Jalangi: A Dynamic Analysis Framework for JavaScript

Manu Sridharan
Uber

Koushik Sen, Liang Gong
University of California, Berkeley

With contributions from:
Why JavaScript?

  - Based on projects hosted at GitHub and questions posted at StackOverflow
Why JavaScript?

Growth in popularity (based on jobs available) from 2012 – 2013

Source: [http://blog.learntoprogram.tv/five-resons-javascript-important-programming-language-learn/](http://blog.learntoprogram.tv/five-resons-javascript-important-programming-language-learn/)
Why JavaScript?

- Client-side JavaScript in Rich Web Applications
- Desktop Apps (Windows 8 and Gnome), Firefox OS, Tizen OS
- Server-side (node.js)
  - Paypal, Ebay, Uber, NYtimes, Linkedin, and many more
- Assembly Language for the Web: emscripten, coffeescript, TypeScript
- A language to implement DSL frameworks
  - Angular.js, Knockout.js, React.js
Why JavaScript?

• Huge ecosystem of libraries and frameworks
• JavaScript has low learning curve
  – people can start coding and get results quickly
• No special installation/execution environment
  – Just use a modern browser
• JavaScript supports functional programming
  – higher order functions
• Modern JavaScript VMs are fast
Atwood’s Law

“Any application that can be written in JavaScript, will eventually be written in JavaScript.”
Why Tools for JavaScript?

• JavaScript has its quirks (many)
Why Tools for JavaScript?

```javascript
var x = "1";
++x;
console.log(x);

var x = "1";
x += 1;
console.log(x);
```
Why Tools for JavaScript?

```javascript
var x = "1";
++x;
console.log(x);
// prints 2

var x = "1";
x += 1;
console.log(x);
// prints 11
```
Why Tools for JavaScript?

• Easy to introduce bugs: correctness, performance, memory
  – Degrees of equality == vs. ===

• Loosely-typed
  – forgiving: implicit type conversion
  – tries hard to execute without throwing exception
    • Like HTML

• Highly reflective
  – eval any dynamically created string

• Asynchronous programming
• Loosely-typed
  – forgiving: implicit type conversion
  – tries hard to execute without throwing exception
    • Like HTML
Tools for Bug Finding and Security Analysis

• Remarkable progress in program-analysis and constraint solving
  – Commercial tools: Coverity, Klocwork, Grammatech, TotalView, Paralloticy, Static Device Verifier from Microsoft, WALA at IBM
  – Open-source tools: GDB, lint, FindBugs, Valgrind
  – Academic tools: SLAM, BLAST, ESP, JPF, Bandera, Saturn, MAGIC, DART, CUTE, jCUTE
  – Mostly focused on C/C++ and Java programs
• Hardly any software quality tool for JavaScript and HTML5
  – Static analysis is difficult for dynamic languages
Jalangi

A powerful browser-independent (dynamic) analysis framework for JavaScript

https://github.com/Samsung/jalangi2

Jalangi: Goals and Requirements

• Framework for Dynamic and hybrid Static/Dynamic analysis
  – supports symbolic execution, bug finding, memory analysis, runtime type analysis, value tracking, taint tracking, performance analysis

• Handle ALL dynamic features
  – not OK to ignore eval, new Function

• Independent of browser
  – source-to-source code instrumentation
  – instrumented program when executed performs analysis

• Easy Implementation of Dynamic Analysis
  – Observe an execution passively: (conventional dynamic analysis)
  – Modify semantics/values
  – Repeatedly execute arbitrary paths within a function
Why not Modify a Browser?

- Hard to keep up with browser development
- Harder to get people to use of customized browser

Excluding merges, 40 authors have pushed 619 commits to master and 668 commits to all branches. On master, 840 files have changed and there have been 74,658 additions and 40,901 deletions.
Jalangi 1 and 2

• Jalangi 1:
  – [https://github.com/SRA-SiliconValley/jalangi](https://github.com/SRA-SiliconValley/jalangi)
  – record execution and replay to perform analysis
  – Shadow values (wrapped objects)
  – No longer supported

• Jalangi 2:
  – [https://github.com/Samsung/jalangi2](https://github.com/Samsung/jalangi2)
  – no record/replay or shadow values
    – optional shadow memory
  – active development
How Jalangi Works?

JavaScript and HTML

Jalangi Runtime

User Written Analysis
How Jalangi Works?

- JavaScript and HTML
- Jalangi Instrumentor
- Instrumented Files
- Jalangi Runtime
- Source Information
- User Written Analysis

Jalangi Analysis Writer

Intermediate
How Jalangi Works?

JavaScript and HTML

Jalangi Instrumentor

Jalangi Instrumented Files

Jalangi Runtime

Source Information

User Written Analysis

Execute in Browser/Node.js

Trace

Output Data

Jalangi

Analysis Writer

Intermediate
How Jalangi Works?

1. **JavaScript and HTML**
2. **Jalangi Instrumentor**
3. **Instrumented Files**
4. **Jalangi Runtime**
5. **Source Information**
6. **Execute in Browser/Node.js**
7. **Trace**
8. **User Written Analysis**
9. **Output Data**
10. **Offline Analysis**
11. **Visualize Output**
12. **Final Output**

**Jalangi**

**Analysis Writer**

**Intermediate**
Jalangi Instrumentation (simplified)

\[ x = y + 1 \quad \Rightarrow \quad x = \text{Write}("x", \text{Binary}(\'+', \text{Read}("y", y), \text{Literal}(1), x) \]

\[ a.f = b.g \quad \Rightarrow \quad \text{PutField}(\text{Read}("a", a), "f", \text{GetField}(\text{Read}("b", b), "g")) \]

\[ \text{if } (a.f()) \ldots \quad \Rightarrow \quad \text{if } (\text{Branch}(\text{Method}(\text{Read}("a", a), "f"))()) \ldots \]
Jalangi Runtime

function Binary(op, left, right, ...) {

    result = left op right;

    return result;
}

function Binary(op, left, right, ...) {

    var aret = analysis.binaryPre(op, left, write, ...);

    result = left op right;
    aret = analysis.binary(op, left, right, result, ...);

    return result;
}
function Binary(op, left, right, ...) {
    var skip = false;
    var aret = analysis.binaryPre(op, left, write, ...);
    if (aret) {
        op = aret.op;
        left = aret.left;
        right = aret.right;
        skip = aret.skip;
    }
    if (!skip)
        result = left op right;
    aret = analysis.binary(op, left, right, result, ...);
    return result;
}
Jalangi Runtime

function Binary(op, left, right, ...) {
    var skip = false;
    var aret = analysis.binaryPre(op, left, write, ...);
    if (aret) {
        op = aret.op;
        left = aret.left;
        right = aret.right;
        skip = aret.skip; }
    if (!skip)
        result = left op right;
    aret = analysis.binary(op, left, right, result, ...);
    if (aret)
        return aret.result;
    else
        return result;
}
Download and Install Jalangi 2

**Download:**

git clone https://github.com/Samsung/jalangi2.git
cd jalangi2

**Install:**
npm install

**Test:**

python scripts/test.traceall.py
python scripts/test.analysis.py
python scripts/test.dlint.py
Jalangi Callbacks

Documentation: jalangi2/docs/MyAnalysis.html

- Each analysis needs to implement a subset of these callbacks.
- Multiple analyses classes can be chained

```javascript
function invokeFunPre (iid, f, base, args, isConstructor, isMethod, functionIid);  
function invokeFun (iid, f, base, args, result, isConstructor, isMethod, functionIid);  
function literal (iid, val, hasGetterSetter);  
function forinObject (iid, val);  
function declare (iid, name, val, isArgument, argumentIndex, isCatchParam);  
function getFieldPre (iid, base, offset, isComputed, isOpAssign, isMethodCall);  
function getField (iid, base, offset, val, isComputed, isOpAssign, isMethodCall);  
function putFieldPre (iid, base, offset, val, isComputed, isOpAssign);  
function putField (iid, base, offset, val, isComputed, isOpAssign);  
function read (iid, name, val, isGlobal, isScriptLocal);  
function write (iid, name, val, lhs, isGlobal, isScriptLocal);  
function _return (iid, val);  
function _throw (iid, val);  
function _with (iid, val);  

function functionEnter (iid, f, dis, args);  
function functionExit (iid, returnVal, wrappedExceptionVal);  
function scriptEnter (iid, instrumentedFileName, originalFileName);  
function scriptExit (iid, wrappedExceptionVal);  
function binaryPre (iid, op, left, right, isOpAssign, isSwitchCaseComparison, isComputed);  
function binary (iid, op, left, right, result, isOpAssign, isSwitchCaseComparison, isComputed);  
function unaryPre (iid, op, left);  
function unary (iid, op, left, result);  
function conditional (iid, result);  
function instrumentCodePre (iid, code);  
function instrumentCode (iid, newCode, newAst);  
function endExpression (iid);  
function endExecution();  
function runInstrumentedFunctionBody (iid, f, functionIid);  
function onReady (cb);  
```
TraceAll.js analysis: prints all callbacks

For Node.js

- `node src/js/commands/jalangi.js --inlineIID --inlineSource --analysis src/js/sample_analyses/ChainedAnalyses.js --analysis src/js/runtime/SMemory.js --analysis src/js/sample_analyses/pldi16/TraceAll.js tests/pldi16/TraceAllTest.js`

For browser:

- `node src/js/commands/esnstrument_cli.js --inlineIID --inlineSource --analysis src/js/sample_analyses/ChainedAnalyses.js --analysis src/js/runtime/SMemory.js --analysis src/js/sample_analyses/pldi16/TraceAll.js --out /tmp/pldi16/TraceAllTest.html tests/pldi16/TraceAllTest.html`
  
- `open file:///tmp/pldi16/TraceAllTest.html`
Sample Analyses

Examples: src/js/sample_analyses/pldi16
Tests: tests/pldi16
Sample analysis:
check if undefined is concatenated with a string

See: src/js/sample_analyses/pldi16/CheckUndefinedConcatenatedToString.js

```javascript
this.binary = function(iid, op, left, right, result){
    if (op === '+' && typeof result==='string' &&
        (left===undefined || right===undefined))
        J$.log("Concatenated undefined with string at "+
                J$.iidToLocation(J$.sid, iid));
}
```
Source Locations

• Instrumentation associates an iid with every expression
• At runtime, each loaded script is given a unique script ID (sid)
• sid of current script stored in J$.sid
• J$.getGlobalIID(iid) gets a globally unique id
• J$.iidToLocation(J$.sid, iid) gets source location
  • filename:start_line:start_col:end_line:end_col
• Tracks locations of enclosing evals
Sample analysis: count branches

See: src/js/sample_analyses/pldi16/BranchCoverage.js

```javascript
var trueBranches = {};
var falseBranches = {};
// initialize ....

this.conditional = function(iid, result) {
    var id = J$.getGlobalIID(iid);
    if (result)
        trueBranches[id]++;
    else
        falseBranches[id]++;
}

this.endExecution = function () {
    print(trueBranches, "True");
    print(falseBranches, "False");
}

function print(map, str) {
    for (var id in map)
        if (map.hasOwnProperty(id)){
            J$.log(str+ " branch taken at " +
            J$.iidToLocation(id)+ " " +map[id] +
            " times");
        }
}
```
Sample analysis:
count number of objects allocated at each site

See: src/js/sample_analyses/pldi16/CountObjectsPerAllocationSite.js

```javascript
var allocCount = {}; this.literal = function (iid, val) {
    var id = J$.getGlobalIID(iid);
    if (typeof val === 'object')
        allocCount[id]++;
}; this.invokeFunPre = function (iid, f, base, args, isConstructor) {
    var id = J$.getGlobalIID(iid);
    if (isConstructor)
        allocCount[id]++;
};
this.endExecution = function () {
    print(allocCount);
};
function print(map) {
    for (var id in map)
        if (map.hasOwnProperty(id)) {
            J$.log("Object allocated at " +
                J$.iidToLocation(id) + "=" + map[id]);
        }
}
```

See: src/js/sample_analyses/pldi16/CountObjectsPerAllocationSite.js
Shadow Objects (SMemory.js)

• Associates a shadow object with each JavaScript object (excludes primitive values including strings and null)
• Associates a shadow object with each activation frame
• Shadow object can store meta-information
• A shadow object contains an unique id
  – can be used as logical address of an object/frame

--analysis src/js/sample_analyses/ChainedAnalyses.js --analysis src/js/runtime/SMemory.js
SMemory.js API

Documentation: jalangi2/docs/SMemory.html

- `getShadowObject(obj, prop, isGetField)`
  This method should be called on a base object and a property name to retrieve the shadow object associated with the object that actually owns the property.

- `getShadowObjectOfObject(val)`
  This method returns the shadow object associated with the argument. If the argument cannot be associated with a shadow object, the function returns undefined.

- `getShadowFrame(name)`
  This method returns the shadow object associated with the activation frame that contains the variable "name". To get the current activation frame's shadow object, call `getShadowFrame('this')`.

- `getIDFromShadowObjectOrFrame(obj)`
  Given a shadow object or frame, it returns the unique id of the shadow object or frame. It returns undefined, if obj is undefined, null, or not a valid shadow object.

- `getActualObjectOrFunctionFromShadowObjectOrFrame(obj)`
  Given a shadow object/frame, it returns the actual object/the function whose invocation created the frame.
**Associate Allocation Site**

*See:* src/js/sample_analyses/pldi16/LogLoadStoreAlloc.js

```javascript
this.literal = function (iid, val, hasGetterSetter) {
    if (typeof val === "object" && val !== null) {
        var sobj = sandbox.smemory.getShadowObjectOfObject(val);
        sobj.allocSite = J$.iidToLocation(J$.sid, iid);
    }
};

this.getFieldPre = function (iid, base, offset, isComputed, isOpAssign, isMethodCall) {
    var sobj = sandbox.smemory.getShadowObject(base, offset, true).owner;
    var ret = "Load " + offset + " of object allocated at" + sobj.allocSite;
    ret += " at " + J$.iidToLocation(J$.sid, iid);
    log(ret);
};
```
Log All Loads and Stores

See: src/js/sample_analyses/pldi16/LogLoadStoreAlloc.js

this.getFieldPre = function (iid, base, offset, isComputed, isOpAssign, isMethodCall) {
    var sobj = sandbox.smemory.getShadowObject(base, offset, true).owner;
    var actualObjectId = sandbox.smemory.getIDFromShadowObjectOrFrame(sobj);
    var ret = "Load of object(id=" + actualObjectId + ")." + offset;
    ret += " at " + J$.iidToLocation(J$.sid, iid);
    log(ret);
};

this.write = function (iid, name, val, lhs, isGlobal, isScriptLocal) {
    var sobj = sandbox.smemory.getShadowFrame(name);
    var frameId = sandbox.smemory.getIDFromShadowObjectOrFrame(sobj);
    var ret = "Store of frame(id=" + frameId + ")." + name;
    ret += " at " + J$.iidToLocation(J$.sid, iid);
    log(ret);
    return {result: val};
};
Sample analysis (modify semantics):
interpret ‘*’ as ‘+’

See: src/js/sample_analyses/pldi16/ChangeSemanticsOfMult.js

```javascript
this.binaryPre = function (iid, op, left, right) {
    if (op === '*')
        return {op: op, left: left, right: right, skip: true};
};

this.binary = function (iid, op, left, right, result) {
    if (op === '*')
        return {result: left + right};
};
```
Sample analysis (modify semantics): skip execution of an evil function

See: src/js/sample_analyses/pldi16/SkipFunction.js

```javascript
this.invokeFunPre = function (iid, f, base, args) {
    if (typeof evilFunction === "function" && f === evilFunction) {
        return {f: f, base: base, args: args, skip: true};
    }
};
```
Sample analysis (modify semantics): loop a function body

See: src/js/sample_analyses/pldi16/BackTrackLoop.js

```javascript
function loop(n) {
    var ret = ret? ret-1: n;
    // do something
    console.log(ret);
    return ret;
}
loop(10);
```
Sample analysis (modify semantics): loop a function body

See: src/js/sample_analyses/pldi16/BackTrackLoop.js

```javascript
function loop(n) {
  var ret = ret? ret-1: n;
  // do something
  console.log(ret);
  return ret;
}
loop(10);
```
Sample analysis (modify semantics):
loop a function body

See: src/js/sample_analyses/pldi16/BackTrackLoop.js

this.functionExit = function (iid, rv, ex) {
  return {returnVal: rv, wrappedExceptionVal: ex, isBacktrack: rv?true:false};
};

---------------------------------- Program ------------------------------------

function loop(n) {
  var ret = ret? ret-1: n;
  // do something
  console.log(ret);
  return ret;
}
loop(10);

Prints 10 to 0
Sample analysis (modify semantics):
MultiSE: Multi-Path Symbolic Execution using Value Summaries (ESEC/FSE 2015)

- Symbolic execution
- Explore all paths in a function
  - but merge state from all paths before exiting the function
- Override default semantics to perform symbolic evaluation
- Backtrack within a function until all paths are explored
- Custom semantics and backtracking
  - for simple abstract interpretation
  - for simple dataflow analysis
Jalangi 2 Summary

- Observe an execution and collect information
- Change values used in an execution
- Change semantics of operators/functions
- Explore arbitrary path in a function
- Re-execute the body of a function repeatedly
- Maintain your own (abstract) state and call stack
- 3x-100x slowdown
Serious Analyses with Jalangi

- "Feedback-Directed Instrumentation for Deployed JavaScript Applications,"
  - Magnus Madsen and Frank Tip and Esben Andreasen and Koushik Sen and Anders Moller (ICSE'16)

- "Trace Typing: An Approach for Evaluating Retrofitted Type Systems,"
  - Esben Andreasen and Colin S. Gordon and Satish Chandra and Manu Sridharan and Frank Tip and Koushik Sen (ECOOP'16)

- "TypeDevil: Dynamic Type Inconsistency Analysis for JavaScript,"
  - Michael Pradel and Parker Schuh and Koushik Sen (ICSE'15)

- "JITProf: Pinpointing JIT-unfriendly JavaScript Code,"
  - Liang Gong and Michael Pradel and Koushik Sen (ESEC/FSE'15)

- "MemInsight: Platform-Independent Memory Debugging for JavaScript,"
  - Simon Jensen and Manu Sridharan and Koushik Sen and Satish Chandra (ESEC/FSE'15)

- "DLint: Dynamically Checking Bad Coding Practices in JavaScript,"
  - Liang Gong and Michael Pradel and Manu Sridharan and Koushik Sen (ISSTA'15)

- "MultiSE: Multi-Path Symbolic Execution using Value Summaries,"
  - Koushik Sen and George Necula and Liang Gong and Wontae Choi, (ESEC/FSE'15)

- "The Good, the Bad, and the Ugly: An Empirical Study of Implicit Type Conversions in JavaScript,"
  - Michael Pradel and Koushik Sen (ECOOP'15)

- "EventBreak: Analyzing the Responsiveness of User Interfaces through Performance-Guided Test Generation,"
  - Michael Pradel and Parker Schuh and George Necula and Koushik Sen (OOPSLA’14)
Serious Analyses with Jalangi

- "Feedback-Directed Instrumentation for Deployed JavaScript Applications,"
  Magnus Madsen and Frank Tip and Esben Andreasen and Koushik Sen and Anders (ICSE’16)

- "Trace Typing: An Approach for Evaluating Retrofitted Type Systems,"
  Esben Andreasen and Colin S. Gordon and Satish Chandra and Manu Sridharan and Frank Tip and Koushik Sen (ECOOP’16)

- "TypeDevil: Dynamic Type Inconsistency Analysis for JavaScript,"
  Michael Pradel and Parker Schuh and Koushik Sen (ICSE’15)

- "JITProf: Pinpointing JIT-unfriendly JavaScript Code,"
  Liang Gong and Michael Pradel and Koushik Sen (ESEC/FSE’15)

- "MemInsight: Platform-Independent Memory Debugging for JavaScript,"
  Simon Jensen and Manu Sridharan and Koushik Sen and Satish Chandra (ESEC/FSE’15)

- "DLint: Dynamically Checking Bad Coding Practices in JavaScript,"
  Liang Gong and Michael Pradel and Manu Sridharan and Koushik Sen (ISSTA’15)

- "MultiSE: Multi-Path Symbolic Execution using Value Summaries,"
  Koushik Sen and George Necula and Liang Gong and Wontae Choi,  (ESEC/FSE’15)

- "The Good, the Bad, and the Ugly: An Empirical Study of Implicit Type Conversions in JavaScript,"
  Michael Pradel and Koushik Sen (ECOOP’15)

- "EventBreak: Analyzing the Responsiveness of User Interfaces through Performance-Guided Test Generation,"
  Michael Pradel and Parker Schuh and George Necula and Koushik Sen (OOPSLA’14)
MemInsight
Platform-Independent Memory Debugging for JavaScript

http://github.com/Samsung/meminsight
JS Apps and Memory

BloatBusters: Eliminating memory leaks in Gmail

Node.js Performance Tip of the Week: Memory Leak Diagnosis
Leaks and Staleness

• **Staleness**: long gap between last use and unreachable

• **Leak**: never unreachable

• Many stale objects indicates a potential problem
Leak Example

```javascript
var name2obj = {};  
var cache = [];

function add(name) {
    var x = new Obj();
    name2obj[name] = x;
    cache.push(x);
}

function remove(name) {
    name2obj[name] = null;
    // forgot to remove from the cache!
}
```

More insidious in web apps, where DOM nodes are involved
Churn

```javascript
if (this.canRevert([ni, nj], color, board) &&
    !this.isContain([ni, nj], ret)) {
    ret.push([parseInt(ni), parseInt(nj)]);
}

function canRevert(place, color, _board) {
    var i = parseInt(place[0]);
    var j = parseInt(place[1]);
    // no further usage of the place array
}

function isContain(place, _array) {
    ... uses place[0] and place[1] ...
},
```
Bloat

```plaintext
return {
    type: type,
    value: id,
    lineNumber: lineNumber,
    lineStart: lineStart,
    range: [start, index]
};
```

```plaintext
return {
    type: type,
    value: id,
    lineNumber: lineNumber,
    lineStart: lineStart,
    start: start,
    end: index
};
```
Heap Snapshots

Chrome Dev Tools

https://developers.google.com/chrome-developer-tools/docs/javascript-memory-profiling
Heap Snapshots

- Capture several snapshots, diff to find possible leaks

- Low overhead, but:
  - No information on staleness (does not track uses)
  - Can miss excessive churn
  - Cannot handle fine-grained time-varying properties
MemInsight

- **Platform independent**: use on any modern browser or node.js

- **Fine-grained behaviors via detailed tracing**
  - computes *exact object lifetimes*
  - enables a variety of client analyses

- **Exposes DOM manipulation**

- **Reasonable overhead**
```javascript
function f() {
    var newDiv = s('<div/>');
    newDiv.html("Hello world");
    newDiv.click(function () {
        newDiv.css("backgroundColor", "red");
    });
    newDiv.appendTo('#contents');
}

function g() {
    document.getElementById('contents').innerHTML = '';
}
```

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>jQuery handler leak</title>
</head>
<body>
    <div id="contents"></div>
    <script src="js/jquery-2.0.3.js"></script>
    <script src="js/scri.js"></script>
    <button onclick="f()">Add</button>
    <button onclick="g()">Remove All</button>
</body>
</html>
```
Memory leak!
jQuery issue!

```javascript
function f() {
    var newDiv = $('<div/>');
    newDiv.html("Hello world");
    newDiv.click(function () {
        newDiv.css("backgroundColor", "red");
    });
    newDiv.appendTo('#contents');
}

function g() {
    //document.getElementById('contents').innerHTML = '';
    $('#contents').empty();
}
```
Memory leak - Details

Allocation at `js/jquery-2.0.3_orig.js`, line 490

```
if (data) {
    return null;
}
if (typeof context === "boolean") {
    keepScripts = context;
    context = false;
}
context = context || document;
var passed = singleTag.wow; data },
scripts = keepScripts && []; // Single tag
if (passed) {
    return [context.createElement(">").call() ];
}
parsed = jQuery.buildFragment( [ data ], context, scripts )
if (scripts) {
```

Call Tree

```
```

Timeline

Access Paths

Click on any time point to show retaining access paths.
Challenges

• Prefer not to modify a browser engine
  ○ Yet handle full JavaScript
  ○ Keep overhead reasonable

• Want to report staleness of DOM nodes, without modifying browser

• Figure out object lifetimes accurately without information from the garbage collector
How does MemInsight work?

Jalangi is a dynamic analysis framework for JavaScript
See FSE 2013, Sen et al.
Trace generation

```javascript
var x = {};
var y = {};
function m(p,q)
{
    p.f = q;
}
m(x,y);
x = null;
```

```plaintext
DECLARE x,y,m;
ALLOCOBJ 2 at 1;
WRITE x,2 at 1;
ALLOCOBJ 3 at 2;
WRITE y,3 at 2;
ALLOCFUN 4 at 3;
WRITE m,4 at 3;
CALL 4 at 7;
DECLARE p = 2,
    q = 3;
PUTFIELD 2,"f",3
    at 5;
LASTUSE 2 at 5;
RETURN at 7;
LASTUSE 4 at 7;
WRITE x,0 at 8;
UNREACHABLE
    2 at 8;
UNREACHABLE
    3 at end;
UNREACHABLE
    4 at end;
```
writes when the old and new values are both primitive, as
To enable client analyses like leak detection, we require that
JavaScript language to Section
physical object sizes in general; see "Limitations" in Section
Most entries includes a source location at the end. The allo-
for computing object staleness.
prohibitively expensive, and last use information is sufficient
use of each object since we found that logging all uses was
for function objects, when it is invoked. We only log the last
each object, where an object is used when it is dereferenced or,
scope handling. For leak detection, we also log the
and also logs declarations of local variables to enable proper
such writes are irrelevant to a memory analysis. A

Figure 6: A simple code example and the corresponding trace.

Trace generation

```
1  var x = {};
2  var y = {};
3  function m(p,q)
4  {
5    p.f = q;
6  };
7  m(x,y);
8  x = null;
```

 DECLARE x,y,m;
 ALLOCObj 2 at 1;
 WRITE x, 2 at 1;
 ALLOCObj 3 at 2;
 WRITE y, 3 at 2;
 ALLOCFun 4 at 3;
 WRITE m, 4 at 3;
 CALL 4 at 7;
 DECLARE p = 2,
 q = 3;
 PUTFIELD 2, "f", 3
 at 5;
 LASTUse 2 at 5;
 RETURN at 7;
 LASTUse 4 at 7;
 WRITE x, 0 at 8;
 UNREACHABLE
 2 at 8;
 UNREACHABLE
 3 at end;
 UNREACHABLE
 4 at end;

Preserve line numbers
Trace generation

```javascript
1 var x = {};  
2 var y = {};  
3 function m(p, q)  
4 {  
5   p.f = q;  
6 }  
7 m(x, y);  
8 x = null;
```

```
DECLARE x, y, m;  
ALLOCOBJ 2 at 1;  
WRITE x, 2 at 1;  
ALLOCOBJ 3 at 2;  
WRITE y, 3 at 2;  
ALLOCFUN 4 at 3;  
WRITE m, 4 at 3;  
CALL 4 at 7;  
DECLARE p = 2,  
   q = 3;  
PUTFIELD 2, "f", 3  
at 5;  
LASTUSE 2 at 5;  
RETURN at 7;  
LASTUSE 4 at 7;  
WRITE x, 0 at 8;  
UNREACHABLE  
   2 at 8;  
UNREACHABLE  
   3 at end;  
UNREACHABLE  
   4 at end;
```

Preserve call stack
Trace generation

```javascript
1 var x = {};
2 var y = {};
3 function m(p, q) {
4     p.f = q;
5 };
6 m(x, y);
7 x = null;
```

```plaintext
DECLARE x, y, m;  DECLARE p = 2, q = 3;
ALLOCObj 2 at 1; PUTFIELD 2, "f", 3
WRITE x, 2 at 1; at 5;
WRITE y, 3 at 2;  
ALLOCObj 3 at 2; UNREACHABLE 2 at 8;
WRITE m, 4 at 3; UNREACHABLE 3 at end;
CALL 4 at 7; UNREACHABLE 4 at end;
DECLARE p = 2, UNREACHABLE
q = 3;
RETURN at 7;
LASTUse 2 at 5;
LASTUse 4 at 7;
WRITE x, 0 at 8;
UNREACHABLE 2 at 8;
```

Only last use
Trace generation

```javascript
1 var x = {}; 
2 var y = {}; 
3 function m(p,q) 
4 { 
5   p.f = q; 
6 } ;
7 m(x,y); 
8 x = null;
```

 DECLARE x, y, m; 
 ALLOCObj 2 at 1; 
 WRITE x, 2 at 1; 
 ALLOCObj 3 at 2; 
 WRITE y, 3 at 2; 
 ALLOCFun 4 at 3; 
 WRITE m, 4 at 3; 
 CALL 4 at 7; 
 DECLARE p = 2, 
 q = 3; 
 PUTFIELD 2, "f", 3 at 5; 
 LASTUse 2 at 5; 
 RETURN at 7; 
 LASTUse 4 at 7; 
 WRITE x, 0 at 8; 
 UNREACHABLE 2 at 8; 
 UNREACHABLE 3 at end; 
 UNREACHABLE 4 at end;

From lifetime analysis
Object lifetimes

• From trace, model runtime heap
  • Including call stack and closures

• Reference counting to compute unreachability time
  • Handle cycles with Merlin algorithm [Hertz et al. ASPLOS’06]

• Insert unreachability times in the enhanced trace
DOM Challenges

- DOM: tree data structure representing rendered HTML
  - Often involved in web app memory leaks

- Many manipulations not directly visible to JavaScript

```javascript
// allocates new div element
var elem = document.createElement("div");

// allocates DOM tree from HTML string and
// updates children of elem
elem.innerHTML = "<p><h1>Hello World!</h1></p>";

// inserts elem into global DOM
document.getElementById("x").appendChild(elem);
```
Our DOM Handling

```javascript
// allocates new div element
var elem = document.createElement("div");

// allocates DOM tree from HTML string and
// updates children of elem
elem.innerHTML = "<p><h1>Hello World!</h1></p>";

// inserts elem into global DOM
document.getElementById("x").appendChild(elem);
```

- `elem` gets reified into a fresh object ID
  - no special handling of `createElement`

- For DOM manipulations, leverage HTML5 mutation observers
  - Provide asynchronous notifications of DOM mutation
  - Handles `innerHTML` manipulation and `appendChild`

- Additional handling of `innerHTML` for better source locations
Other tricky features

- **Constructors**: need to properly handle `this`, and get good source locations
- **Eval**: instrument on the fly
- **Getters / setters**: don’t treat calls as reads / writes
- Global object, prototypes, further native models, …
Clients built atop MemInsight

- **Leak detection:** increasing stale object count at idle points (empty call stack)
- **Non-escaping:** no object escapes allocating function
  - Leverages *execution index* [Xin et al. PLDI’08]
- **Inlineable:** objects consistently “owned” by objects from another site
- Many more are possible!
Case Studies
(see paper for details)

• Leaks
  • Fixed in one Tizen app shopping_list (patch accepted)
  • Confirmed existing patch fixes leak in DataTables
  • Leaks found by internal users in other apps

• Churn
  • Fixed in one Tizen app annex for 10% speedup (patch accepted)
  • 10X speedup for escodegen (patch accepted)

• Bloat: Found object inlining opportunity in old esprima version (since fixed)
Leak in Shopping List app

```javascript
if (self.currentView.resetListOfLists) {
    ShoppingListApp.listoflists.innerHTML = "";
}
```

Should have used `.empty()`!
Run an instrumented app
Interactive staleness analysis
Interactive staleness analysis
## Overhead

<table>
<thead>
<tr>
<th>benchmark</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>richards</td>
<td>10.4X</td>
</tr>
<tr>
<td>deltablue</td>
<td>15X</td>
</tr>
<tr>
<td>crypto</td>
<td>47.1X</td>
</tr>
<tr>
<td>raytrace</td>
<td>41.3X</td>
</tr>
<tr>
<td>earley-boyer</td>
<td>99.8X</td>
</tr>
<tr>
<td>regexp</td>
<td>26.7X</td>
</tr>
<tr>
<td>splay</td>
<td>43.4X</td>
</tr>
<tr>
<td>navier-stokes</td>
<td>45.4X</td>
</tr>
<tr>
<td>pdfjs</td>
<td>31.8X</td>
</tr>
<tr>
<td>box2d</td>
<td>35.8X</td>
</tr>
<tr>
<td>typescript</td>
<td>77.2X</td>
</tr>
</tbody>
</table>

**Low overhead for (most) interactive apps**

74
Reducing Overhead

- Only log the last use of an object (not all uses)
- Don’t log operations on primitive fields
- Enhanced Jalangi to do selective instrumentation
- Binary trace format
- Work with simulated heap as opposed to real heap
  - Reflection too expensive / fragile
Advanced Jalangi Usage
Tracing

- Common technique: store a trace, and do heavyweight analysis over the trace
  - Supported directly in Jalangi 1 via record/replay
  - But, hard to debug and write analyses
- `lib/analysis/Loggers.ts` has all analysis tracing code
- Under Node.js, dump trace to file system (`BinaryFSLogger`)
- From web, trace over web socket (`BinaryWebSocketLogger`)
  - `lib/server/server.ts` has server code
  - pipes trace directly to running lifetime analysis
Integrating Static Analysis

• MemInsight needs the “free variables” of each function
  • Captured by closures, relevant for lifetimes
• Computed by freeVarsAstHandler.ts
• Provided as an AST handler to Jalangi instrumentation
• Jalangi stores result of AST handler inside instrumented code
• For eval’d code, use the instrumentCode callback
Native Methods

- Built-in methods that cannot be instrumented
  - Standard JS library, DOM routines
  - (In general, any uninstrumented code)
- Modeling is analysis-specific
  - For MemInsight, `lib/analysis/NativeModels.ts`
- Also, careful with callbacks from native methods
  - may see `functionEnter` without `invokeFunPre`
Analysis Configuration

- May want analysis-wide configuration options
  - E.g., MemInsight allows for a debug function for dumping ref counts
- Use `--initParam` option to `instrument.js (web)` or `esnstrument_cli.js (node.js)`
- values stored in `J$.initParams`
Debugging with JSDelta

https://github.com/WALA/jsdelta
JSDelta: motivation

• Building a Jalangi analysis
• Works great on unit tests
• But, crashes on jQuery!
• What went wrong? Need a **minimized input**
• Jsdelta does **automatic input minimization**
  • Via **delta debugging** [Zeller, FSE’99]
JSDelta: Demo

Google “JS Delta Walkthrough”
Using JSDelta

- Easy: write a script that prints a message when error occurs
- Also works for JSON, entire directories
- For a Jalangi analysis:
  - Check for errors in uninstrumented program first
  - Always run with a timeout (e.g., with `timeout` command)
  - For browser code, use PhantomJS, Selenium, etc.
DLint and JITProf

DLint: Dynamically Checking JS Coding Practice

Liang Gong, Michael Pradel, Manu Sridharan, Koushik Sen

JITProf: Find JS code that prohibit JIT-optimization

[FSE’15] JITProf: Pinpointing JIT-unfriendly JavaScript code
Liang Gong, Michael Pradel, Koushik Sen
DLint and JITProf for Web Pages

mitmproxy
Observe requests & intercepts responses that contain JS and webpages
DLint and JITProf for Web Pages

mitmproxy
Observe requests & intercepts responses that contain JS and webpages
DLint and JITProf

DLint: Dynamically Checking JS Coding Practice
Liang Gong, Michael Pradel, Manu Sridharan, Koushik Sen

JITProf: Find JS code that prohibit JIT-optimization
[FSE’15] JITProf: Pinpointing JIT-unfriendly JavaScript code
Liang Gong, Michael Pradel, Koushik Sen
What are coding practices?

• Good coding practices
  • Informal rules
  • Improve code quality

• Better quality means:
  • Fewer correctness issues
  • Better performance
  • Better usability
  • Better maintainability
  • Fewer security loopholes
  • Fewer surprises
  • ...
Rule: avoid using `for..in` over arrays

```javascript
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```
Rule: avoid using `for..in` over arrays

```javascript
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?

11 + 22 + 33 => 66
0 + 1 + 2 => 3
0+"0"+"1"+"2" => "0012"
```

array index
(not array value)
array index: string
Rule: avoid using `for..in` over arrays

```javascript
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

11 + 22 + 33 => 66
0 + 1 + 2 => 3
0+"0"+"1"+"2" => "0012"

- Cross-browser issues
- Result depends on the Array prototype object

array index (not array value)
array index: string

> "0012index0ftoString..."
Rule: avoid using `for..in` over arrays

```javascript
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?

for (i=0; i < array.length; i++) {
    sum += array[i];
}

function addup(element, index, array) {
    sum += element;
}
array.forEach(addup);
```
Rule: avoid using `for..in` over arrays

```javascript
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
  sum += value;
}
> sum ?

for (i=0; i < array.length; i++) {
  sum += array[i];
}

function addup(element, index, array) {
  sum += element;
}
array.forEach(addup);
```
Coding Practices and Lint Tools

• **Existing Lint-like checkers**
  – Inspect source code
  – Detect common mistakes

• **Limitations:**
  – Approximates behavior
  – Unknown aliases
  – Lint tools favor precision over soundness

• **Difficulty:** Precise static program analysis
DLint

- Dynamic Linter checking code quality rules for JS
- Open-source, robust, and extensible framework
- Formalized and implemented 28 rules
  - Counterparts of static rules
  - Additional rules
- Empirical study
  - It is better to use DLint and static linter together

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.
Detect **for..in** over arrays with Jalangi

```javascript
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}

> sum ?

for (i=0; i < array.length; i++) {
    sum += array[i];
}

function addup(element, index, array) {
    sum += element;
}
array.forEach(addup);
```
Detect `for..in` over arrays with Jalangi

```javascript
for (value in obj) {
    sum += value;
}
```
Detect *for..in* over arrays with Jalangi

```javascript
for (value in obj) {
    sum += value;
}
```

Have a warning when `obj` in *for-in* is an array.
Detect `for..in` over arrays with Jalangi

```
for (value in obj) {
    sum += value;
}
```

Have a warning when `obj` in `for-in` is an array.

Jalangi Instrumented Code
Detect *for..in* over arrays with Jalangi

```javascript
for (value in obj) {
    sum += value;
}
```

Have a warning when *obj* in *for-in* is an array.

Instrumentation

Jalangi Instrumented Code

```javascript
function forinObject(iid, val) {

}
```
Detect \textit{for..in} over arrays with Jalangi

\begin{verbatim}
for (value in obj) {
    sum += value;
}
\end{verbatim}

Have a warning when \texttt{obj} in \textit{for-in} is an array.

\textbf{Jalangi Instrumented Code}

\begin{verbatim}
function forinObject(iid, val) {
}
\end{verbatim}
Detect `for..in` over arrays with Jalangi

```javascript
for (value in obj) {
    sum += value;
}
```

Have a warning when `obj` in `for-in` is an array.

**Jalangi Instrumented Code**

```javascript
function forinObject(iid, val) {
    if (isArray(val)) {
        // report warning!
    }
}
```
Detect \textit{for..in} over arrays with Jalangi

\begin{verbatim}
for (value in obj) {
    sum += value;
}
\end{verbatim}

Have a warning when \textit{obj} in \textit{for-in} is an array.

Jalangi Instrumented Code

\begin{verbatim}
function forinObject(iid, val) {
    if (isArray(val)) {
        // report warning!
    }
}
\end{verbatim}
Detect **for..in** over arrays with Jalangi

```javascript
for (value in obj) {
  sum += value;
}
```

Have a warning when `obj` in **for-in** is an array.

**Jalangi Instrumented Code**

```javascript
function forinObject(iid, val) {
  if (isArray(val)) {
    J$.iidToLocation(iid);
  }
}
```

Detect **for..in** over arrays with Jalangi

```javascript
for (value in obj) {
  sum += value;
}
```

Have a warning when `obj` in **for-in** is an array.

**Jalangi Instrumented Code**

```javascript
function forinObject(iid, val) {
  if (isArray(val)) {
    J$.iidToLocation(iid);
  }
}
```
Detect `for..in` over arrays with Jalangi

```javascript
for (value in obj) {
    sum += value;
}
```

Have a warning when `obj` in `for-in` is an array.

Jalangi Instrumented Code

```javascript
function forinObject(iid, val) {
    if (isArray(val)) {
        J$.iidToLocation(iid);
    }
}
```

Detect for..in over arrays with Jalangi
Detect *for..in* over arrays with Jalangi

```javascript
for (value in obj) {
    sum += value;
}
```

Have a warning when `obj` in *for-in* is an array.

Jalangi Instrumented Code

```javascript
function forinObject(iid, val) {
    if (isArray(val)) {
        J$.iidToLocation(iid);
    }
}
```

`file.js:<start line>:<start col>:<end line>:<end col>`
Checkers

CheckNaN.js
ConcatUndefinedToString.js
NonObjectPrototype.js
SetFieldToPrimitive.js
OverflowUnderFlow.js
StyleMisuse.js
ToStringGivesNonString.js
UndefinedOffset.js
NoEffectOperation.js
AddEnumerablePropertyToObject.js
ConstructWrappedPrimitive.js
InconsistentNewCallPrefix.js
UncountableSpaceInRegexp.js
FloatNumberEqualityComparison.js

FunctionToString.js
ShadowProtoProperty.js
ForInArray.js
NonNumericArrayProperty.js
OverwrittenPrototype.js
GlobalThis.js
CompareFunctionWithPrimitives.js
InconsistentConstructor.js
FunctionCalledWithMoreArguments.js
IllegalUseOfArgumentsVariable.js
DoubleEvaluation.js
EmptyClassInRegexp.js
UseArrObjConstrWithoutArg.js
MissRadixArgInParseNum.js
PutField(Read("a", a), "f", GetField(Read("b", b), "g"))

\[ a.f = b.g \]
Other Resources

Jalangi (v2) Github
https://github.com/Samsung/jalangi2

DLint + JITProf Github based on Jalangi (v2)
https://github.com/ksen007/jalangi2analyses

JITProf Visualization Github based on Jalangi (v2)
https://github.com/JacksonGL/jitprof-visualization
DLint and JITProf

DLint: Dynamically Checking JS Coding Practice
Liang Gong, Michael Pradel, Manu Sridharan, Koushik Sen

JITProf: Find JS code that prohibit JIT-optimization
[FSE’15] JITProf: Pinpointing JIT-unfriendly JavaScript code
Liang Gong, Michael Pradel, Koushik Sen
Motivation of JITProf

Dynamic language features:

Simplifies coding

• Write less, do more
  → more productive
• Code is less verbose
  → easier to understand
Motivation of JITProf

Dynamic language features:

Simplifies coding
• Write less, do more
  → more productive
• Code is less verbose
  → easier to understand

Slow execution
• Too many runtime checks
• Object property lookup -> hash table lookup
  …
Pinpointing JIT-unfriendly JavaScript Code

• Code snippet from Google Octane Benchmark:

```javascript
SplayTree.prototype.insert = function(key, value) {
  ...
  var node = new SplayTree.Node(key, value);
  if (key > this.root_.key) {
    node.left = this.root_;  
    node.right = this.root_.right;
  }
  else {
    node.right = this.root_;  
    node.left = this.root_.left;
  }
  this.root_ = node;
};
```

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.
Pinpointing JIT-unfriendly JavaScript Code

• Code snippet from Google Octane Benchmark:

    SplayTree.prototype.insert = function(key, value) {
        ...
        var node = new SplayTree.Node(key, value);
        if (key > this.root_.key) {
            node.left = this.root_;  
            node.right = this.root_.right;
            ...
        } else {
            node.right = this.root_;  
            node.left = this.root_.left;
            ...
        }
        this.root_ = node;
    };

    • node has two layouts:
      offset of left in node can be 0 or 1
    • JIT cannot replace node.left with node[0] or node[1]
Pinpointing JIT-unfriendly JavaScript Code

- Code snippet from Google Octane Benchmark:

```javascript
SplayTree.prototype.insert = function(key, value) {
    ...
    var node = new SplayTree.Node(key, value);
    if (key > this.root_.key) {
        node.left = this.root_;  // Performance boost: 15%
        node.right = this.root_.right;
        ...
    } else {
        node.right = this.root_;  // Performance boost: 6.7%
        node.left = this.root_.left;
        ...
    }
    this.root_ = node;
};
```

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.
Pinpointing JIT-unfriendly JavaScript Code

• Code snippet from Google Octane Benchmark:

```javascript
SplayTree.prototype.insert = function(key, value) {
  var node = new SplayTree.Node(key, value);
  if (key > this.root_.key) {
    node.left = this.root_; // Performance boost 15%
    node.right = this.root_.right;
    ...
  } else {
    node.right = this.root_; // Performance boost 6.7%
    node.left = this.root_.left;
    ...
  }
  this.root_ = node;
};
```

JITProf Simulates the Hidden Classes based on the information provided by Jalangi
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000;i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000;i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000;i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
Back to the Motivating Example

```javascript
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000;i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
```

Hidden class simulation before the statement

Hidden class simulation after the statement
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000;i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000;i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
function Thing(flag) {
  if (!flag) {
    this.b = 4;
    this.a = 3;
  } else {
    this.a = 2;
    this.b = 1;
  }
}

for(var i = 0; i<1000000;i++) {
  var o = new Thing(i%2);
  result += o.a + o.b;
}

function putFieldPre (iid, base, offset, val ... ) {
  // logic for updating the hidden class
}

© Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.
Back to the Motivating Example

```javascript
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for (var i = 0; i < 1000000; i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
```

```
function putFieldPre (iid, base, offset, val ... ) {
    // logic for updating the hidden class
}
```

Hidden class simulation before the statement

```
this.b = 4;
```

invoke

Hidden class simulation after the statement
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000;i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}

function putFieldPre (iid, base, offset, val ... ) {
    // logic for updating the hidden class
}
function Thing(flag) {
  if (!flag) {
    this.b = 4;
    this.a = 3;
  } else {
    this.a = 2;
    this.b = 1;
  }
}

for(var i = 0; i<1000000;i++) {
  var o = new Thing(i%2);
  result += o.a + o.b;
}

function putFieldPre (iid, base, offset, val ... ) {
  // logic for updating the hidden class
}

function putFieldPost (lid, base, offset, val ... ) {
  // logic for updating the hidden class
}
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000;i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}

function putFieldPre (iid, base, offset, val ... ) {
    // logic for updating the hidden class
}

Hidden class simulation before the statement

Hidden class simulation after the statement
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000; i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}

function putFieldPre (iid, base, offset, val ... ) {
    var sobj = J$.smemory.getShadowObject(base);
    sobj.hiddenClass ...
}
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i < 1000000; i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}

var o = {a: 1, b: 2};

Intercept *putField* to update the hidden class
Back to the Motivating Example

```
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000;i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}

var o = {a: 1, b: 2};
```

Intercept `putField` to update the hidden class

Intercept `invokeFun` to record object creation location
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for(var i = 0; i<1000000;i++) {
    var o = new Thing(i%2);
    result += o.a + o.b,
}

var o = {a: 1, b: 2};
function Thing(flag) {
  if (!flag) {
    this.b = 4;
    this.a = 3;
  } else {
    this.a = 2;
    this.b = 1;
  }
}

for(var i = 0; i<1000000;i++) {
  var o = new Thing(i%2);
  result += o.a + o.b;
}

var o = {a: 1, b: 2};
JIT-unfriendly Code Checked by JITProf

- Use inconsistent object layout
- Access undeclared property or array element
- Store non-numeric value in numeric arrays
- Use in-contiguous keys for arrays
- Not all properties are initialized in constructors
- … and more
Rule #5: Use Contiguous Keys for Array

```javascript
var array = [];
for (var i=10000;i>=0;i--){
    array[i] = i;
}
```
Rule #5: Use Contiguous Keys for Array

```javascript
var array = [];
for (var i=10000;i>=0;i--){
    array[i] = i;
}

array[10000] = 10000;
array[9999] = 9999;
...
```

• non-contiguous array

• To save memory, JIT-engine decides to represent the array with slow data structures like hash table.
Rule #5: Use Contiguous Keys for Array

```javascript
var array = [];
for (var i=10000; i>=0; i--){
    array[i] = i;
}

for (var i=0; i<=10000; i++){
    array[i] = i;
}
```

10X+ speedup!
Rule #5: Use Contiguous Keys for Array

```javascript
var array = [];
for (var i=10000;i>=0;i--){
    array[i] = i;
}
```

- Intercept `putField` operation of arrays
- Rank locations by number assignments to non-contiguous arrays
<table>
<thead>
<tr>
<th></th>
<th>group</th>
<th>average</th>
<th>improve rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunspider-chrome-sha1 (*)</td>
<td>original</td>
<td>1884.7588</td>
<td>26.3%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>1299.0706</td>
<td></td>
</tr>
<tr>
<td>octane-firefox-Splay</td>
<td>original</td>
<td>11331.59</td>
<td>3.5%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>12198.65</td>
<td></td>
</tr>
<tr>
<td>Sunspider-String-Tagcloud (*)</td>
<td>original</td>
<td>9178.76</td>
<td>11.7%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>9457.53</td>
<td></td>
</tr>
<tr>
<td>octane-firefox-DeltaBlue</td>
<td>original</td>
<td>28473.53</td>
<td>1.4%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>31154.06</td>
<td></td>
</tr>
<tr>
<td>octane-chrome-Box2D</td>
<td>original</td>
<td>24569.47</td>
<td>7.5%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>24915.00</td>
<td></td>
</tr>
<tr>
<td>octane-chrome-RayTrace</td>
<td>original</td>
<td>43595.94</td>
<td>12.9%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>48140.35</td>
<td></td>
</tr>
</tbody>
</table>

(*) means smaller is better

higher → better
<table>
<thead>
<tr>
<th>(*) means smaller is better</th>
<th>group</th>
<th>average</th>
<th>improve rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>octane-chrome-Splay</td>
<td>original</td>
<td>10278.59</td>
<td>15.1%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>11885.71</td>
<td></td>
</tr>
<tr>
<td>octane-chrome-SplayLatency</td>
<td>original</td>
<td>20910.24</td>
<td>3.8%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>21994.82</td>
<td></td>
</tr>
<tr>
<td>sunspider-chrome-3d-Cube (*)</td>
<td>original</td>
<td>597.047059</td>
<td>1.1%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>593.744118</td>
<td></td>
</tr>
<tr>
<td>sunspider-firefox-sha1 (*)</td>
<td>original</td>
<td>680.476471</td>
<td>3.3%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>669.932353</td>
<td></td>
</tr>
<tr>
<td>sunspider-firefox-Xparb (*)</td>
<td>original</td>
<td>364.6824</td>
<td>19.7%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>357.2235</td>
<td></td>
</tr>
<tr>
<td>sunspider-chrome-md5 (*)</td>
<td>original</td>
<td>774.3500</td>
<td>24.6%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>665.8382</td>
<td></td>
</tr>
<tr>
<td>sunspider-chrome-format-tofte (*)</td>
<td>original</td>
<td>212.2029</td>
<td>3.4%</td>
</tr>
<tr>
<td></td>
<td>refactored</td>
<td>200.9000</td>
<td></td>
</tr>
</tbody>
</table>

higher \(\rightarrow\) better
Install DLint and JITProf with Jalangi2

https://github.com/ksen007/jalangi2analyses

```bash
npm install
```

**mitmproxy** (third-party framework)

```bash
pip install pyOpenSSL
pip install mitmproxy==0.11.3
```

Install the mitmproxy certificate manually *(drag-and-drop)*
• man-in-the-middle proxy
• Interactive, SSL-capable proxy for HTTP with a console interface.
• Intercept http communication between the client and the server for instrumentation.
Install mitmproxy

- `pip install pyOpenSSL`
- `pip install mitmproxy==0.11.3`
Install mitmproxy

- `pip install pyOpenSSL`
- `pip install mitmproxy==0.11.3`
The HTTPS Problem

- **Man-in-the-middle Proxy**
- **SSL and HTTPS** is designed against MITM
- **HTTPS Handle shake error** due to uncertified modification via instrumentation

Browser ⇔ mitmproxy + Jalangi ⇔ Server

Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.
The HTTPS Problem

- **Man-in-the-middle Proxy**
- **SSL and HTTPS** is designed against MITM
- **HTTPS Handle shake error** due to uncertified modification via instrumentation

Browser | mitmproxy + Jalangi Instrumentation + a Certificate Authority Implementation | Server
The HTTPS Problem

• Man-in-the-middle Proxy
• SSL and HTTPS is designed against MITM
• HTTPS Handle shake error due to uncertified modification via instrumentation

Browser

mitmproxy + Jalangi Instrumentation + a Certificate Authority Implementation

Server

request

forwarded request

response
Other Resources

Jalangi (v2) Github
https://github.com/Samsung/jalangi2

DLint + JITProf Github based on Jalangi (v2)
https://github.com/ksen007/jalangi2analyses

JITProf Visualization Github based on Jalangi (v2)
https://github.com/JacksonGL/jitprof-visualization

Questions
Thank You