline code transformation
  - Partial interposition (dynamic code, eval, ...)
  - Tied to a particular version of JavaScript
  - Transformation hard to maintain

- Special contract syntax
  - Requires a special JavaScript parser
Shortcomings of JSConTest

- Implemented by an offline code transformation
  - Partial interposition (dynamic code, eval, ...)
  - Tied to a particular version of JavaScript
  - Transformation hard to maintain
- Special contract syntax
  - Requires a special JavaScript parser
- Efficiency issues
  - Naive representation of access paths
  - Wastes memory and impedes scalability
Redesign and reimplemention of JSConTest based on JavaScript proxies
Redesign and reimplementation of JSConTest based on JavaScript proxies

Advantages

- Full interposition for the full language
  - Including dynamically loaded code and `eval`
- Safe for future language extensions
  - No transformation to maintain
- Runs faster in less memory
  - Efficient representation of access paths
  - Incremental path matching
- Maintenance is simplified
  - No custom syntax for contracts
Effects for JavaScript

Only some parts of an object are accessible:

```javascript
var proxy = APC.permit('(a.?+b*)', {a:{b:5}, b:{b:11}});
a = proxy.a; // APC.permit('?', {b:5});
a.b = 3;
```

- APC encapsulates JSConTest2
- `permit` wraps an object with a permission. Arguments:
  1. Permission encoded in a string
  2. Object that is protected by the permission
- Contract specifies permitted access paths
  - Last property is readable/writeable
  - Prefix is read-only
  - Not addressed properties are neither readable nor writeable
  - Read-only paths possible (@ denotes a non-existing property)
Contracts on Functions

```javascript
var func = APC.permitArgs('arguments.0.(a.?+b*)',
  function(arg0) {
    // do something
  });
```

- `permitArg` wraps a function with permissions
  - 1 contract applied to function arguments
  - 2 function

- Arguments accessed by position `arguments.0`
  - No reliable way to access parameter names
  - Function may use unlisted parameters
  - Parameter names may not be unique
Interaction of Contracts

```javascript
var x = APC.permit('((a+a.b)+b.b.@)', {a:{b:3}, b:{b:5}});
x.a = x.b; // APC.permit('b.@', {b:5});
y = x.a; // APC.permit('b & b.@', {b:5});
y.b = 7; // violation
```

- Line 2 reads `x.b` and writes `x.a`
- Afterwards, `x.b` and `x.a` are aliases
- JSConTest2 enforces both contracts reaching `x.b` and `x.a`
- `x.a` carries contract `'(ε+b)&b.@' = 'b.@'`
- Thus, writing to `x.a.b` is not permitted
Syntax of Access Permission Contracts

\[ \text{Literal} \quad \exists \ l \ ::= \ \emptyset \mid \emptyset \mid r \]

\[ \text{Contract} \quad \exists \ C \ ::= \ \varepsilon \mid l \mid C^* \mid C+C \mid C\&C \mid C.C \]

- Each literal \( l \) defines a property access
Syntax of Access Permission Contracts

\[ \text{Literal} \ni \ell ::= \emptyset | \emptyset | r \]
\[ \text{Contract} \ni C ::= \epsilon | \ell | C^* | C+C | C&C | C.C \]

- Each literal \( \ell \) defines a property access
- Access contracts are regular expressions on literals
  - \( \mathcal{L}[C] \) denotes the language of \( C \), that defines a set of permitted access paths
The JSConTest2 Approach

- Full interposition of contracted objects
  - Proxy intercepts all operations
  - Proxy-handler contains contract \( C \) and path set \( \mathcal{P} \)
  - Forwards the operation or signals a violation

- Returned object contains the remaining contract (*Membrane*)
JavaScript Proxies

Meta-Level

Base-Level

Handler

Proxy `p.foo` Target
JavaScript Proxies

Meta-Level

Base-Level

Handler

h.get(t, 'foo', p)

Proxy

p.foo

Target
Meta-Level

Base-Level

JavaScript Proxies

Handler

```
Proxy.p.foo
```

Target

```
t[foo]
```

```
let h = { get: t => Target[t'] };
const p = new Proxy(t, h);
```

```
h.get(t, 'foo', p)
```
JavaScript Proxies

Meta-Level

Base-Level

Handler

h.set(t, 'bar', 4711, p)

Proxy

p.bar=4711

Target

t['bar']=4711

Matthias Keil, Peter Thiemann
Access Path: \( P \)
Contract: \( C \)
Access Path: $\mathcal{P}$
Contract: $\mathcal{C}$
Access Path: \( \mathcal{P} \)
Contract: \( C \)
Access Path: $\mathcal{P}$
Contract: $\mathcal{C}$

Access Path: $\mathcal{P}.p$
Contract: $\partial_p(\mathcal{C})$
\[ \partial_p(C) \] is the Brzozowski derivative of \( C \) with respect to \( p \)

\[ \partial_p(C) \] accepts the quotient language:

\[ p^{-1}L[C] = \{w \mid pw \in L[C]\} \]
Membrane issues

What if a contract is applied to a proxy?
What if a contract is applied to a proxy?

1. The proxy is wrapped in another proxy
   - Tradeoff: Inefficient due to chains of proxies
Membrane issues

What if a contract is applied to a proxy?

1. The proxy is wrapped in another proxy
   - Tradeoff: Inefficient due to chains of proxies

2. The existing proxy is reused with updated information
   - Requires merge operations for contracts and paths
     - Intersection of contracts
     - Union of path sets
Native representations of path sets waste space
Path update becomes inefficient

*Solution:* Store paths in a trie structure
Access Paths

- Native representations of path sets waste space
- Path update becomes inefficient
- **Solution**: Store paths in a trie structure

Access Permission Contracts

- Contracts get large and may contain redundant parts
- Computing derivative becomes more expensive
- **Solution**: Contract rewriting
Suppose that $\mathcal{L}[\mathcal{C}] \subseteq \mathcal{L}[\mathcal{C}']$. Then simplify

- $\mathcal{C} + \mathcal{C}'$ to $\mathcal{C}'$
- $\mathcal{C} \& \mathcal{C}'$ to $\mathcal{C}$

**Definition (Containment)**

A contract $\mathcal{C}$ is contained in another contract $\mathcal{C}'$, written as $\mathcal{C} \sqsubseteq \mathcal{C}'$, iff $\mathcal{L}[\mathcal{C}] \subseteq \mathcal{L}[\mathcal{C}']$. 
Suppose that $\mathcal{L}[\mathcal{C}] \subseteq \mathcal{L}[\mathcal{C}']$. Then simplify

- $C + C'$ to $C'$
- $C \& C'$ to $C$

**Definition (Containment)**

A contract $C$ is contained in another contract $C'$, written as $C \sqsubseteq C'$, iff $\mathcal{L}[\mathcal{C}] \subseteq \mathcal{L}[\mathcal{C}']$.

**Requirement**

- Decide $C \sqsubseteq C'$ quickly
- Use Antimirov’s technique, based on derivatives
Lemma (Containment)

\[ C \subseteq C' \iff \nu(\partial_\mathcal{P}(C')) \text{ for all } \mathcal{P} \in \mathcal{L}[C] \]
Lemma (Containment)

\[ C \subseteq C' \iff \nu(\partial_P(C')) \text{ for all } P \in \mathcal{L}[C] \] (1)

Lemma (Containment2)

\[ C \subseteq C' \iff \partial_p(C) \subseteq \partial_p(C') \land (\nu(C) \Rightarrow \nu(C')) \text{ for all } p \in \{ p \mid pw \in \mathcal{L}[C] \} \] (2)
Lemma (Containment)

\[ C \sqsubseteq C' \iff \nu(\partial P(C')) \text{ for all } P \in L[C] \]  

Lemma (Containment2)

\[ C \sqsubseteq C' \iff \partial_p(C) \sqsubseteq \partial_p(C') \land (\nu(C) \Rightarrow \nu(C')) \text{ for all } p \in \{ p \mid pw \in L[C] \} \]  

Drawback

Literal \( r \) leads to an infinite alphabet

- Requires infinitely many test
Literal-based derivative

Definition (First Contract Literals)

\[
\begin{align*}
\text{first}(\ell) &= \{\ell\} \\
\text{first}(\epsilon) &= \emptyset \\
\text{first}(C\ast) &= \text{first}(C) \\
\text{first}(C+C') &= \text{first}(C) \cup \text{first}(C') \\
\text{first}(C\&C') &= \{\ell \cap_r \ell' \mid \ell \in \text{first}(C), \ell' \in \text{first}(C')\} \\
\text{first}(C.C') &= \begin{cases} 
\text{first}(C) \cup \text{first}(C'), & \nu(C) \\
\text{first}(C), & \text{otherwise}
\end{cases}
\end{align*}
\]
Definition (First Contract Literals)

\[
\begin{align*}
\text{first}(\ell) & = \{\ell\} \\
\text{first}(\epsilon) & = \{\} \\
\text{first}(C^\ast) & = \text{first}(C) \\
\text{first}(C + C') & = \text{first}(C) \cup \text{first}(C') \\
\text{first}(C \& C') & = \{\ell \cap_r \ell' \mid \ell \in \text{first}(C), \ell' \in \text{first}(C')\} \\
\text{first}(C.C') & = \begin{cases} 
\text{first}(C) \cup \text{first}(C'), & \nu(C) \\
\text{first}(C), & \text{otherwise}
\end{cases}
\end{align*}
\]

It holds that:

\[
\{p \mid pw \in \mathcal{L}[C]\} = \mathcal{L}[\text{first}(C)] \tag{3}
\]
\( \nabla_\ell(C) \) is the literal-based derivative of \( C \) with respect to \( \ell \)

**Lemma (Syntactic derivative of contracts)**

\[
\mathcal{L} [\nabla_\ell(C)] = \bigcap_{\rho \in \mathcal{L}[\ell]} \mathcal{L} [\partial_\rho(C)]
\]  

(4)
\( \nabla_\ell(C) \) is the literal-based derivative of \( C \) with respect to \( \ell \)

**Lemma (Syntactic derivative of contracts)**

\[
\mathcal{L}[\nabla_\ell(C)] = \bigcap_{p \in \mathcal{L}[\ell]} \mathcal{L}[\partial_p(C)]
\]  \hspace{1cm} (4)

**Theorem (Containment)**

\[
C \sqsubseteq C' \iff \nabla_\ell(C) \sqsubseteq \nabla_\ell(C') \land (\nu(C) \Rightarrow \nu(C'))
\]  \hspace{1cm} for all \( \ell \in \text{first}(C) \)  \hspace{1cm} (5)
Implementation

- Implementation based on the JavaScript Proxy API
- Implemented since Firefox 18.0 and Chrome 3.5
Implementation based on the JavaScript Proxy API
- Implemented since Firefox 18.0 and Chrome 3.5
- Implementation provides an proxy-handler
Implementation based on the JavaScript Proxy API
- Implemented since Firefox 18.0 and Chrome 3.5
- Implementation provides an proxy-handler
- Two evaluation modes:
  1. *Observer Mode*: Only path and violation logging
  2. *Protector Mode*: Omits forbidden read and write access
Implementation

- Implementation based on the JavaScript Proxy API
  - Implemented since Firefox 18.0 and Chrome 3.5
- Implementation provides an proxy-handler
- Two evaluation modes:
  1. **Observer Mode**: Only path and violation logging
  2. **Protector Mode**: Omits forbidden read and write access

Limitations

1. **Cannot directly protect DOM objects**
   - Because of the browser’s sandbox
Implementation based on the JavaScript Proxy API
- Implemented since Firefox 18.0 and Chrome 3.5
- Implementation provides an proxy-handler
- Two evaluation modes:
  1. Observer Mode: Only path and violation logging
  2. Protector Mode: Omits forbidden read and write access

Limitations

1. Cannot directly protect DOM objects
   - Because of the browser’s sandbox
2. Proxies are not transparent with respect to equality
   - For distinct proxies == and === returns false, even if the target object is the same
Benchmark Programs

- Google V8 Benchmark Suite
- Benchmarks accompanying the TAJS system
- Libraries like jQuery
- Dumped web pages like youtube or twitter
Benchmark Programs

- Google V8 Benchmark Suite
- Benchmarks accompanying the TAJS system
- Libraries like jQuery
- Dumped web pages like youtube or twitter

Applied access contract inference by logging with universal contract?
Benchmark Programs
- Google V8 Benchmark Suite
- Benchmarks accompanying the TAJS system
- Libraries like jQuery
- Dumped web pages like youtube or twitter

Applied access contract inference by logging with universal contract?

Prepared customized contracts to protect objects
Evaluation

- Benchmark Programs
  - Google V8 Benchmark Suite
  - Benchmarks accompanying the TAJS system
  - Libraries like jQuery
  - Dumped web pages like youtube or twitter

- Applied access contract inference by logging with universal contract ?*

- Prepared customized contracts to protect objects

Initial implementation: quickly ran out of memory
Evaluation

- Benchmark Programs
  - Google V8 Benchmark Suite
  - Benchmarks accompanying the TAJs system
  - Libraries like jQuery
  - Dumped web pages like youtube or twitter

- Applied access contract inference by logging with universal contract ?
- Prepared customized contracts to protect objects

*Initial implementation*: quickly ran out of memory

*Final implementation*: acceptable performance

- Using trie structures and contract rewriting
### Google V8 Benchmark Suite

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Baseline</th>
<th>Contracts only</th>
<th>Without logging</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegExp</td>
<td>2.4sec</td>
<td>2.4sec</td>
<td>2.4sec</td>
<td>2.4sec</td>
</tr>
<tr>
<td>NavierStokes</td>
<td>2.3sec</td>
<td>2.3sec</td>
<td>2.3sec</td>
<td>2.3sec</td>
</tr>
<tr>
<td>EarleyBoyer</td>
<td>4.3sec</td>
<td>4.4sec</td>
<td>4.4sec</td>
<td>4.4sec</td>
</tr>
<tr>
<td>DeltaBlue</td>
<td>2.3sec</td>
<td>3.3sec</td>
<td>9.5sec</td>
<td>9.8sec</td>
</tr>
<tr>
<td>Richards</td>
<td>2.3sec</td>
<td>3.3sec</td>
<td>18.6min</td>
<td>22.5min</td>
</tr>
<tr>
<td>RayTrace</td>
<td>2.3sec</td>
<td>1.6min</td>
<td>1.1h</td>
<td>1.2h</td>
</tr>
<tr>
<td>Crypto</td>
<td>4.4sec</td>
<td>2.6min</td>
<td>2.5h</td>
<td>4.2h</td>
</tr>
<tr>
<td>Splay</td>
<td>2.3sec</td>
<td>2.3sec</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Most time consuming parts are *Path Generation* and *Contract Derivation*
Conclusion

- Effect logging and dynamic enforcement of access contracts with proxies
- Shortcomings of previous, translation-based implementation avoided
  - Support for the full JavaScript language
  - Guarantees full interposition
Conclusion

- Effect logging and dynamic enforcement of access contracts with proxies
- Shortcomings of previous, translation-based implementation avoided
  - Support for the full JavaScript language
  - Guarantees full interposition
- Contract rewriting extending results by results from Brzozowski and Antimirov to reduce memory consumption
Effect logging and dynamic enforcement of access contracts with proxies

Shortcomings of previous, translation-based implementation avoided
  - Support for the full JavaScript language
  - Guarantees full interposition

Contract rewriting extending results by results from Brzozowski and Antimirov to reduce memory consumption

Practical applicability of access permission contracts
  - Runtime overhead of pure contract enforcement is negligible

Full effect logging incurs some overhead
  - Primarily used for program understanding and debugging
Questions?

Thank you for your attention.