The Eval That Men Do

Conference Paper · July 2011

DOI: 10.1007/978-3-642-22655-7_4

CITATIONS
67

READS
419

4 authors, including:

Christian Hammer
Universität Potsdam
54 PUBLICATIONS **1,234** CITATIONS

SEE PROFILE

Jan Vitek
Northeastern University
297 PUBLICATIONS **5,876** CITATIONS

SEE PROFILE

The Eval that Men Do

A Large-scale Study of the Use of Eval in JavaScript Applications

by Gregor Richards et al.

Changhee Park @ PLRG

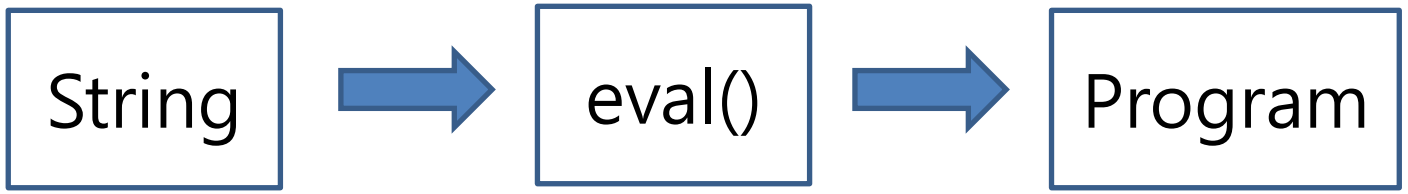
2011. 3. 18

What is eval?

> *eval is evil. Avoid it.*
> *eval has aliases. Don't use them.*
> —Douglas Crockford

What is eval?

- eval()



- Ex)

```
eval("var a=3; var b=4; a+ b")
```

→ 7

The power of eval

- What eval can do …
  - New library installation
  - Adding and removing field and method from objects
  - Changing prototype hierarchy

The power of eval

- Scope access
  - Global scope : indirect call
    - Ex) var anothereval = eval
  - Local scope : direct call
    - Ex)

```
Point = function() {
    var x=0; var y=0;
    return function(op,sel,val) {
      if(op=="r") return eval(sel);
      if(op=="w") return eval(sel+"="+val);
    }
}
```

```
p = Point();
p("r","x"); // returns 0
p("w","y",3); // set y to 3
p("r","y"); // returns 3
```

Handling eval

- Some researches …

1. Christopher Anderson and Sophia Drossopoulou. BabyJ: From object based to class based programming via types. *Electr. Notes Theor. Comput. Sci.*, 82(7), 2003.
2. Christopher Anderson and Paola Giannini. Type checking for JavaScript. *Electr. Notes Theor. Comput. Sci.*, 138(2), 2005.

12. Dongseok Jang and Kwang-Moo Choe. Points-to analysis for JavaScript. In *Symposium on Applied Computing (SAC)*, 2009.

24. Peter Thiemann. Towards a type system for analyzing JavaScript programs. In *European Symposium on Programming (ESOP)*, 2005.

Ignore eval!!

Handling eval

- Some researches …

8. S. Guarnieri and Benjamin Livshits. Gatekeeper: Mostly static enforcement of security and reliability policies for JavaScript code. In *USENIX Security Symposium*, 2009.
19. Jan Kasper Martinsen and Hakan Grahn. A comparative evaluation of the execution behavior of javascript benchmarks and real-world web applications (poster). In *Symposium on Computer Performance, Modeling, Measurements and Evaluation (Performance)*, 2010.

Assume eval is hardly used

Handling eval

- Some researches …

9. Arjun Guha, Shriram Krishnamurthi, and Trevor Jim. Using static analysis for ajax intrusion detection. In *International Conference on World Wide Web (WWW)*, 2009.

14. Simon Holm Jensen, Anders Møller, and Peter Thiemann. Type analysis for JavaScript. In *Static Analysis Symposium (SAS)*, 2009.

Assume eval is used safely

- [9] assumes eval is used mainly for JSON deserialization and sometimes for loading of library code

Handling eval

- JSON(JavaScript Object Notation)
  - EX)

```
{ "Image": { "Title": "View from 15th Floor", "IDs": [116, 943, 234, 38793],
    "Thumbnail": { "Height": 125, "Width": "100" }}}
```

- JSON serialization
  - Object -> String
- JSON deserialization
  - String -> Object

Handling eval

- Some researches …

4. Ravi Chugh, Jeffrey A. Meister, Ranjit Jhala, and Sorin Lerner. Staged information flow for JavaScript. In *Conference on Programming language design and implementation (PLDI)*, pages 50–62, 2009.
5. Manuel Egele, Peter Wurzinger, Christopher Kruegel, and Engin Kirda. Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks. In *Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2009.
7. Ben Feinstein and Daniel Peck. Caffeine monkey: Automated collection, detection and analysis of malicious JavaScript. In *Black Hat USA 2007*, 2007.
13. Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. An empirical study of privacy-violating information flows in JavaScript web applications. In *Conference on Computer and communications security (CSS*, pages 270–283, 2010.
17. Sergio Maffeis, John Mitchell, and Ankur Taly. Isolating JavaScript with filters, rewriting, and wrappers. In *Computer Security (ESORICS)*, pages 505–522. 2009.

Assume eval is a serious security threat

Handling eval

- Summary of assumptions
  - eval is hardly used
  - eval is safely used
    - eval is used primarily for JSON deserialization
  - eval is a serious security threat

Which one is true??

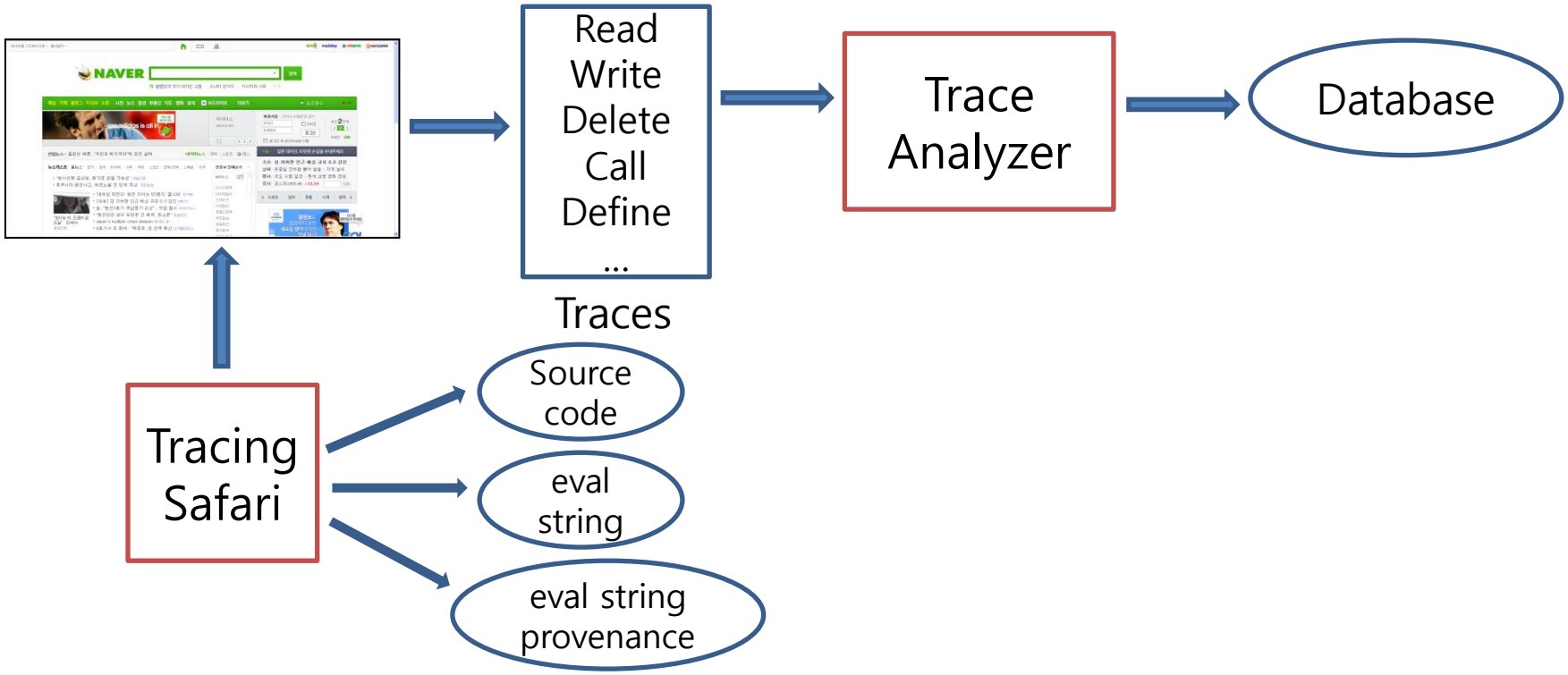
This paper

- Conducts a thorough evaluation of the real-world use of eval

Methodology

Methodology

- Infrastructure
  - TracingSafari : an instrumented version of WebKit



Methodology

- Corpus
  - The most popular top 100 and 10000 sites according to alexa.com
  - Three kinds of executions

INTERACTIVE	Manual interaction with web sites.
PAGELOAD	Data set obtained by recording JavaScript behavior for 30 seconds when a web page is loaded.
RANDOM	Data set obtained by recording 30 seconds of page load activity and randomly generated events.

Methodology

- Corpus
  - The rationale for three data sets

Data Sets	Good	Bad
Interactive	Most representative	Small coverage
Pageload	Large coverage	No interaction
Random	Large coverage	Unrealistic

Methodology

- Limitation
  - No consideration for dynamic code injection provided by DOM
    - Ex) document.write, document.createElement(“script”)
  - No exhaustive coverage
  - Only results in WebKit and Safari

Usage of Eval

JavaScript and eval usase

- Usage statistics(JS percentage)

Data Set	JavaScript used	eval use	Avg eval (bytes)	Avg eval calls	total eval calls	total eval size (bytes)	total JS size (MB)
INTERACTIVE	100%	59%	1486	38	2,434	3,616,822	59.8
PAGELOAD	91%	41%	685	28	111,866	76,669,599	1,725
RANDOM	91%	43%	687	85	367,544	252,340,684	1,829

  - The top most 100 : 100 %
  - The top most 10000 : 91 %

JavaScript and eval usase

- Usage statistics(JS size)

Data Set	JavaScript used	eval use	Avg eval (bytes)	Avg eval calls	total eval calls	total eval size (bytes)	total JS size (MB)
INTERACTIVE	100%	59%	1486	38	2,434	3,616,822	59.8
PAGELOAD	91%	41%	685	28	111,866	76,669,599	1,725
RANDOM	91%	43%	687	85	367,544	252,340,684	1,829

JavaScript and eval usase

- Usage statistics(JS size)

Outliers

Data Set	JavaScript used	eval use	Avg ev (bytes	l size s)	total JS size (MB)
INTERACTIVE	100%	59%	1486	322	59.8
PAGELOAD	91%	41%	685	599	1,725
RANDOM	91%	43%	687	,684	1,829

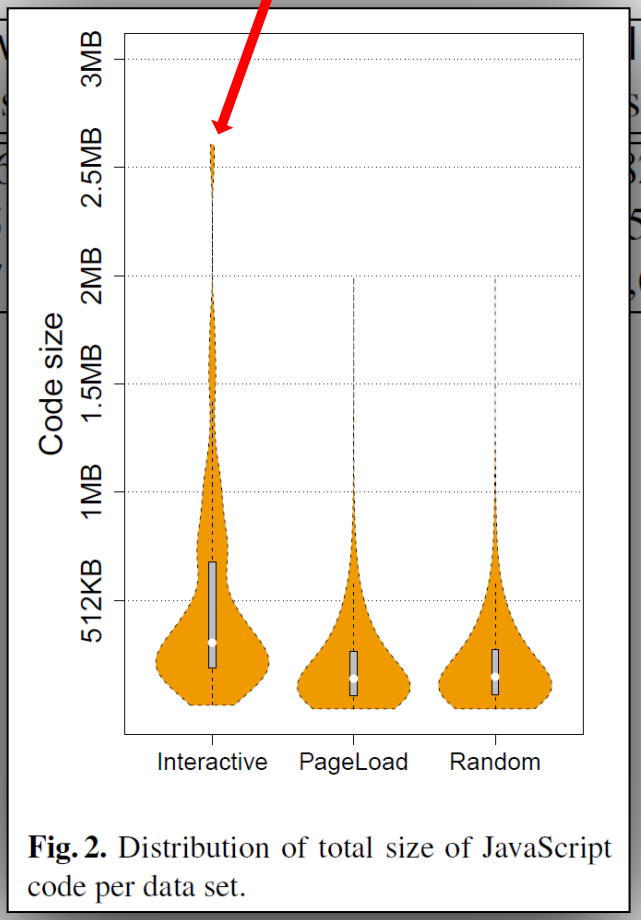


Fig. 2. Distribution of total size of JavaScript code per data set.

JavaScript and eval usase

- Usage statistics(eval percentage)

Data Set	JavaScript used	eval use	Avg eval (bytes)	Avg eval calls	total eval calls	total eval size (bytes)	total JS size (MB)
INTERACTIVE	100%	59%	1486	38	2,434	3,616,822	59.8
PAGELOAD	91%	41%	685	28	111,866	76,669,599	1,725
RANDOM	91%	43%	687	85	367,544	252,340,684	1,829

  - Total 481,833 calls and 317MB string data
  - Pageload 41% vs Random 43%

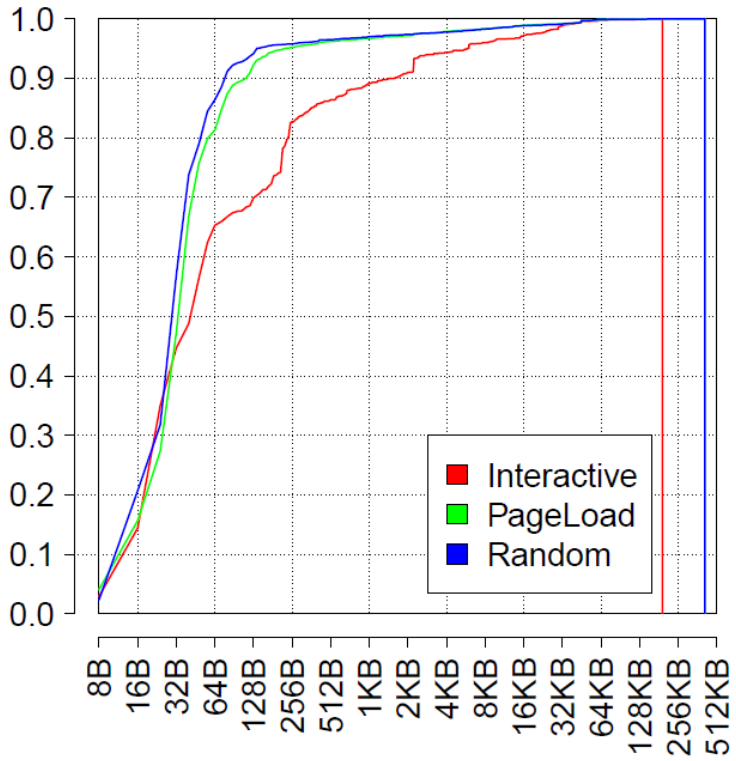
JavaScript and eval usase

- Usage statistics(eval size)

Data Set	JavaScript used	eval use	Avg eval (bytes)	Avg eval calls	total eval calls	total eval size (bytes)	total JS size (MB)
INTERACTIVE	100%	59%	1486	38	2,434	3,616,822	59.8
PAGELOAD	91%	41%	685	28	111,866	76,669,599	1,725
RANDOM	91%	43%	687	85	367,544	252,340,684	1,829

JavaScript and eval usase

- Distribution of eval string sizes



- Below 64B
  - Interative : 2/3
  - Pageload : 80%
  - Random : 85%
- Maximum
  - Interactive : 193KB
  - Pageload, Random : 413KB

JavaScript and eval usase

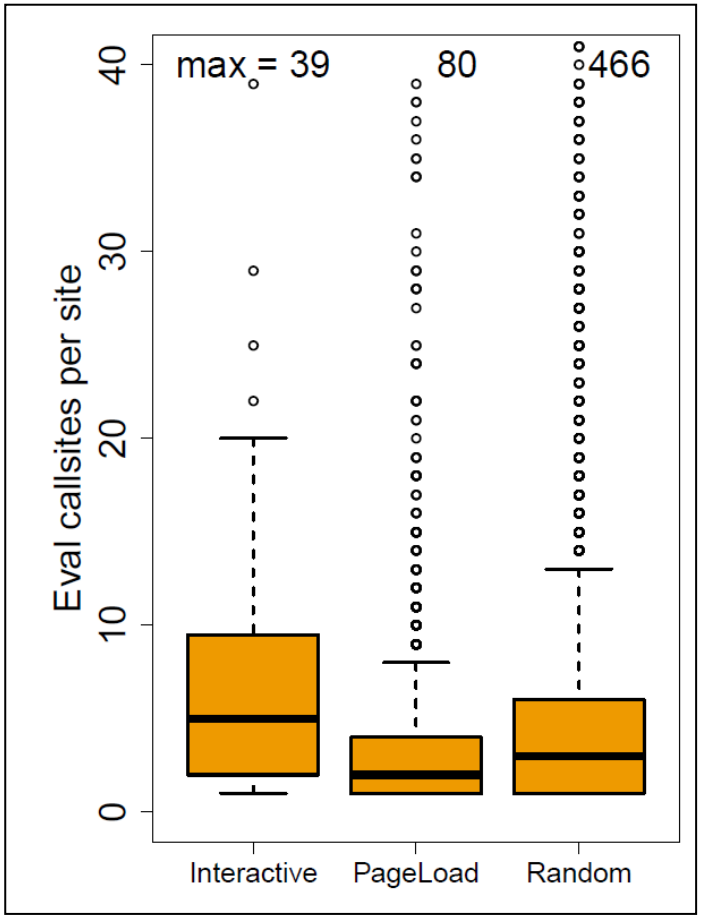
- Usage statistics(eval calls)

Data Set	JavaScript used	eval use	Avg eval (bytes)	Avg eval calls	total eval calls	total eval size (bytes)	total JS size (MB)
INTERACTIVE	100%	59%	1486	38	2,434	3,616,822	59.8
PAGELOAD	91%	41%	685	28	111,866	76,669,599	1,725
RANDOM	91%	43%	687	85	367,544	252,340,684	1,829

  - eval in the whole life cycle of web pages
  - Average eval calls
    - Interactive 38 vs Random 85

JavaScript and eval usase

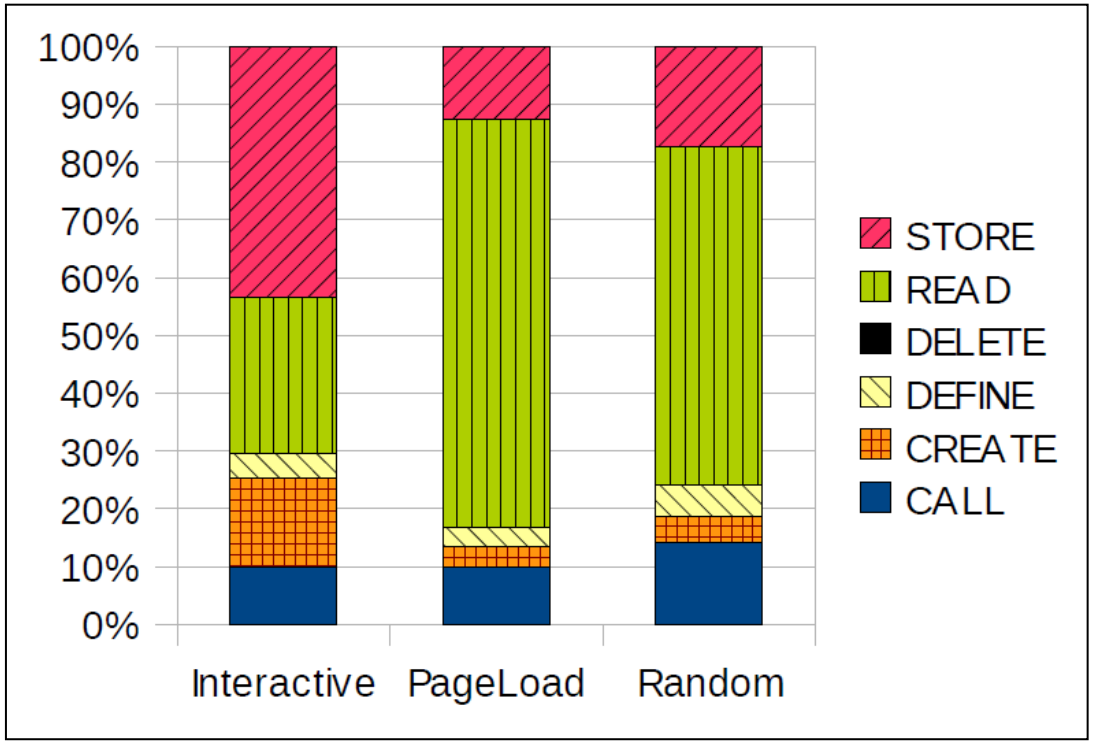
- Distribution of number of eval call sites per site



- Lower mean value in PageLoad
- Max number in PageLoad : 80

JavaScript and eval usase

- Distribution of operation types in eval



- More STORE and CREATE in Interactive : JSON-like object
- More CALL in Random

JavaScript and eval usase

- Common libraries

Data Set	jQuery	Prototype	MooTools
INTERACTIVE	54%	9%	10%
PAGELOAD	48%	6%	4%
RANDOM	52%	7%	5%

  - Some libraries loaded for dynamism
  - MooTools popular in top 100
  - Google Closure excluded

A Taxonomy of Eval

Taxonomy of eval

- 4 axes
  - Scope
    - Changing shared variables violate assumptions
  - Patterns
    - Enable purpose-specific analyses
  - Provenance
    - For the analyses related to code injection
  - Consistency

A Taxonomy of Eval :
Scope

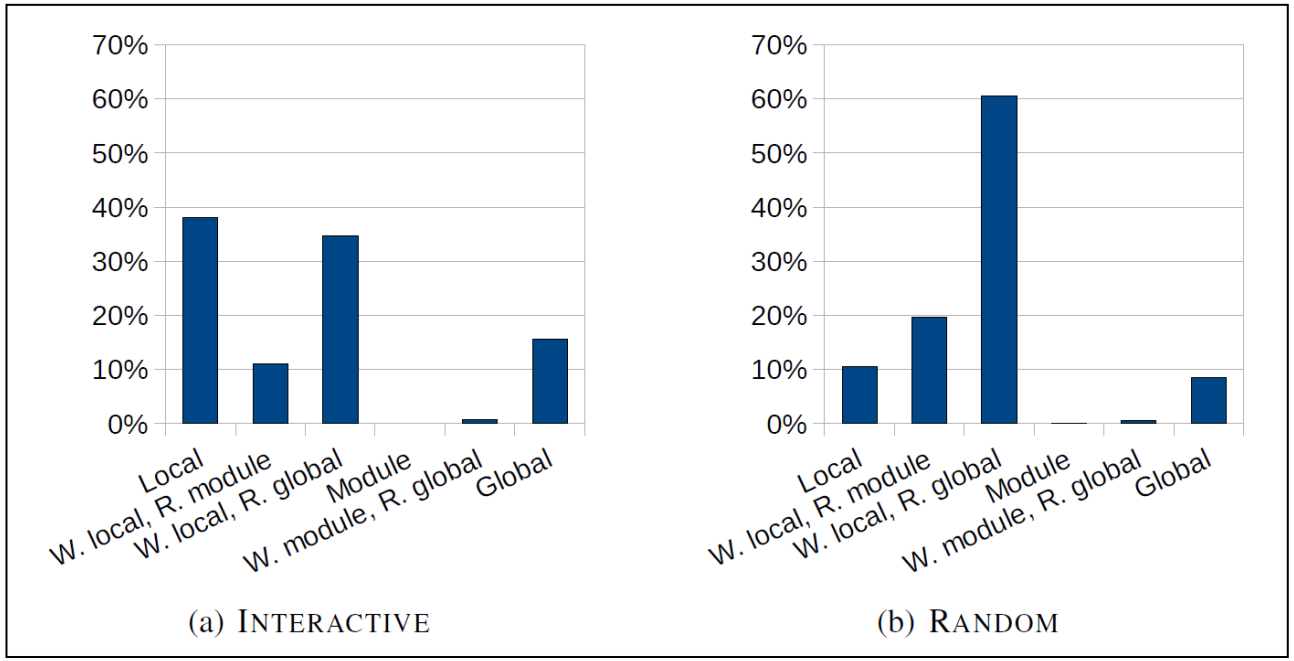
Scope

- Categorization of the locality

Data Sets	Read	Write
Purely local	Local	Local
Writes local, reads module	Module	Local
Writes local, reads global	Global	Local
Purely module-local	Module	Module
Writes module, reads global	Global	Module
Global	Global	Global

Scope

- Scope of eval



- Pure but not self-contained
- Potentially harm

A Taxonomy of Eval :
Patterns

Patterns

- 11 categories
  1. JSON
  2. Relaxed JSON
  3. =JSON
  4. Member
  5. Variable
  6. Variable declaration
  7. Typeof
  8. Try/catch
  9. Call
  10. Library
  11. Other

Patterns

- 11 categories
  1. JSON

     - Strict JSON format defined by ECMAScript standard

  2. Relaxed JSON

     - No quotation or single quotation allowed instead of double quotation

     - ex) {x:0}, {‘x’:0}

Patterns

- 11 categories
  3. =JSON
     - Ex) eval(“v={x:0}”)
  4. Member
  5. Varaiable
     - Easy to access global variables
  6. Variable declaration
     - Modifies the local scope

Patterns

- 11 categories
  7. Typeof
     - Ex) typeof(x) != “undefined”
  8. Try/catch
  9. Call
     - Ex) document.getElementById

Patterns

- 11 categories

  10. Library

  - Each string longer than 512 bytes which defines function

  - Why? How?

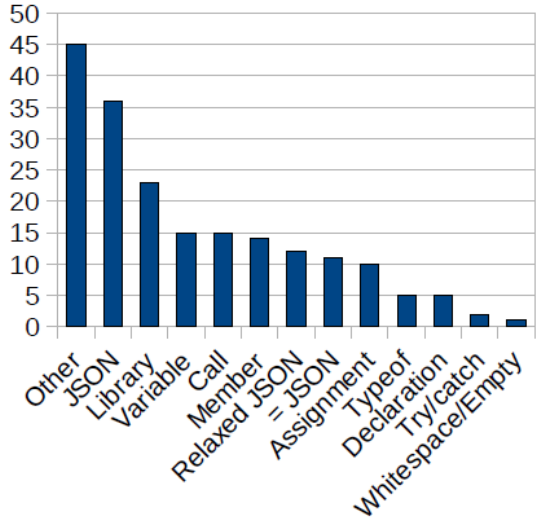
- Answers
  - Combination of AJAX(XMLHttpRequest) and eval prevents page rendering from blocking with <script> tag
  - 512 bytes obtained by semantic analysis

Patterns

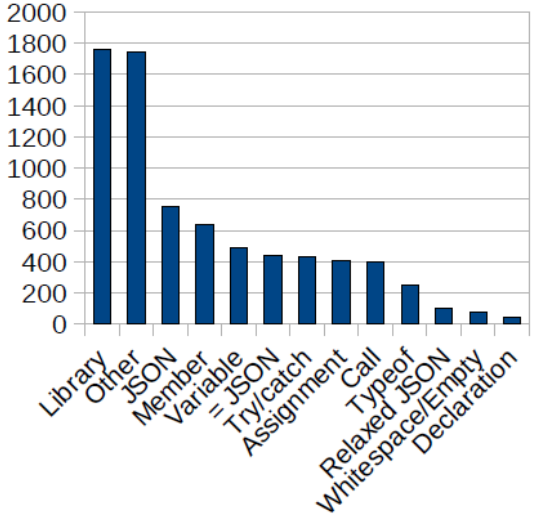
- 11 categories
  11. Other
      - Empty string and white space
      - Other complex code

Patterns

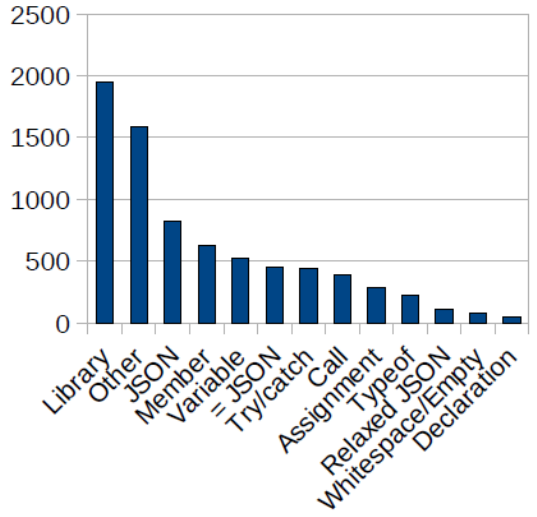
- The number of web sites



(a) INTERACTIVE



(b) PAGELOAD

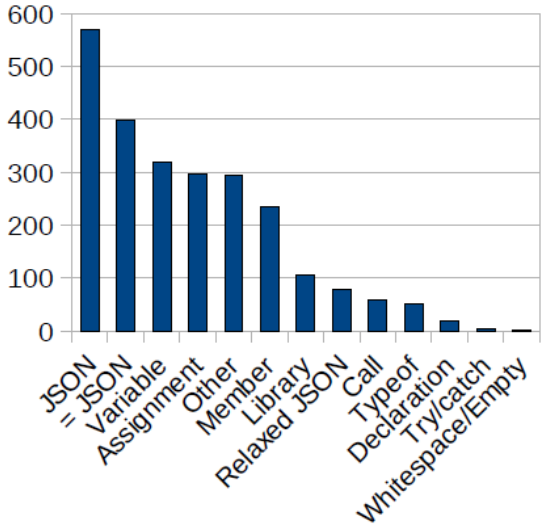


(c) RANDOM

Most are uncategorizable!!

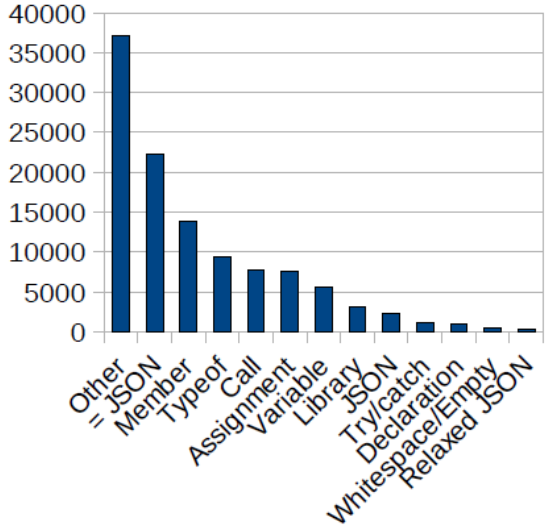
Patterns

- The number of **eval**s



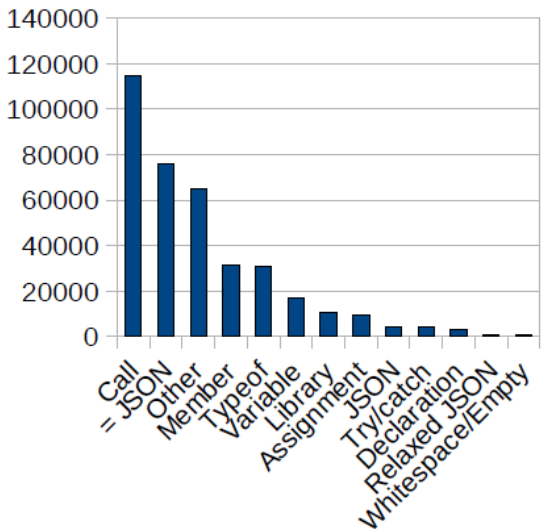
(a) INTERACTIVE

Other : 12.1%
JSON : 44%
Strict JSON



(b) PAGELOAD

Other : 33.1%



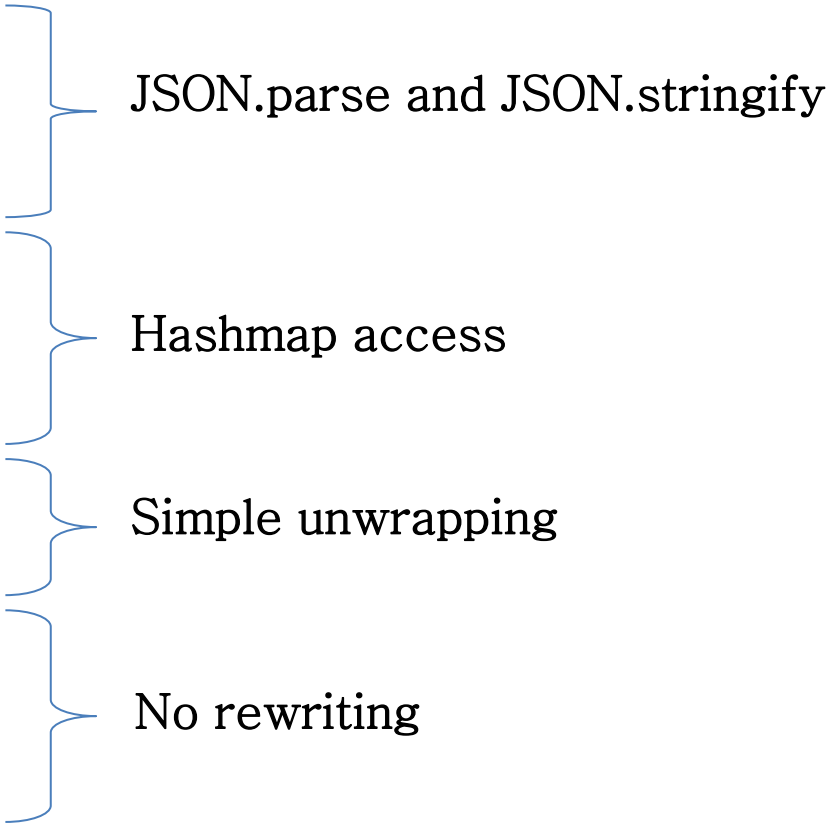
(c) RANDOM

Other : 17.7%
JSON : 21%
CALL : 31%

- Those with side-effect are less common

Rewriting eval

- Rewritable pattern
  1. JSON
  2. Relaxed JSON
  3. =JSON
  4. Member
  5. Variable
  9. Call
  7. Typeof
  8. Try/catch
  6. Variable declaration
  10. Library
  11. Other



Rewriting eval

- Rewritable patterns
  - Hashmap access
    - 4. Member and 5. Variable
      - eval(“foo.”+ x+ “=3;”) => foo[x]=3;
    - 9. Call
      - eval(“update(obj);”) => window[“update”](obj)

Rewriting eval

- Rewritable patterns
  - Simple unwrapping
    - 7. Typeof
      - typeof(x) != “undefined” => “x” in window
    - 8. Try/catch
      - try{throw v=14} catch(e){} => v=14

Rewriting eval

- Rewritable patterns
  - Possible in categories other than Variable declaration, Library, and Other
  - 83%

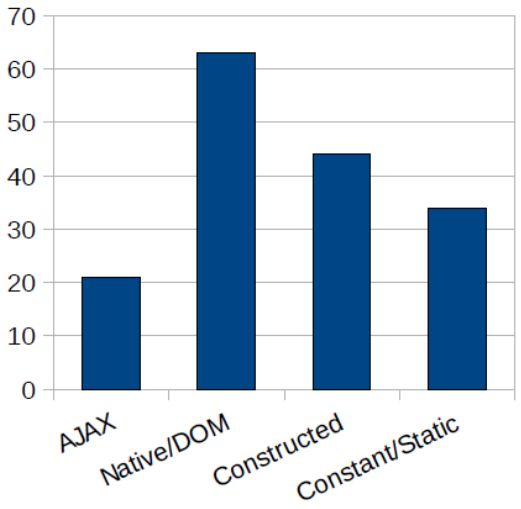
A Taxonomy of Eval :
Provenance

Provenance

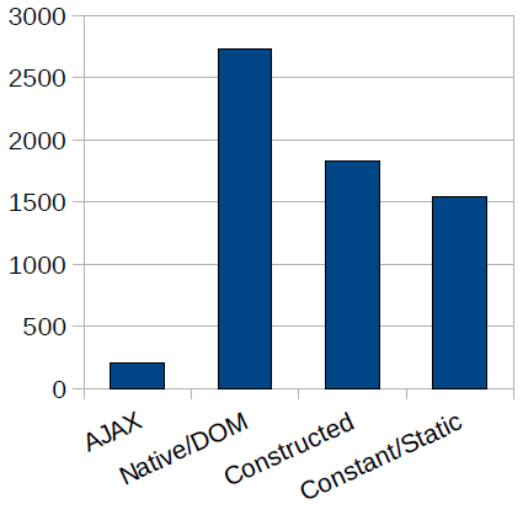
- 4 categories
  - AJAX : string from AJAX call
  - Native/DOM : string from native method or DOM
  - Constructed : concatenated string
  - Constant

Provenance

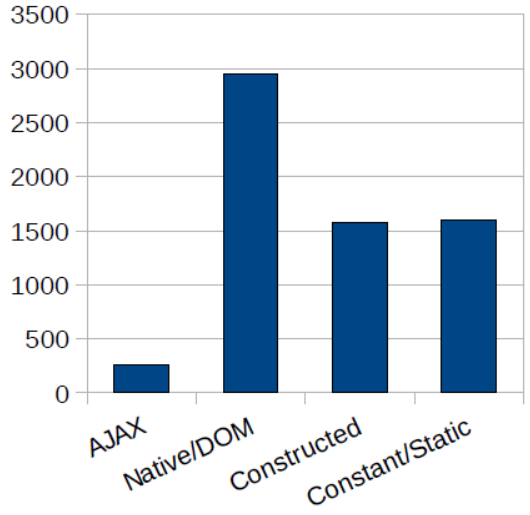
- The number of sites



(a) INTERACTIVE



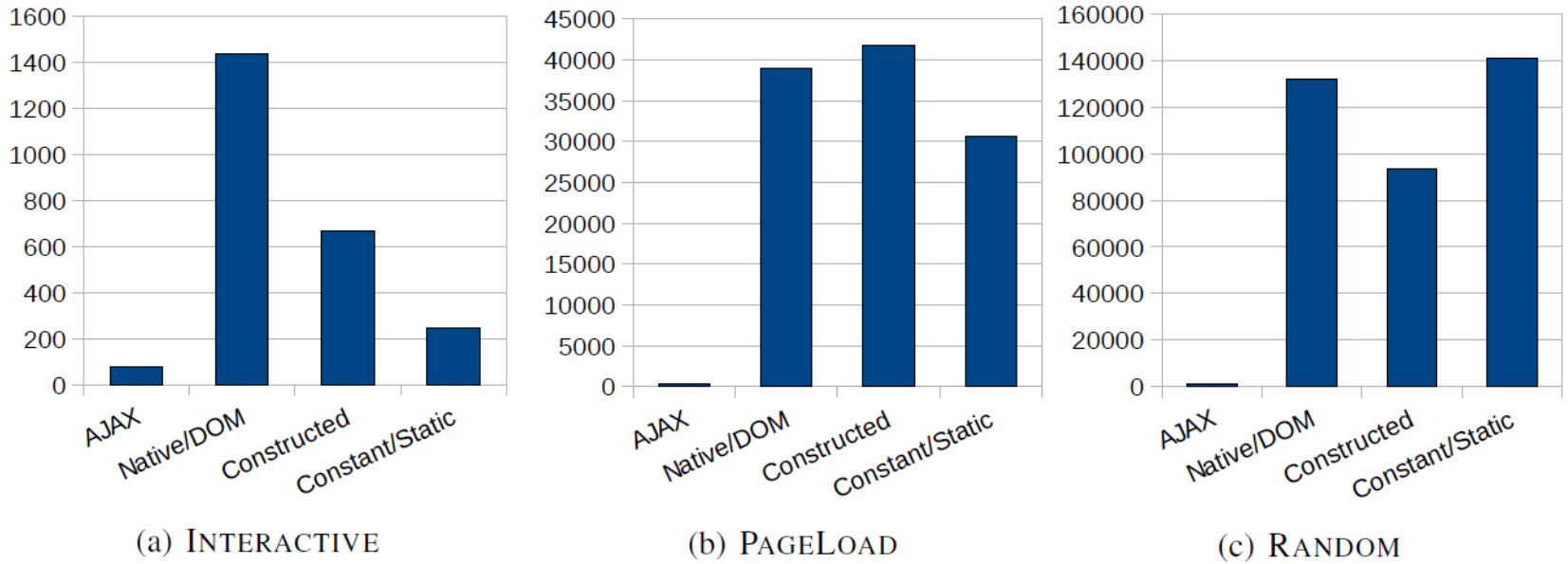
(b) PAGELOAD



(c) RANDOM

Provenance

- The number of eval strings

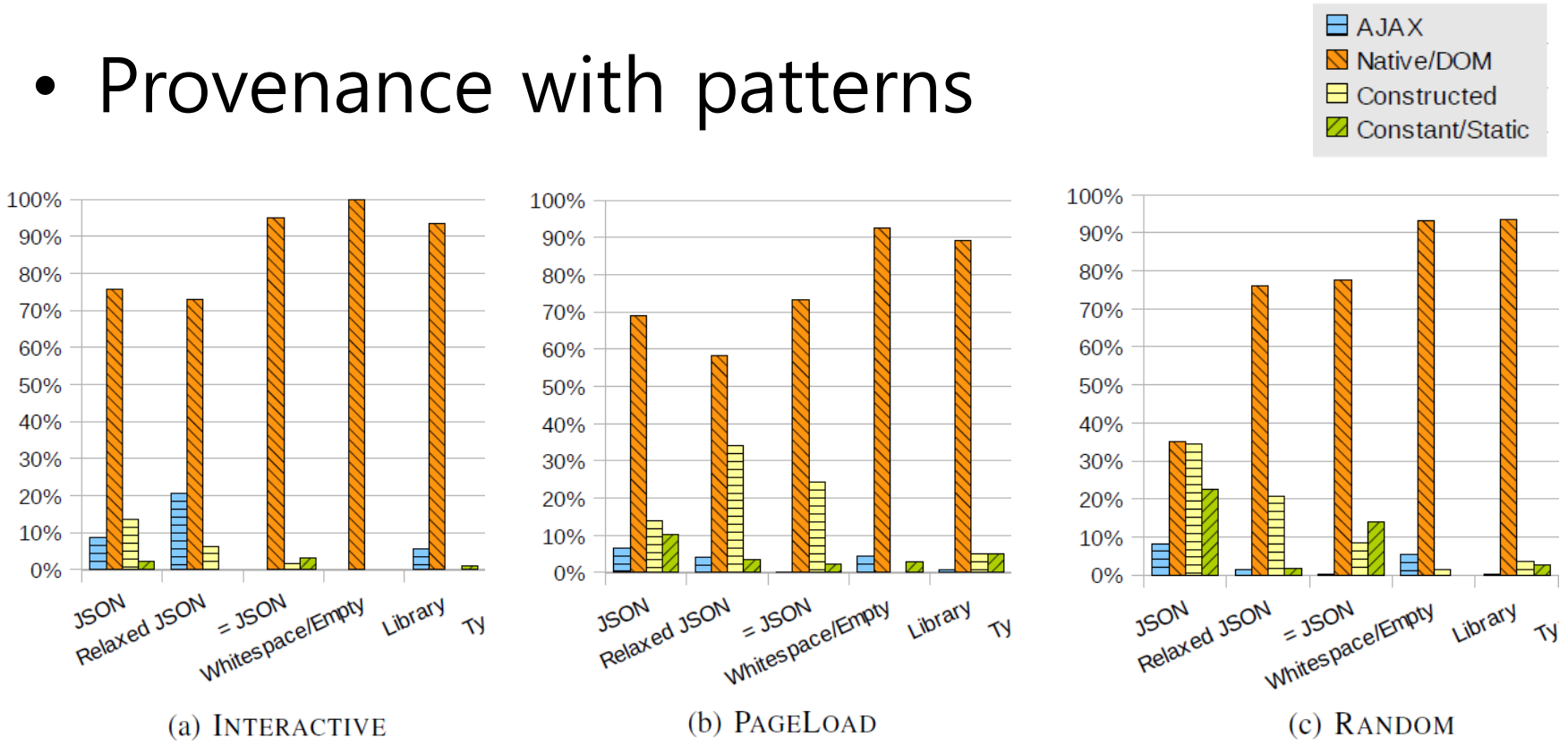


(a) INTERACTIVE (b) PAGELOAD (c) RANDOM

AJAX : much less common

Provenance

- Provenance with patterns



(a) INTERACTIVE (b) PAGELOAD (c) RANDOM

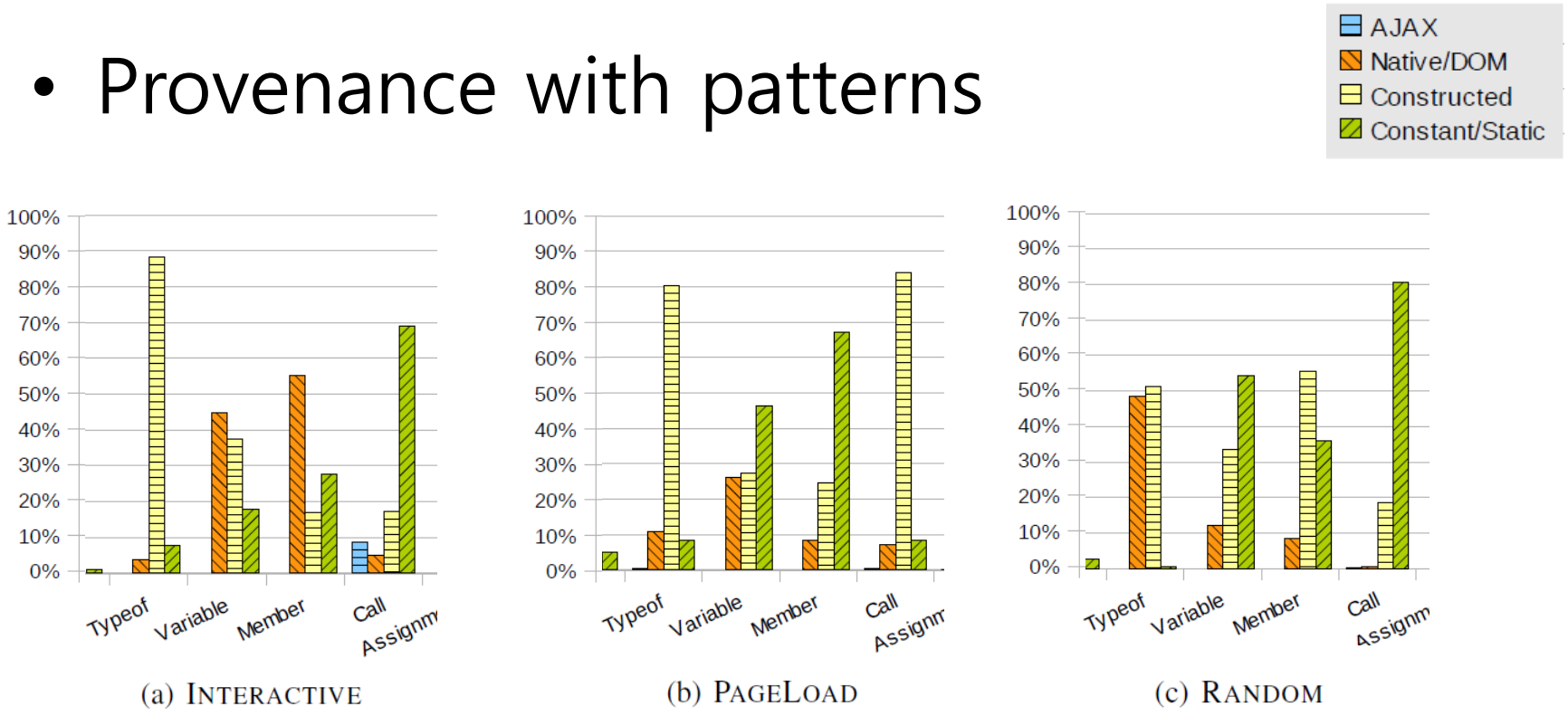
JSON is not mainly originated from AJAX!!

Provenance

- JSON non-originated from AJAX
  - Ex) google.com
    - uses a dynamically created script tag
    - JSON string is considered as compile time constant in that tag
    - has a separate server with sub-domain containing JavaScript code
    - JS code from AJAX is limited by SOP(Same Origin Policy)

Provenance

- Provenance with patterns



(a) INTERACTIVE (b) PAGELOAD (c) RANDOM

A Taxonomy of Eval : Consistency

Consistency

- Inconsistent evals : 431 call sites
- EX)
  - Constant switch
    : “4” -> “5” -> “a”
  - Field <-> method
    : window.location -> dw_Inf.get(dw_Inf.ar) -> dw_Inf.x0()
  - JSON <-> non-JSON
    : “(null)” -> “(undefined)”

Contribution

- Infrastructure tracking JavaScript behavior
- Large scale survey over 10,000 most popular websites
- Detailed analysis of eval in JavsScript

Utilize it for the further research

Lessons

- eval is hardly used
  - False
  - 59% of the most popular websites
- eval is safely used
  - Partly true
  - Assignment and declarations are less common

Lessons

- eval is used primarily for JSON deserialization
  - False
  - At most 45%
- 83% of eval can be rewritten