Efficient Dynamic Access Analysis Using JavaScript Proxies

DLS'13

UNI FREIBURG

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

October 28, 2013, Indianapolis, Indiana, USA.

Motivation

UNI FREIBURG

92 %

of all web sites use JavaScript

Motivation

UNI FREIBURG

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

Home | Video | Themen | Forum | English | DER SPIEGEL | SPIEGEL TV | Abo | Shop Schlagzeilen | Wetter | TV-Programm | mehr ▼

Login | Registrierung

SPIEGEL ONLINE

Politik | Wirtschaft | Panorama | Sport | Kultur | Netzwelt | Wissenschaft | Gesundheit | einestages | Karriere | Uni | Schule | Reise | Auto

1. Oktober 2013 Top-Themen: US-Haushaltsstreit | Deutschland nach der Bundestagswahl | Champions League Folgen:

SPIEGEL ONLINE zur Startseite machen Hinweis nicht mehr anzeigen

Italiens Regierungskrise

Berlusconis 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

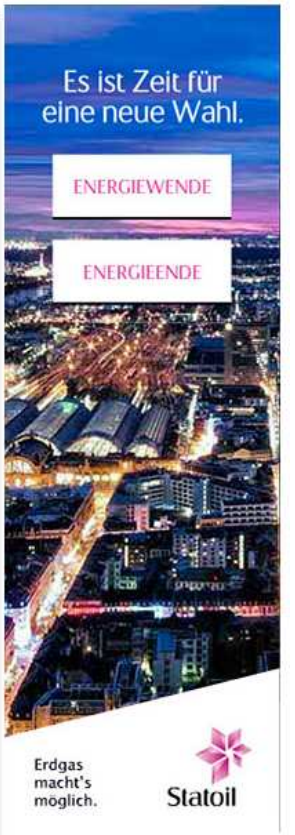
Obama bittet Shutdown-Opfer um Geduld

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

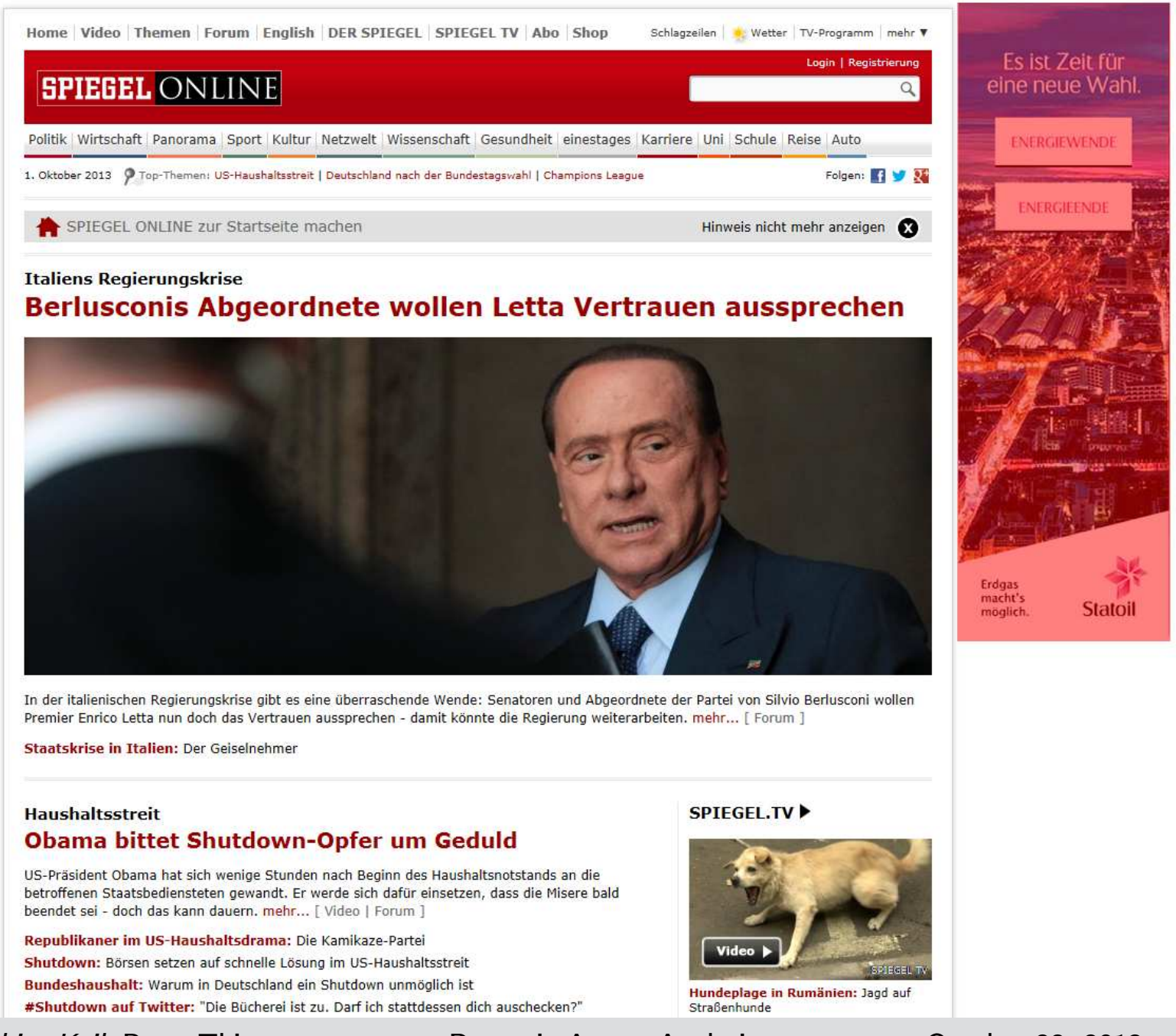
Republikaner im US-Haushaltsdrama: Die Kamikaze-Partei
Shutdown: Börsen setzen auf schnelle Lösung im US-Haushaltsstreit
Bundeshaushalt: Warum in Deutschland ein Shutdown unmöglich ist
#Shutdown auf Twitter: "Die Bücherei ist zu. Darf ich stattdessen dich auschecken?"

SPIEGEL.TV ▶

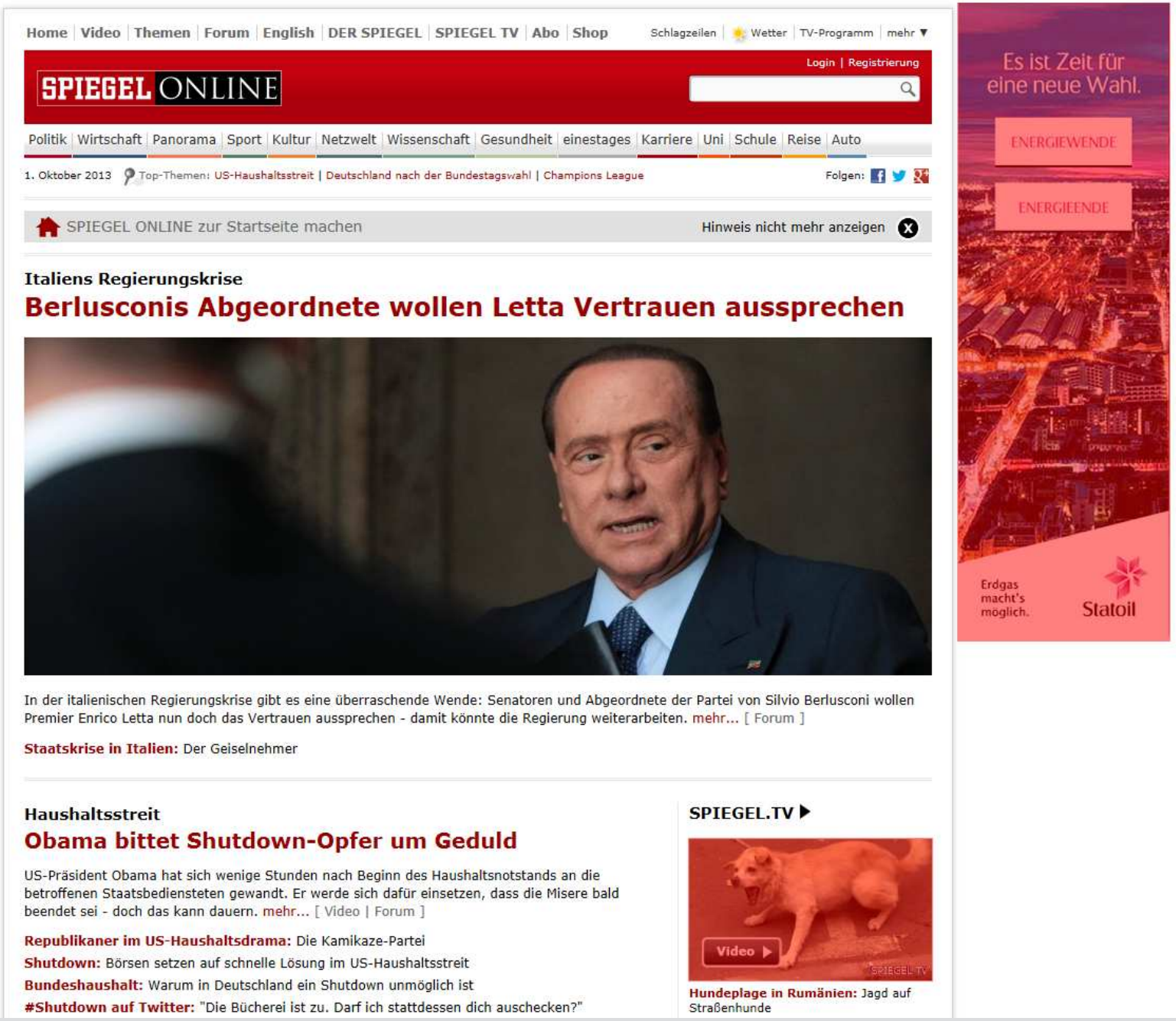
Hundeplage in Rumänien: Jagd auf Straßenhunde



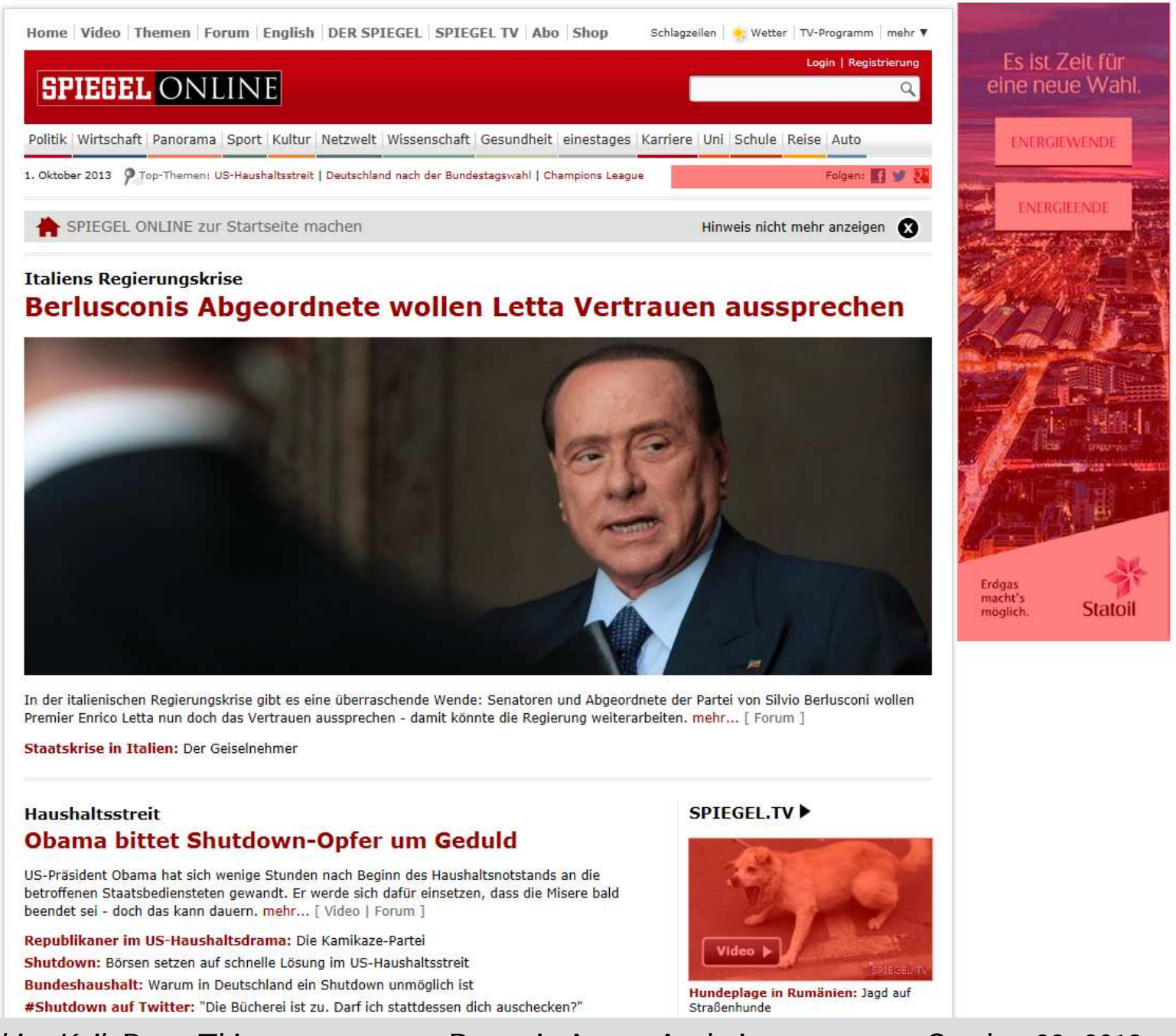
Situation of a Web-developer



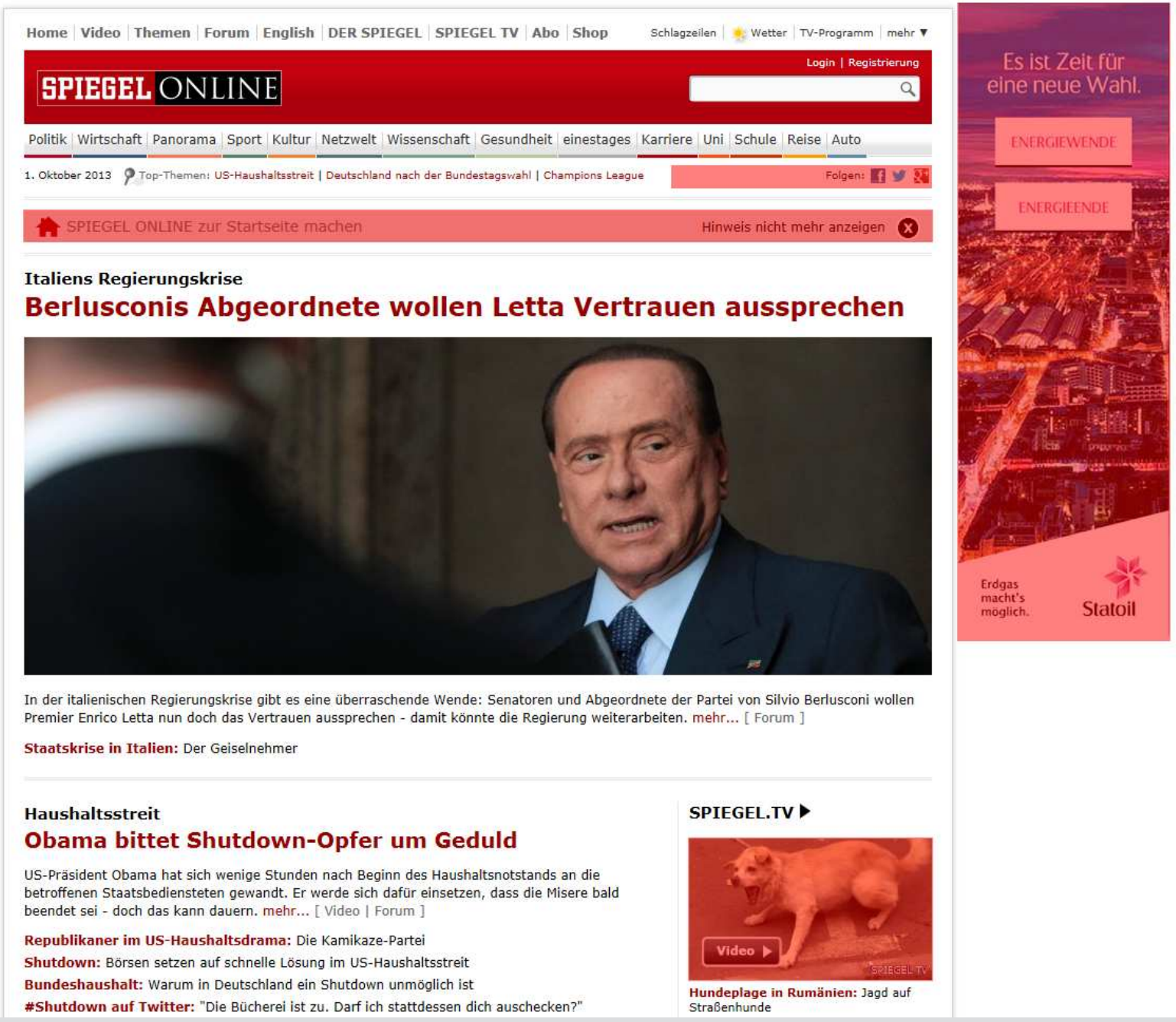
Situation of a Web-developer



Situation of a Web-developer



Situation of a Web-developer



JavaScript issues

UNI FREIBURG

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

JavaScript issues

UNI FREIBURG

- 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
Type and effect contracts with run-time checking

UNI FREIBURG

- JSConTest, a tool for effect monitoring and inference

UNI FREIBURG

- JSConTest, a tool for effect monitoring and inference

Type and effect contracts

- Type contracts

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

UNI FREIBURG

- JSConTest, a tool for effect monitoring and inference

Type and effect contracts

- Type contracts

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

- Effect contracts specifying access paths

```
1 js:tree.() -> int with [this./left|right/*.bal]
```

- JSConTest, a tool for effect monitoring and inference

Type and effect contracts

- Type contracts

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

- Effect contracts specifying access paths

```
1 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

UNI FREIBURG

- 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

UNI FREIBURG

- 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

JSConTest2

UNI FREIBURG

Redesign and reimplementation of JSConTest based on JavaScript proxies

JSConTest2

UNI FREIBURG

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

UNI FREIBURG

Only some parts of an object are accessible:

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

```
1  var func = APC.permitArgs('arguments.0.(a.?+b*)',
2  function(arg0) {
3    // do something
4  });
```

- 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

```
1   var x = APC.permit('((a+a.b)+b.b.@)', {a:{b:3}, b:{b:5}});
2   x.a = x.b; // APC.permit('b.@', {b:5});
3   y = x.a; // APC.permit('b & b.@', {b:5});
4   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

$$
\begin{array}{lrcl}
\textit{Literal} & \ni \ell & ::= & \emptyset \mid \texttt{@} \mid r \\
\textit{Contract} & \ni \mathcal{C} & ::= & \epsilon \mid \ell \mid \mathcal{C}* \mid \mathcal{C}{+}\mathcal{C} \mid \mathcal{C}\&\mathcal{C} \mid \mathcal{C}.\mathcal{C}
\end{array}
$$

- Each literal ℓ defines a property access

Syntax of Access Permission Contracts

$$
\begin{array}{lrcl}
\textit{Literal} & \ni \ell & ::= & \emptyset \mid @ \mid r \\
\textit{Contract} & \ni \mathcal{C} & ::= & \epsilon \mid \ell \mid \mathcal{C}* \mid \mathcal{C}{+}\mathcal{C} \mid \mathcal{C}\&\mathcal{C} \mid \mathcal{C}.\mathcal{C}
\end{array}
$$

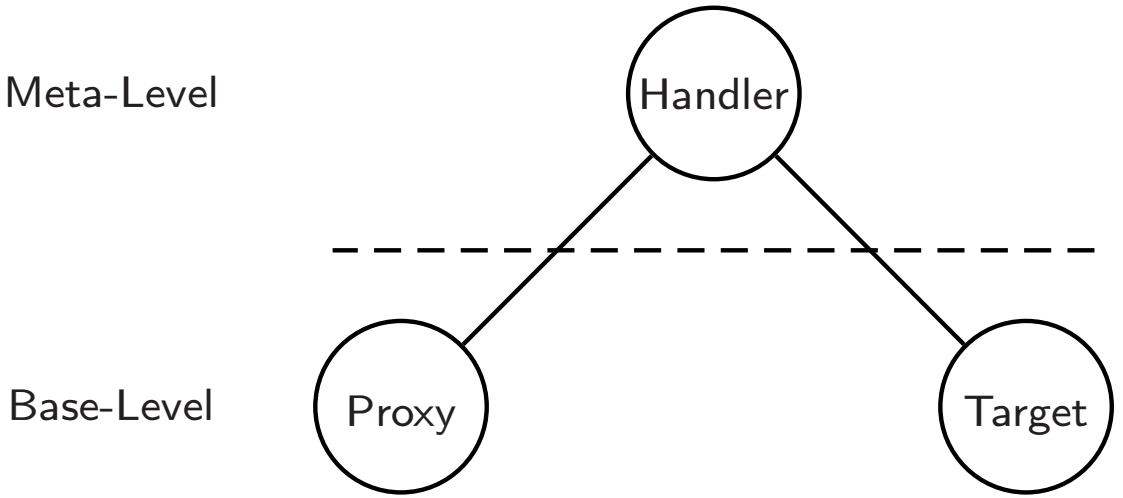
- Each literal ℓ defines a property access
- Access contracts are regular expressions on literals
  - $\mathcal{L}[\![\mathcal{C}]\!]$ denotes the language of $\mathcal{C}$, that defines a set of permitted access paths

The JSConTest2 Approach

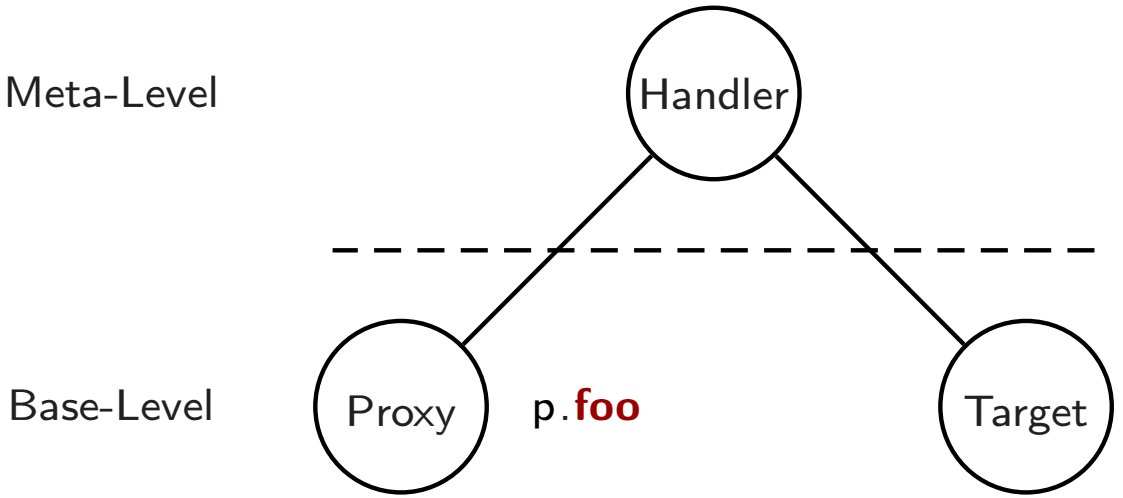
UNI FREIBURG

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

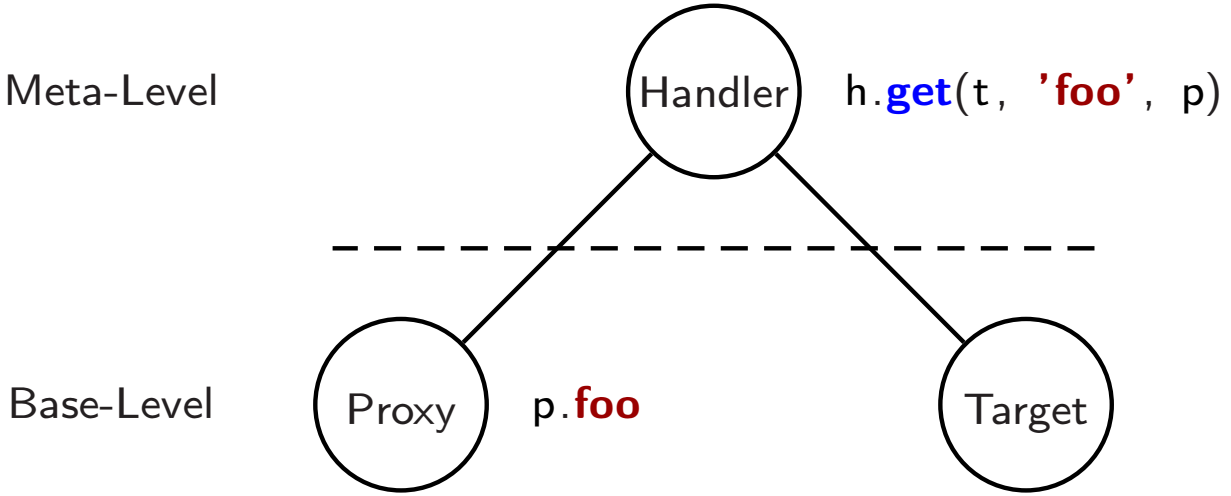
JavaScript Proxies



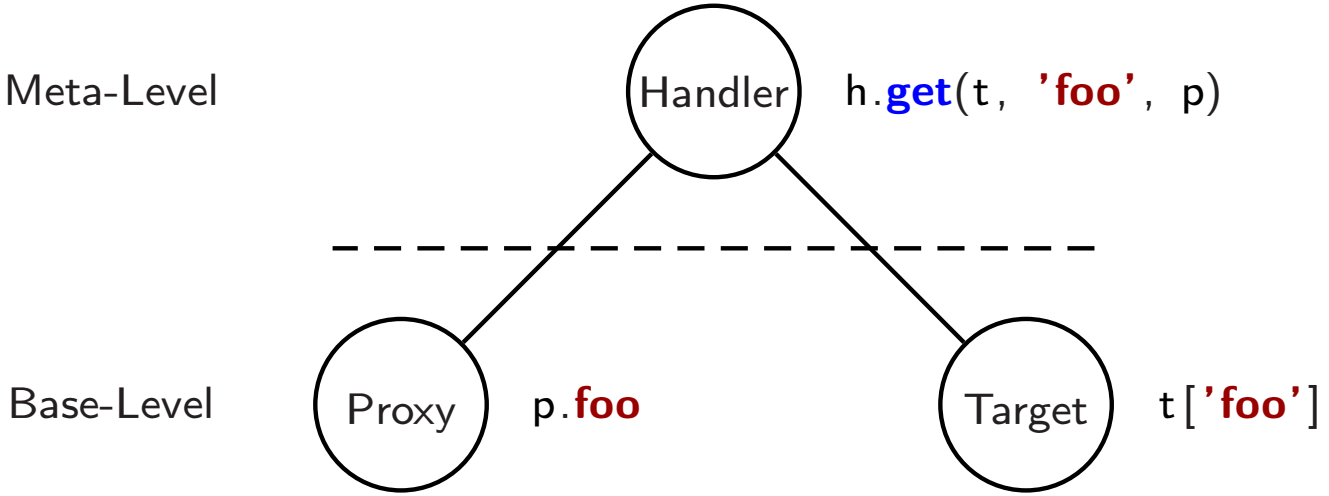
JavaScript Proxies



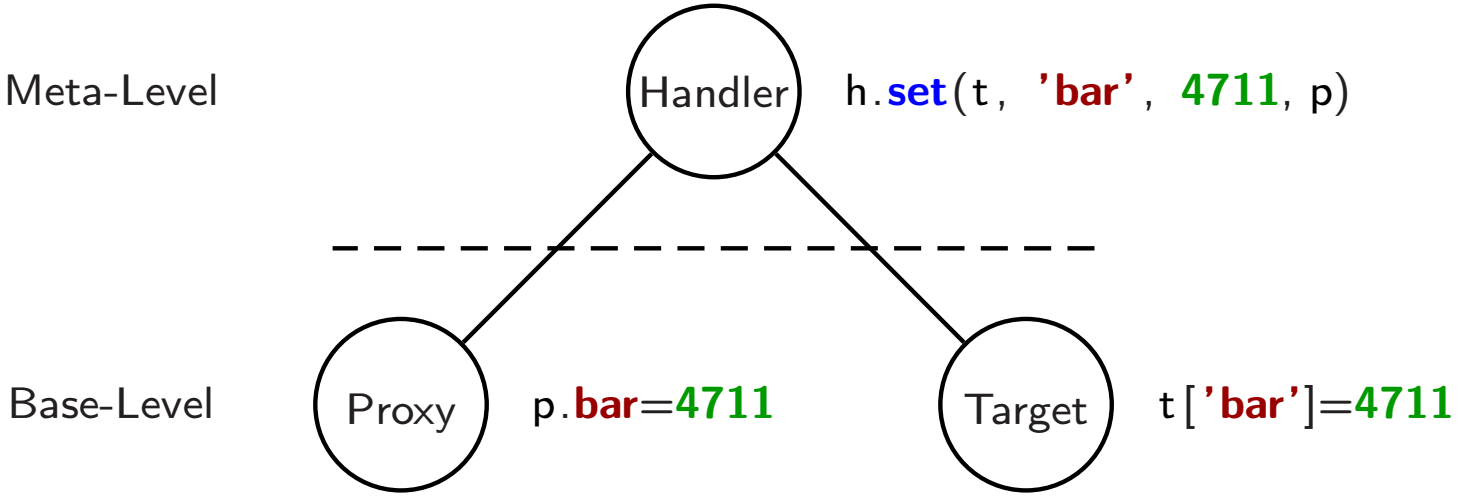
JavaScript Proxies



JavaScript Proxies

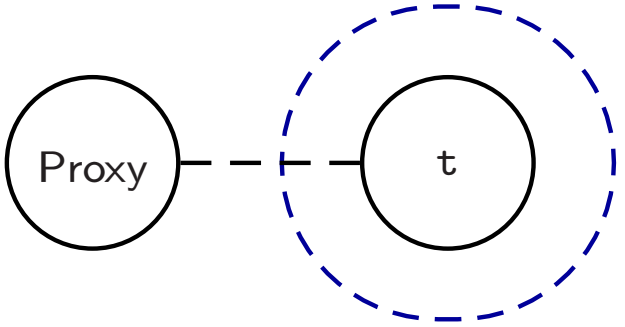


JavaScript Proxies



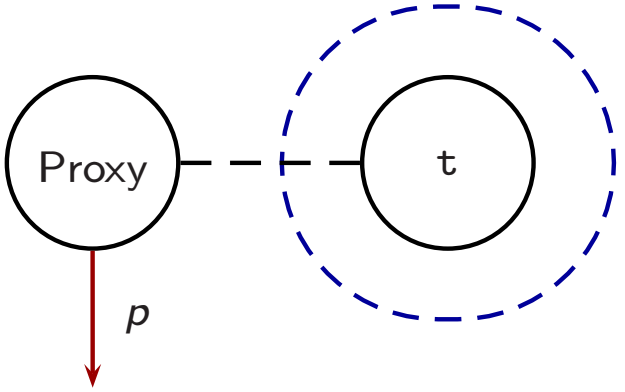
Membranes

Access Path: $\mathcal{P}$
Contract: $\mathcal{C}$



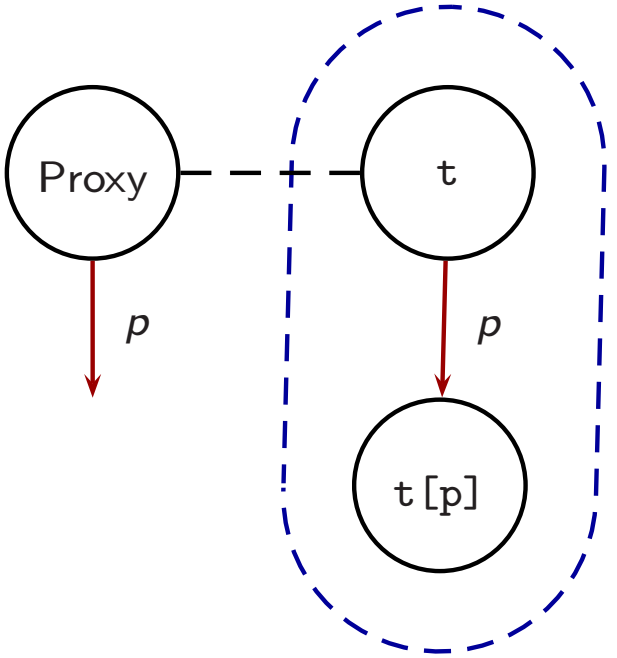
Membranes

Access Path: $\mathcal{P}$
Contract: $\mathcal{C}$



Membranes

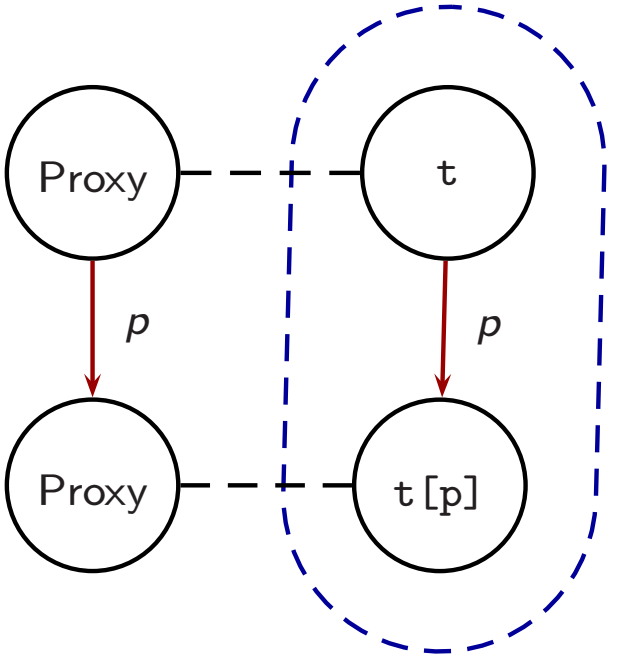
Access Path: $\mathcal{P}$
Contract: $\mathcal{C}$



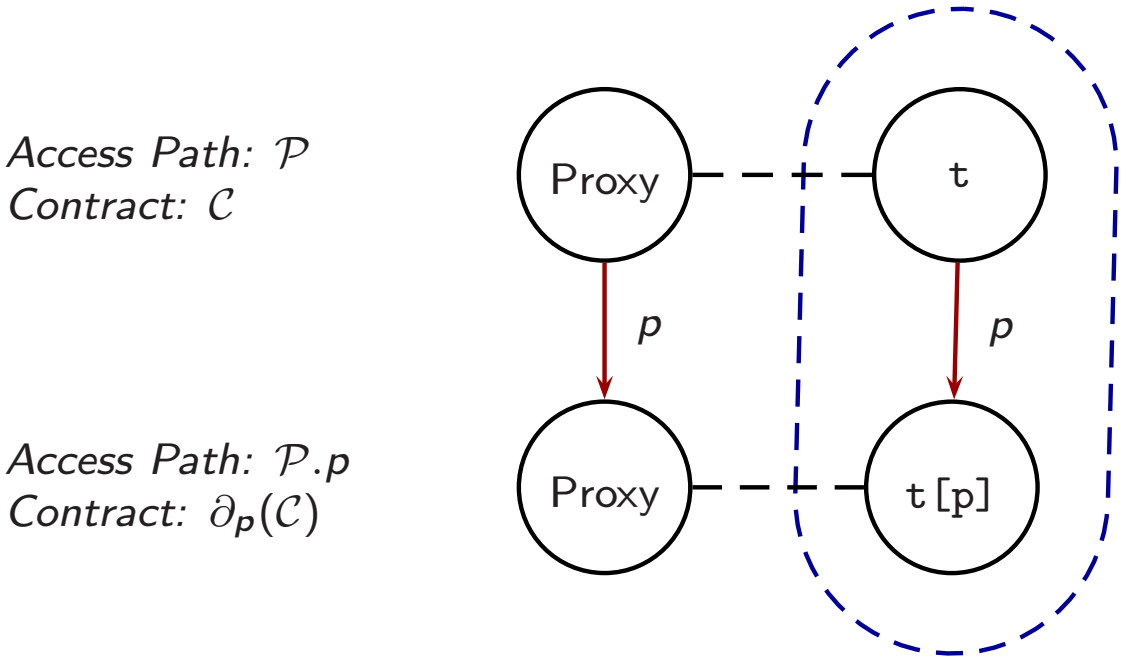
Membranes

Access Path: $\mathcal{P}$
Contract: $\mathcal{C}$

Access Path: $\mathcal{P}.p$
Contract: $\partial_p(\mathcal{C})$



Membranes



- $\partial_p(\mathcal{C})$ is the *Brzozowski derivative* of $\mathcal{C}$ with respect to p
- $\partial_p(\mathcal{C})$ accepts the quotient language:
  $p^{-1}\mathcal{L}[\![\mathcal{C}]\!] = \{w \mid pw \in \mathcal{L}[\![\mathcal{C}]\!]\}$

Membrane issues

UNI FREIBURG

What if a contract is applied to a proxy?

Membrane issues

UNI FREIBURG

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

UNI FREIBURG

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

Reuse Updated Proxy

UNI FREIBURG

Access Paths

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

Reuse Updated Proxy

UNI FREIBURG

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

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 $\mathcal{L}[\![\mathcal{C}]\!] \subseteq \mathcal{L}[\![\mathcal{C}']\!]$.

Contract Rewriting

UNI FREIBURG

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 $\mathcal{L}[\![\mathcal{C}]\!] \subseteq \mathcal{L}[\![\mathcal{C}']\!]$.

Requirement

- Decide $\mathcal{C} \sqsubseteq \mathcal{C}'$ quickly
- Use Antimirov's technique, based on derivatives

Antimirov: Deciding Containment by Rewriting

UNI FREIBURG

Lemma (Containment)

$$\mathcal{C} \sqsubseteq \mathcal{C}' \;\Leftrightarrow\; \nu(\partial_{\mathcal{P}}(\mathcal{C}')) \text{ for all } \mathcal{P} \in \mathcal{L}[\![\mathcal{C}]\!] \tag{1}$$

Antimirov: Deciding Containment by Rewriting

UNI FREIBURG

Lemma (Containment)

$$\mathcal{C} \sqsubseteq \mathcal{C}' \;\Leftrightarrow\; \nu(\partial_{\mathcal{P}}(\mathcal{C}')) \text{ for all } \mathcal{P} \in \mathcal{L}[\![\mathcal{C}]\!] \tag{1}$$

Lemma (Containment2)

$$\mathcal{C} \sqsubseteq \mathcal{C}' \;\Leftrightarrow \partial_p(\mathcal{C}) \sqsubseteq \partial_p(\mathcal{C}') \;\wedge\; (\nu(\mathcal{C}) \Rightarrow \nu(\mathcal{C}')) \quad \text{for all } p \in \{p \mid pw \in \mathcal{L}[\![\mathcal{C}]\!]\} \tag{2}$$

Antimirov: Deciding Containment by Rewriting

UNI FREIBURG

Lemma (Containment)

$$\mathcal{C} \sqsubseteq \mathcal{C}' \;\Leftrightarrow\; \nu(\partial_{\mathcal{P}}(\mathcal{C}')) \text{ for all } \mathcal{P} \in \mathcal{L}[\![\mathcal{C}]\!] \tag{1}$$

Lemma (Containment2)

$$\mathcal{C} \sqsubseteq \mathcal{C}' \;\Leftrightarrow \partial_p(\mathcal{C}) \sqsubseteq \partial_p(\mathcal{C}') \;\wedge\; (\nu(\mathcal{C}) \Rightarrow \nu(\mathcal{C}')) \quad \text{for all } p \in \{p \mid pw \in \mathcal{L}[\![\mathcal{C}]\!]\} \tag{2}$$

Drawback

Literal r leads to an infinite alphabet

- Requires infinitely many test

Literal-based derivative

Definition (First Contract Literals)

$$
\begin{array}{lcl}
\mathsf{first}(\ell) & = & \{\ell\} \\
\mathsf{first}(\epsilon) & = & \{\} \\
\mathsf{first}(\mathcal{C}*) & = & \mathsf{first}(\mathcal{C}) \\
\mathsf{first}(\mathcal{C}+\mathcal{C}') & = & \mathsf{first}(\mathcal{C}) \cup \mathsf{first}(\mathcal{C}') \\
\mathsf{first}(\mathcal{C}\&\mathcal{C}') & = & \{\ell \sqcap_r \ell' \mid \ell \in \mathsf{first}(\mathcal{C}), \ell' \in \mathsf{first}(\mathcal{C}')\} \\
\mathsf{first}(\mathcal{C}.\mathcal{C}') & = & \begin{cases} \mathsf{first}(\mathcal{C}) \cup \mathsf{first}(\mathcal{C}'), & \nu(\mathcal{C}) \\ \mathsf{first}(\mathcal{C}), & \text{otherwise} \end{cases}
\end{array}
$$

Literal-based derivative

Definition (First Contract Literals)

$$
\begin{array}{lcl}
\mathsf{first}(\ell) & = & \{\ell\} \\
\mathsf{first}(\epsilon) & = & \{\} \\
\mathsf{first}(\mathcal{C}*) & = & \mathsf{first}(\mathcal{C}) \\
\mathsf{first}(\mathcal{C}+\mathcal{C}') & = & \mathsf{first}(\mathcal{C}) \cup \mathsf{first}(\mathcal{C}') \\
\mathsf{first}(\mathcal{C}\&\mathcal{C}') & = & \{\ell \sqcap_r \ell' \mid \ell \in \mathsf{first}(\mathcal{C}), \ell' \in \mathsf{first}(\mathcal{C}')\} \\
\mathsf{first}(\mathcal{C}.\mathcal{C}') & = & \begin{cases} \mathsf{first}(\mathcal{C}) \cup \mathsf{first}(\mathcal{C}'), & \nu(\mathcal{C}) \\ \mathsf{first}(\mathcal{C}), & \text{otherwise} \end{cases}
\end{array}
$$

It holds that:

$$\{p \mid pw \in \mathcal{L}[\![\mathcal{C}]\!]\} \;=\; \mathcal{L}[\![\mathsf{first}(\mathcal{C})]\!] \tag{3}$$

Containment with Contract Literals

- $\nabla_\ell(\mathcal{C})$ is the literal-based derivative of $\mathcal{C}$ with respect to ℓ

Lemma (Syntactic derivative of contracts)

$$\mathcal{L}[\![\nabla_\ell(\mathcal{C})]\!] = \bigcap_{p \in \mathcal{L}[\![\ell]\!]} \mathcal{L}[\![\partial_p(\mathcal{C})]\!] \tag{4}$$

Containment with Contract Literals

- $\nabla_\ell(\mathcal{C})$ is the literal-based derivative of $\mathcal{C}$ with respect to ℓ

Lemma (Syntactic derivative of contracts)

$$\mathcal{L}[\![\nabla_\ell(\mathcal{C})]\!] = \bigcap_{p \in \mathcal{L}[\![\ell]\!]} \mathcal{L}[\![\partial_p(\mathcal{C})]\!] \tag{4}$$

Theorem (Containment)

$$\mathcal{C} \sqsubseteq \mathcal{C}' \Leftarrow \nabla_\ell(\mathcal{C}) \sqsubseteq \nabla_\ell(\mathcal{C}') \wedge (\nu(\mathcal{C}) \Rightarrow \nu(\mathcal{C}')) \quad \textit{for all } \ell \in \textit{first}(\mathcal{C}) \tag{5}$$

Implementation

UNI FREIBURG

- Implementation based on the JavaScript Proxy API
  - Implemented since Firefox 18.0 and Chrome 3.5

Implementation

- Implementation based on the JavaScript Proxy API
  - Implemented since Firefox 18.0 and Chrome 3.5
- Implementation provides an proxy-handler

Implementation

UNI FREIBURG

- 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

- 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

Evaluation

UNI FREIBURG

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

Evaluation

UNI FREIBURG

- 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 ?*

Evaluation

UNI FREIBURG

- 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

UNI FREIBURG

- 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

UNI FREIBURG

Benchmark	Baseline	Contracts only	Without logging	Full
RegExp	2.4sec	2.4sec	2.4sec	2.4sec
NavierStokes	2.3sec	2.3sec	2.3sec	2.3sec
EarleyBoyer	4.3sec	4.4sec	4.4sec	4.4sec
DeltaBlue	2.3sec	3.3sec	9.5sec	9.8sec
Richards	2.3sec	3.3sec	18.6min	22.5min
RayTrace	2.3sec	1.6min	1.1h	1.2h
Crypto	4.4sec	2.6min	2.5h	4.2h
Splay	2.3sec	2.3sec	-	-

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

Conclusion

UNI FREIBURG

- 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

UNI FREIBURG

- 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

Conclusion

UNI FREIBURG

- 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 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.