The Eval that Men Do

Gregor Richard Christian Hammer Brian Burg Jan Vitek

Vincent Foley-Bourgon
COMP-621 - Winter 2014
McGill University

February 2014

The paper

Information

- 3 authors from Purdue University (Go Boilermakers!)
- Presented at ECOOP 2011
- Empirical study of the usage of *eval*

Plan

1. What is *eval*?
2. Methodology
3. Results
4. Conclusion

Disclaimers

1. Tables and figures are taken from the paper

Disclaimers

1. Tables and figures are taken from the paper
2. Won't show all the results

Disclaimers

1. Tables and figures are taken from the paper
2. Won't show all the results
3. Personal bias

What is *eval*?

What is *eval*?

Question: Who can tell me what *eval* does?[1]

[1] Someone who is not part of the MCJS team

What is *eval*?

eval takes a string as input and executes it

```
var name = "bar";
eval("o." + name + "='" + name + "!!'");
```

Same as:

```
o.bar = 'bar!!';
```

What is *eval*?

eval takes a string as input and executes it

```
var name = "bar";
eval("o." + name + "='" + name + "!!'");
```

Same as:

```
o.bar = 'bar!!';
```

You can pass any arbitrary string to *eval*:

- Assignments
- Conditionals and loops
- Functions
- Other calls to *eval*

What is *eval*?

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

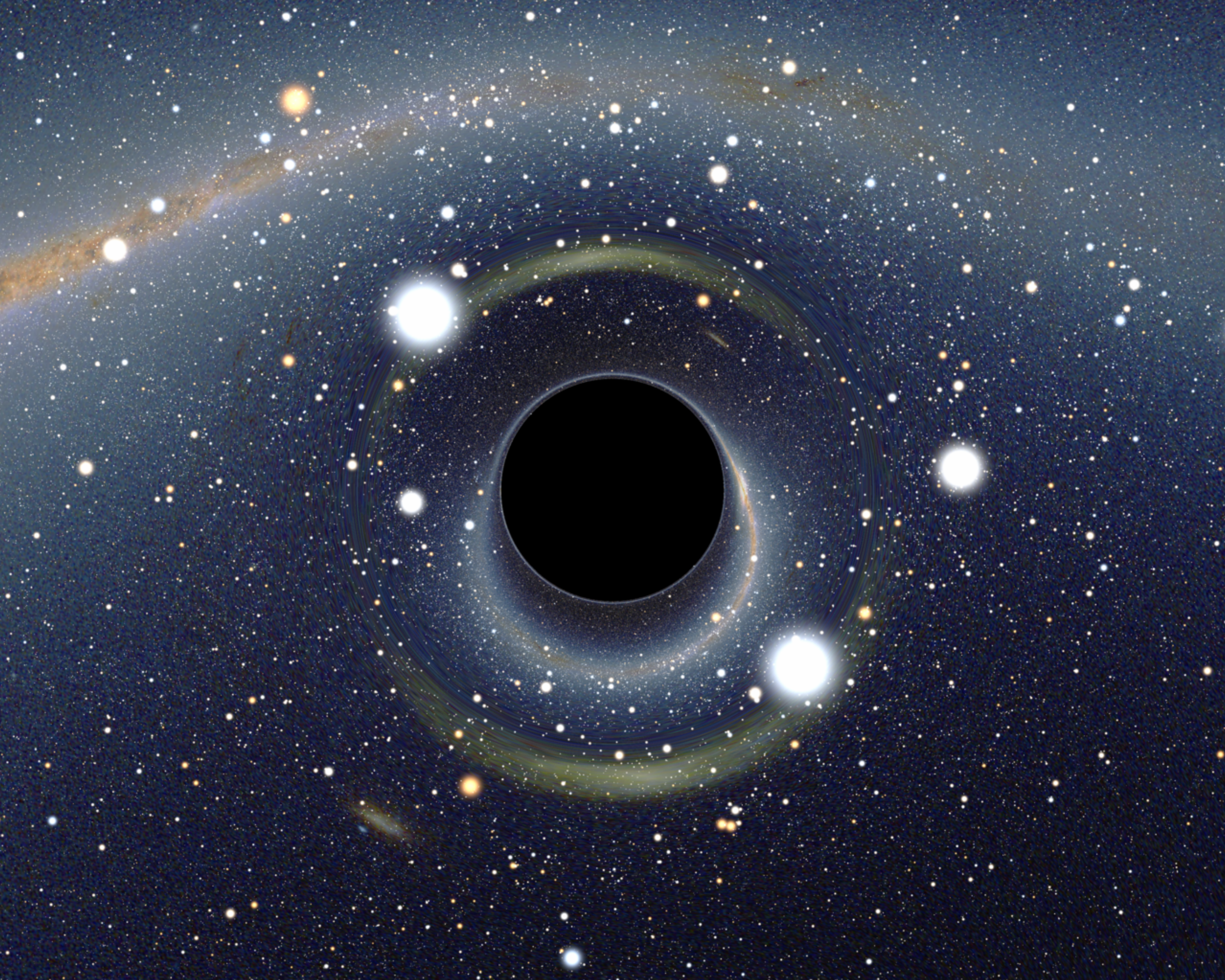
What is *eval*?

Question: What are some problems with *eval*?

What is *eval*?

Question: What are some problems with *eval*?

How does it affect static analysis?



What is *eval*?

eval is the black hole of static analysis

- It kills everything
- It generates nothing

What is *eval*?

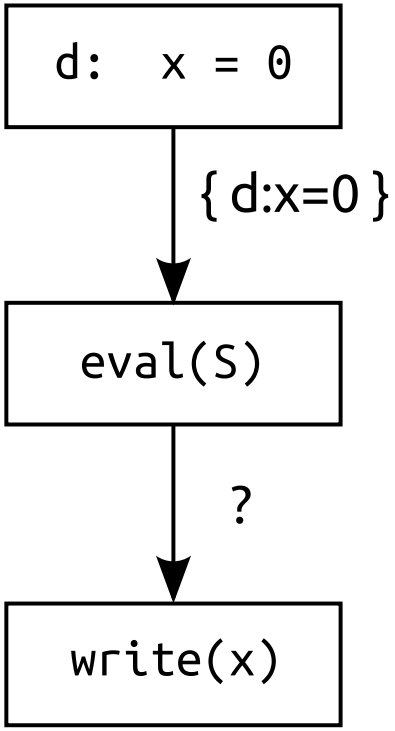
Example: Reaching defs

Let's think about how *eval* would affect *reaching defs*.

"A definition `d: x = ...` reaches a point p if there exits a path from d to p that does **not** pass through another definition of x."

What is *eval*?

Example: Reaching defs



What is *eval*?

Example: Reaching defs

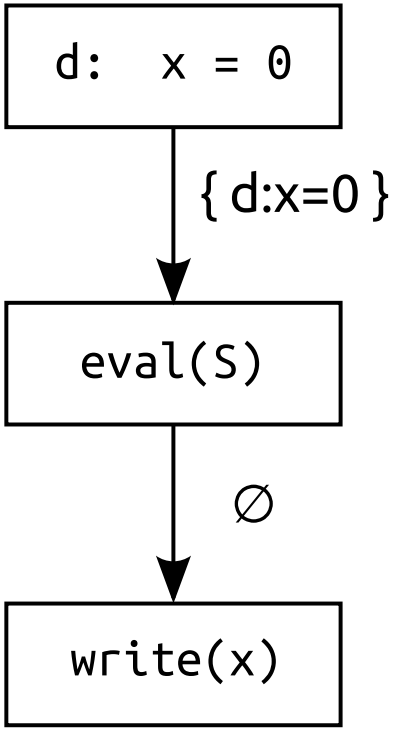
$out(S_i) = gen(S_i) \cup (in(S_i) - kill(S_i))$

where:

- $gen(S_i) = \{ d_i \}$ if S_i is a statement that defines x
- $kill(S_i) = \{ d_j \mid d_j \text{ defines } x \}$

What is *eval*?

Example: Reaching defs



What is *eval*?

The paper explores *how eval* is used in practice, and, hopefully, shows that we can replace some of *eval*'s usages with more structured constructs.

Methodology

Methodology

Infrastructure

TracingSafari: "records a trace containing most ops performed by the interpreter (reads, writes, deletes, calls, defines, etc.)"

Also records properties specific to *eval*: in particular the *provenance* of strings, since they could be used as an argument to *eval*.

Methodology

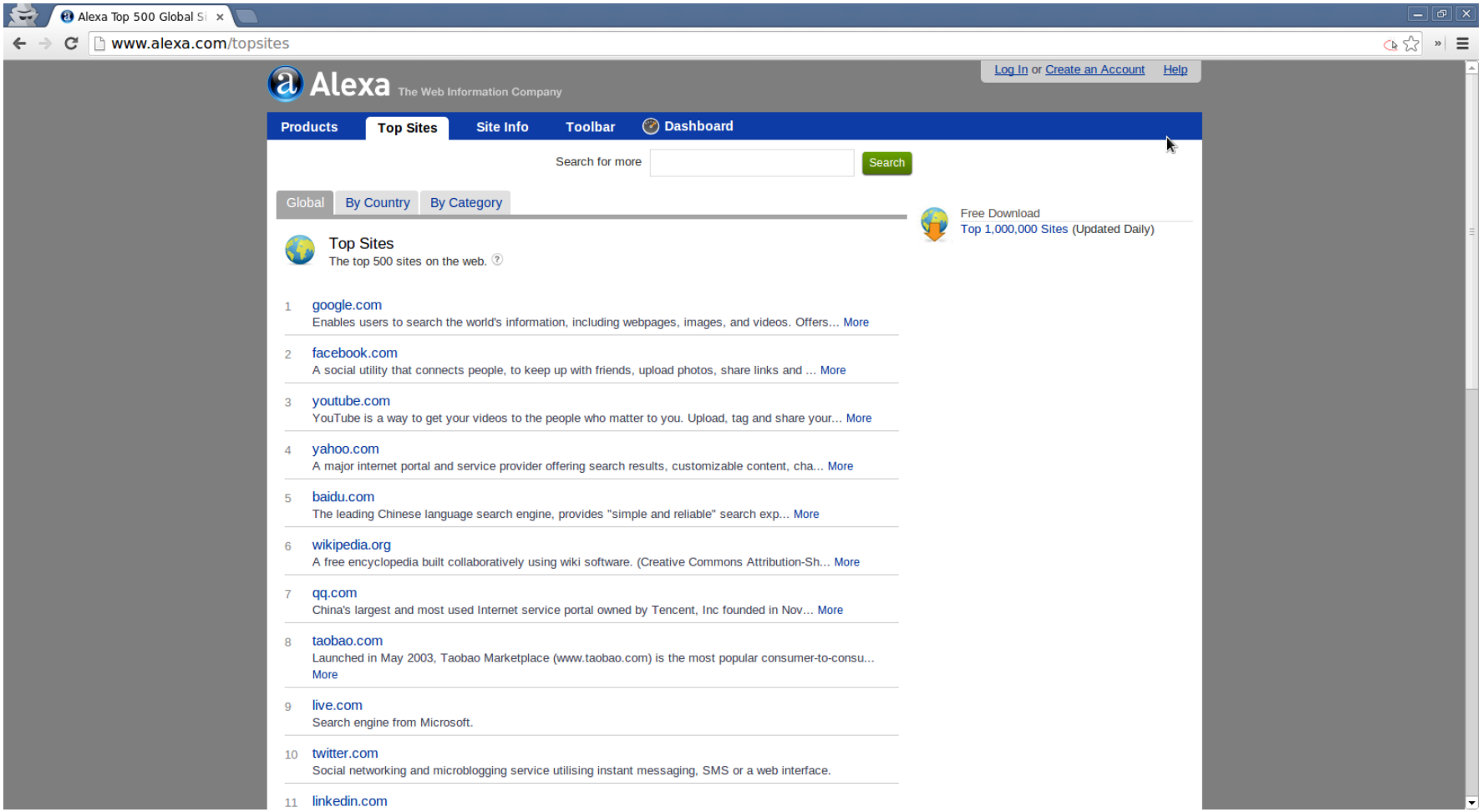
Corpus

Question: if you want to do any kind of research on the web, where do you go first?

Methodology

Corpus

Question: if you want to do any kind of research on the web, where do you go first?



Methodology

Corpus

Interactive	PageLoad	Random
Manual interaction with web sites	First 30 seconds of execution of a web page	PageLoad with randomly generated events
Top 100	Top 10,000	Top 10,000
∼1-5 minutes	30 seconds	At most 30 events, 1 ev/sec

Methodology

Threats to validity

- *Program coverage*: they believe their corpus is representative of typical web browsing, even if they miss some functionality.
- *Diversity*: web applications in JS vastly out-number any other type of application written in JS.

Results

Are JavaScript and *eval* even used?

Are JavaScript and *eval* even used?

Table 1. Eval usage statistics.

Data Set	JavaScript used	eval use	Avg eval (bytes)	Avg eval calls	total eval calls	total eval size (MB)	total JS size (MB)
INTERACTIVE	100%	82%	1,210	84	7,078	8.2	204
PAGELOAD	89%	50%	655	34	158,994	99.3	1,319
RANDOM	89%	52%	627	61	384,286	229.6	1,823

- All top 100 sites use JS and 82 of them use *eval*
- 90% of the top 10,000 use JS and 50% use *eval*
- Events trigger more calls to eval

What about JS frameworks?

What about JS frameworks?

Data Set	jQuery	Prototype	MooTools
INTERACTIVE	54%	11%	7%
PAGELOAD	53%	6%	4%
RANDOM	60%	7%	6%

- Manual inspection reveals that *eval* is not required for their operation
- Used mostly as a fallback for browsers lacking *JSON.parse*

Patterns of *eval*

Patterns of *eval*

- Many common patterns in the use of *eval*
- Some are accepted industry practices (e.g. JSON, async content and library loading)
- Many result from a poor understanding of JavaScript

Patterns of *eval*

JSON	A JSON string or variant.
JSONP	A padded JSON string.
Library	One or more function definitions.
Read	Read access to an object's property.
Assign	Assignment to a local variable or object property.
Typeof	Type test expression.
Try	Trivial try/catch block.
Call	Simple function/method call.
Empty	Empty or blank string.
Other	Uncategorized string.

Patterns of *eval*

JSON

```
m = eval('{"a": "foo", "b": [1,2,3]}');
```

Funny note: *JSON* was invented by Douglas Crockford, so that *eval* could be used to parse it.

Patterns of *eval*

JSONP

```
eval('m = {"a": "foo", "b": [1,2,3]}');

eval('f({"a": "foo", "b": [1,2,3]})');
```

- Used for load balancing across domains (work around the same origin policiy)

Patterns of *eval*

Library

Libraries loaded with <script> tag are downloaded, parsed and evaluated synchronously.

Workaround: download the library with AJAX, and load it with *eval*.

Detection heuristic: any *eval* string longer than 512 bytes and definining at least one function.

Patterns of *eval*

Read

Field accesses and pseudo arrays.

```
eval("foo." + x)  // foo[x]

eval("arr_" + 3)
```

An alias to *eval* can also be used to access a shadowed variable.

Patterns of *eval*

Assign

Patterns similar to READ, but with assignments.

Patterns of *eval*

Typeof

Strange patterns involving *typeof*.

```
eval("typeof(x) === 'undefined'")
    // typeof(x) === 'undefined'
    // 'x' in window
```

Patterns of *eval*

Try

"Another case for which we have no satisfying explanation, labeled **Try**, is to eval try/catch blocks."

From bbc.co.uk:

```
eval('try{throw v=4}catch(e){}') // v = 4
```

Authors assume it's the result of a corner case of a code generator.

Patterns of *eval*

Call

Method invocations (typically, global functions strings) with parameters that are not padded JSON.

```
eval(meth+'(x)') // window[meth](x)
```

Patterns of *eval*

Empty

eval is called with empty string (or all blanks).

```
eval("")
```

Likely the default case for *eval* strings in a code generator.

Patterns of *eval*

Other

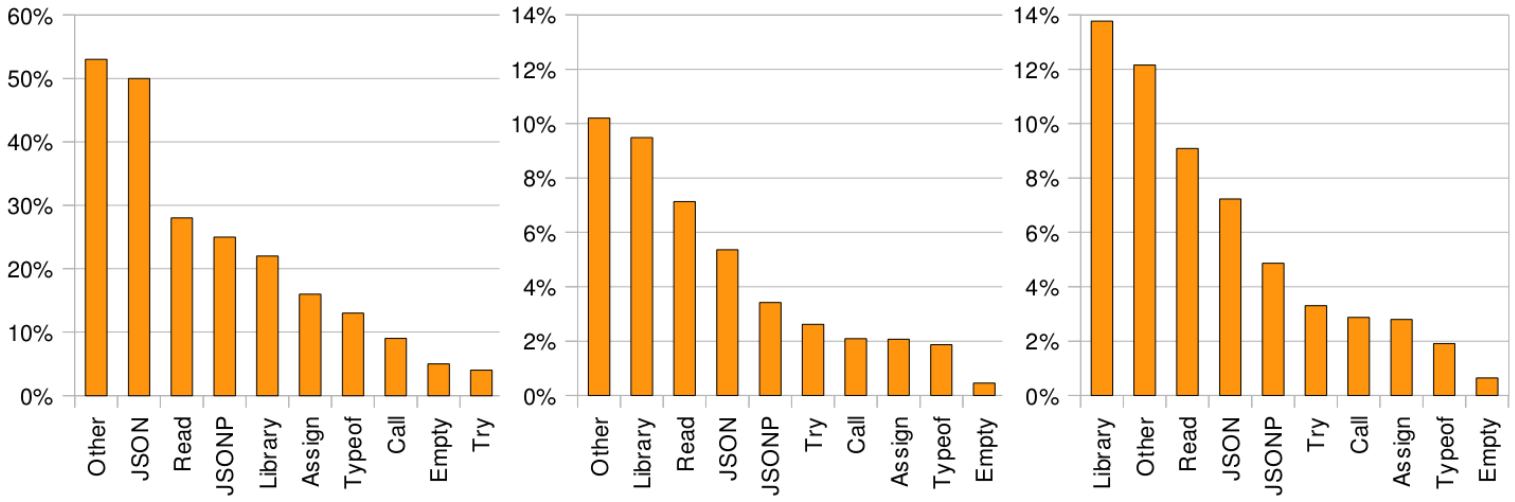
Patterns not captured by the previous categories.

```
eval("img1.src='http://f.ca/t.php?ip=xx';");
```

"Encodes data in a URL and sends an HTTP GET request in order to circumvent the same origin policy imposed by the DOM."

Patterns of *eval*

Patterns by websites

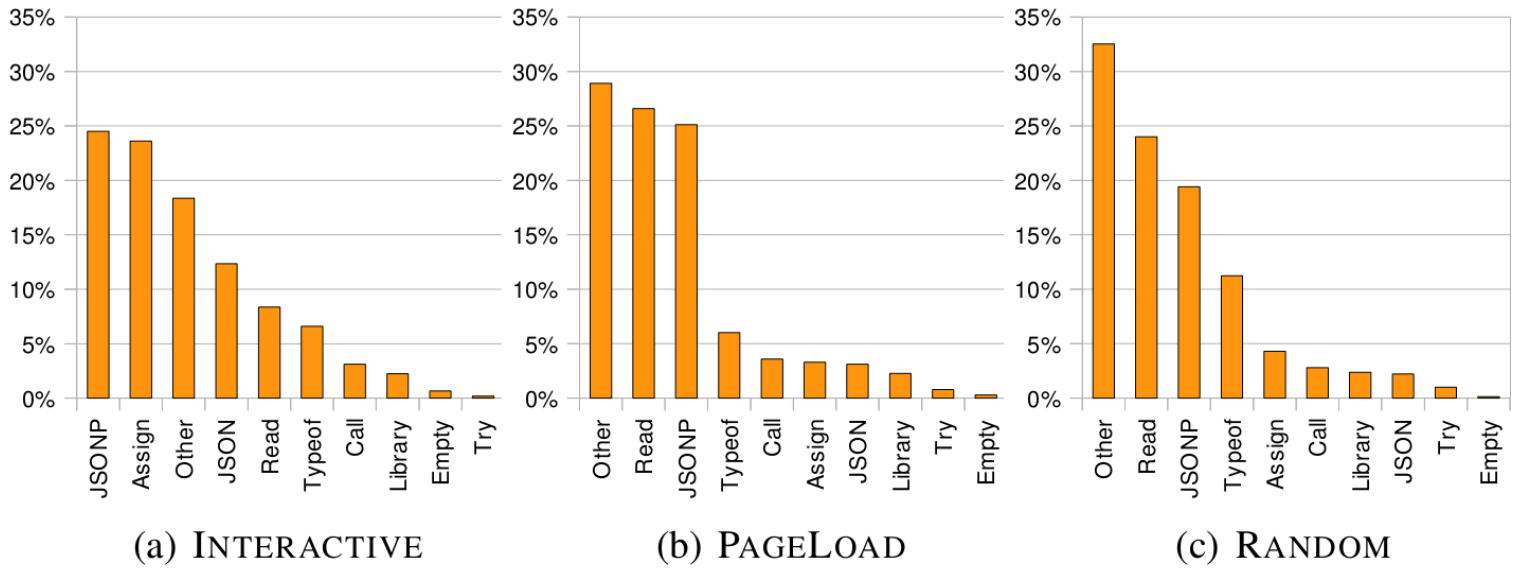


(a) INTERACTIVE (b) PAGELOAD (c) RANDOM

Fig. 8. Patterns by websites. Number of web sites in each data set with at least one eval argument in each category (a single web site can appear in multiple categories).

Patterns of *eval*

Patterns distribution



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

Fig. 9. Patterns. Ratio of evals in each category.

Patterns of *eval*

Impact on analysis

"Most *eval* call sites in categories other than **Library**, **Other**, **Call** are replaceable by less dynamic features such as *JSON.parse*, hashmap access, and proper use of JavaScript arrays. On INTERACTIVE, these categories account for 76% of all *eval*'d strings; thus, **a majority of *eval* uses are not necessary**."

Pattern replacements

Pattern replacements

JSON	JSON.parse(str)
JSONP	window[id] = JSON.parse(str) or window[id](JSON.parse(str))
Read	window[id] or window[id][propertyName]
Assign	window[id] = window[id] or window[id][propertyName]=window[id]
Typeof	typeof(window[id]) or id in window
Try	(Not trivially replaceable)
Call	window[id](window[id], ...) or window[id].apply(window, [...])
Empty	undefined or void 0

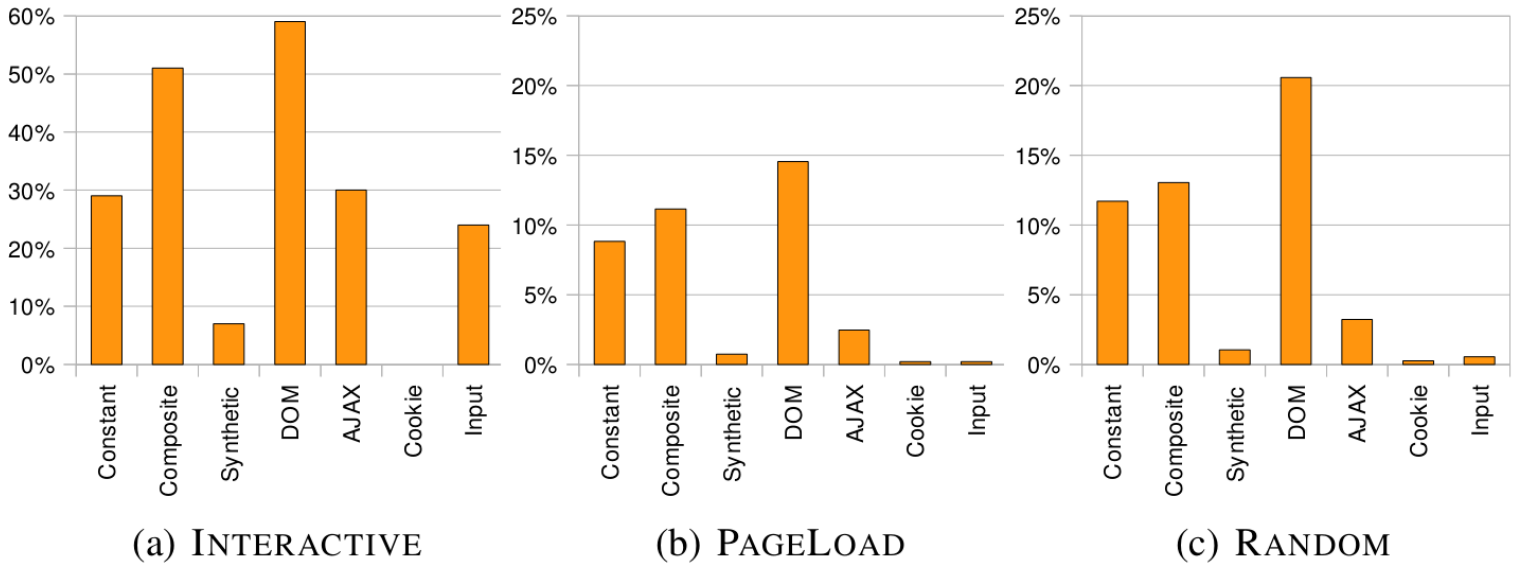
Provenance of *eval* strings

Provenance of strings

Where do the strings passed to *eval* come from? Authors used *TracingSafari* to track their provenance:

Constant	Strings that appear in the source code.
Composite	String constructed by concatenating constants and primitive values.
Synthetic	Strings that are constants in a nested eval.
DOM	Strings obtained from DOM or native calls.
AJAX	Strings that contain data retrieved from an AJAX call.
Cookies	Strings retrieved from a cookie or other persistent storage.
Input	Strings entered by a user into form elements.

Provenance of strings



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

Fig. 10. Provenance by websites. Percentage of web sites using a string of given provenance at least once in an eval expression.

Provenance of strings

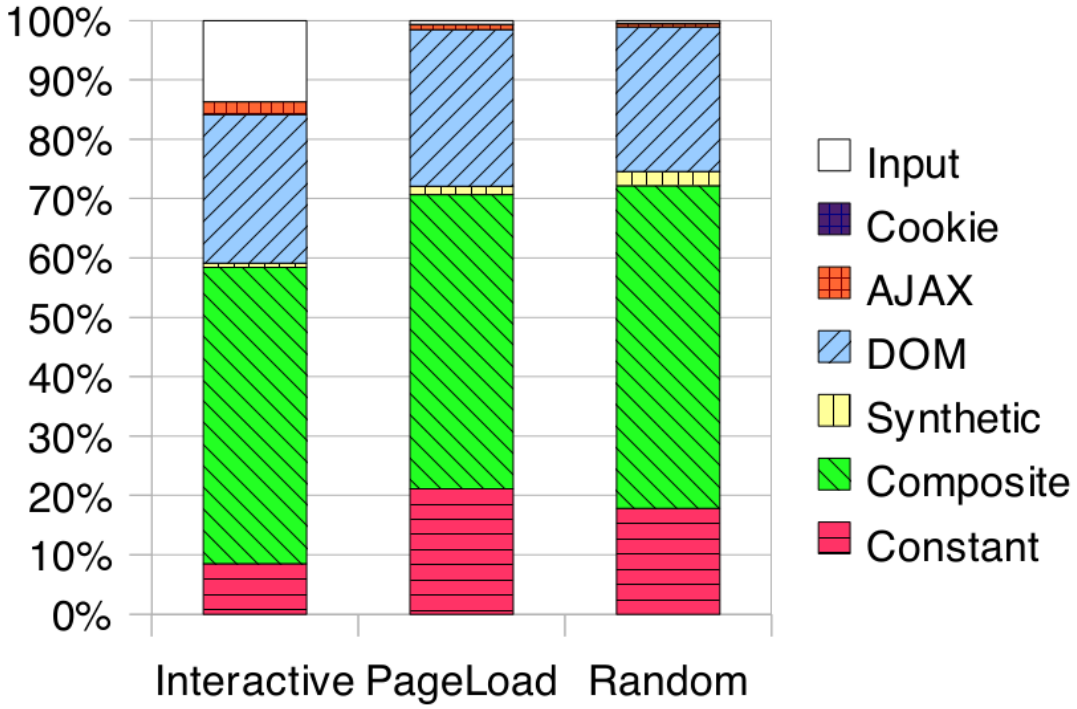
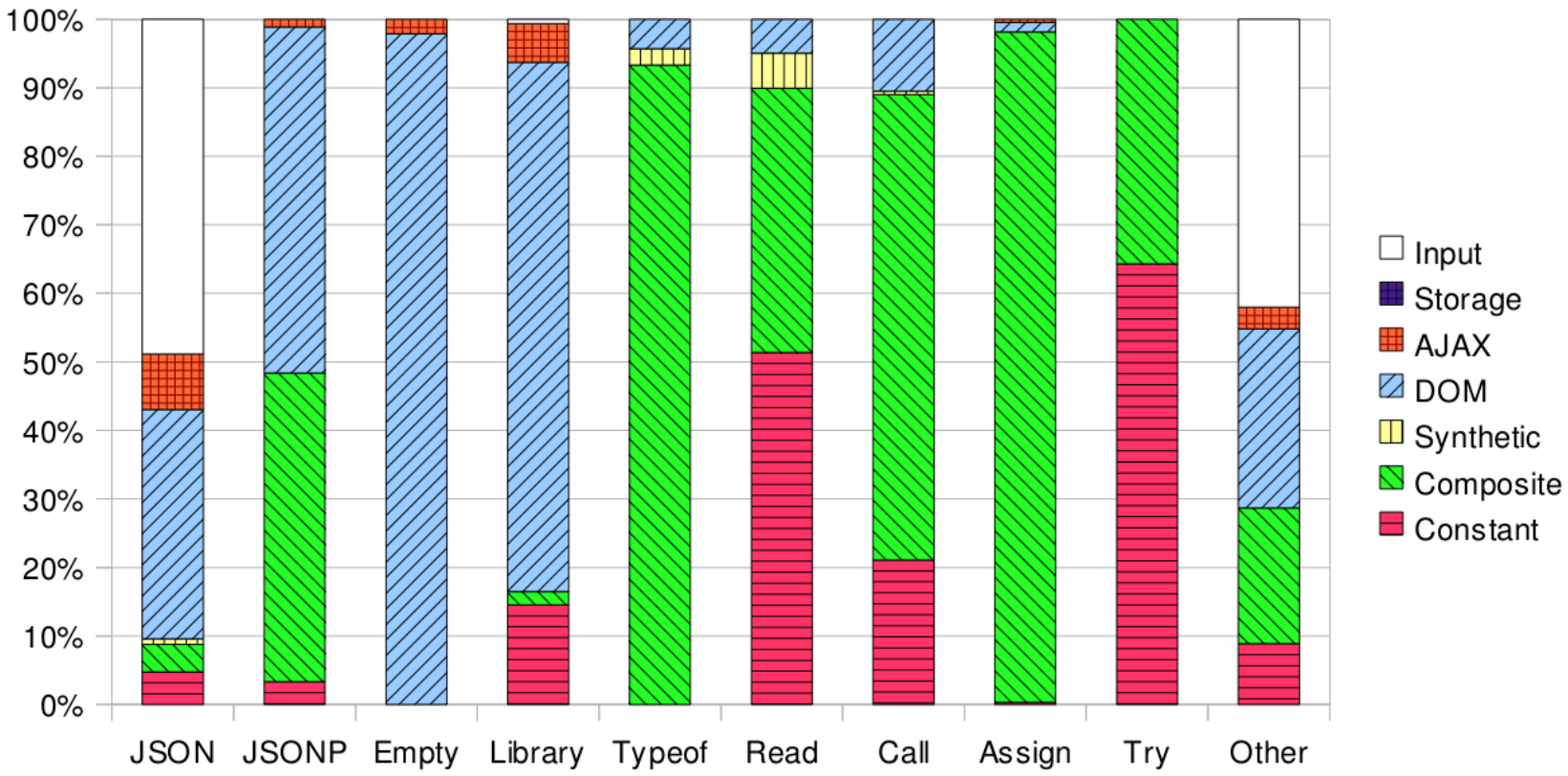


Fig. 11. Provenance. Proportion of strings with given provenance in eval'd strings for the three data sets.

Notice how many *eval*'ed strings are constant and composite!

Provenance of strings



(a) INTERACTIVE

Performance impact of *eval*

Performance impact of *eval*

```
function E() {
  eval(evalstr); x++;
  return x;
}
```

```
enter
init_lazy_reg r0
init_lazy_reg r2
init_lazy_reg r1
create_activation r0
resolve_with_base r4, r3,
                    eval(@id0)
resolve r5, evalstr(@id1)
call_eval r3, 2, 12
op_call_put_result r3
resolve_with_base r4, r3, x(@id2)
pre_inc  r3
put_by_id r4, x(@id2), r3
resolve r3, x(@id2)
tear_off_activation r0, r2
ret r3
```

```
function NoE() {
  id(evalstr);
  x++;
  return x;
}
```

```
enter
get_global_var r0, −8
mov r1, undefined(@k0)
get_global_var r2, −12
call r0, 2, 9
get_global_var r0, −11
pre_inc r0
put_global_var −11, r0
get_global_var r0, −11
ret r0
```

Fig. 17. Bytecode generated by WebKit.

Conclusion

Conclusion

"We started this work with the hope that it would show that *eval* can be replaced by other features. Unfortunately our data does not support this conclusion."

Conclusion

"*eval* is a convenient way of providing a range of features that weren't planned for by the language designers. For example, JSON was created to support (de-)serialization of JavaScript objects."

Conclusion

"Most accepted uses of *eval* have been transformed into libraries or new language features recently, and as such no best practices recommends usage of *eval*."

</presentation>

Big question

How would you design an analysis to identify *constant* and *composite* strings, so that you could offer suggestions to a programmer that his usage of eval is perhaps not necessary?