BLeak: Automatically Debugging Memory Leaks in Web Applications

John Vilk, Emery D. Berger

PLDI’18

https://home.cse.ust.hk/~yzhanghw/

Presented by: ZHANG Yushan
Outline

• Memory Leaks in Web Applications
• Why existing work are not effective
• BLeak’s Solution
• Summary
• Discussions
Type of memory leaks in web applications

• Failing to dispose of unneeded event listeners
• Repeatedly injecting iframes and CSS files
• Failing to call cleanup routines in third-party libraries
Memory Leak in Web Applications

```javascript
class Preview extends PureComponent {
  // Runs when Preview is added to GUI
  componentDidMount() {
    const { codeMirror } = this.props.editor;
    const wrapper = codeMirror.getWrapperElement();
    codeMirror.on("scroll", this.onScroll);
    wrapper.addEventListener("mouseover", this._mover);
    wrapper.addEventListener("mouseup", this._mup);
    wrapper.addEventListener("mousedown", this._mdown);
  }
}
```

Fig 1. *Firefox’s Debugger* leaks ‘this’ four times for every execution
How to find memory leaks manually?

Currently, developers debug leaks by first examining heap snapshots to find leaking objects (Figure 2a). Then, they try to use retaining paths to locate the code responsible.
Existing Methods

- **Staleness metric**
  - Find untouched memory objects and rank them for garbage collection

  The four listeners execute every time when mouse is moved, marking the objects as “fresh”

- **Growth method** (growing memory usage) requires leak objects to satisfy:
  - owned (dominated) by a single object or
  - they form strongly connected components in the heap

  The four listeners are owned by different objects (leak roots) & are strongly connected to nearly the whole heap
BLeak's Idea

• Over a single session, users repeatedly return to the same visual state in modern web sites.
• visits to the same (approximate) visual state should consume roughly the same amount of memory.
• sustained memory growth between visits is a strong indicator of a memory leak.

After we read on email and return to Inbox, Gmail should not have continuous growth of memory usage.
BLeak’s Workflow

Input Scripts → Identifying Leaked Heap Objects → Diagnosing Stack Trace → Ranking Leaks → Reports
B Leak’s Workflow

Perform round-trip visits to a visual state
```javascript
class Preview extends PureComponent {
  // Runs when Preview is added to GUI
  componentDidMount() {
    const { codeMirror } = this.props.editor;
    const wrapper = codeMirror.getWrapperElement();
    codeMirror.on("scroll", this.onScroll);
    wrapper.addEventListener("mouseover", this._mover);
    wrapper.addEventListener("mouseout", this._mup);
    wrapper.addEventListener("mousedown", this._mdown);
  }
}

exports.loop = [
  // Loop that repeatedly opens and closes a source document.
  // First, open a source document in the text editor.
  {
    check: function() {
      const nodes = $('node');
      // No documents are open
      return $('source-tab').length === 0 &&
        // Target document appears in doc list
        nodes.length > 1 && nodes[1].innerText === "main.js";
    },
    next: function() { $('node')[1].click(); }
  },
  // Next, close the document after it loads.
  {
    check: function() {
      // Contents of main.js are in editor
      return $('CodeMirror-line').length > 2 &&
        // Editor displays a tab for main.js
        $('source-tab').length === 1 &&
        // Tab contains a close button
        $('close-btn').length === 1;
    },
    next: function() { $('close-btn').click(); }
  }]

(a) This script runs the Firefox debugger in a loop, and is the only input BLEAK requires to automatically locate memory leaks. For brevity, we modify the script to use jQuery syntax.
```
B Leak’s Workflow

Find growing memory usage
Round Trip

window
globals
history
“/homes”

resize event listeners

document

GUI objects

click event listeners

resize

click

1MB
Round Trip

globals
window
document
GUI objects

history

resize event listeners

click event listeners

"/homes"

"/foryou"

"/homes"

resize
resize

1MB
1MB

1MB
1MB

2019/6/27
ZHANG Yushan
A very large subgraph

ZHANG Yushan
32 leak roots found on airbnb.com!
BLeak's Workflow

1. Input Scripts
2. Identifying Leaked Heap Objects
3. Ranking Leaks
4. Diagnosing Stack Trace
5. Reports

Prioritize leaks with memory usage
Ranking Leaks: **LeakShare**

- Split memory usage of an object among leak roots

Free more memory effectively!
BLeak’s Workflow

- Input Scripts
- Identifying Leaked Heap Objects
- Ranking Leaks
- Diagnosing Stack Trace

Collect stack trace for leaks
Round Trip

Collected via JavaScript reflection + program transformations

window

document

history

resize event listeners

click event listeners

"/homes"

resize

click

vendor.js:32:120, -

app.js:10:930, -

app.js:10:952, -
BLeak’s Workflow
An example of leak text report

ZHANG Yushan

---

```javascript
class Preview extends PureComponent {
  // Runs when Preview is added to GUI
  componentDidMount() {
    const { codeMirror } = this.props.editor;
    const wrapper = codeMirror.getWrapperElement();
    codeMirror.on("scroll", this.onScroll);
    wrapper.addEventListener("mouseover", this._mover);
    wrapper.addEventListener("mousemove", this._mup);
    wrapper.addEventListener("mousedown", this._mdown);
  }
}
```

# Leak Root 1 [LeakShare: 811920]

## Leak Paths

* Event listeners for 'mouseover' on window.cm.display.wrapper

## Stack Traces Responsible

1. Preview.componentDidMount
3. measureLifeCyclePerf
   http://localhost:8000/assets/build/debugger.js:81531:11
5. CallbackQueue.notifyAll
   http://localhost:8000/assets/build/debugger.js:61800:21
6. ReactReconcileTransaction.close
   http://localhost:8000/assets/build/debugger.js:83305:25
7. ReactReconcileTransaction.closeAll
   http://localhost:8000/assets/build/debugger.js:42268:24
Evaluation - impacts of fixing the bugs

Figure 8. Impact of fixing memory leaks found with BLEAK: Graphs display live heap size over round trips; error bars indicate the 95% confidence interval. Fixing the reported leaks eliminates an average of 93% of all heap growth.
## Evaluation - Effectiveness

<table>
<thead>
<tr>
<th>Program</th>
<th>Loop LOC</th>
<th>Leak Roots</th>
<th>False Pos.</th>
<th>Distinct Leaks</th>
<th>Stale Leaks</th>
<th>Prec.</th>
<th>Growth Reduction</th>
<th>Total</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airbnb</td>
<td>17</td>
<td>32</td>
<td>2</td>
<td>32</td>
<td>4</td>
<td>94%</td>
<td>1.04 MB (81.0%)</td>
<td>224 s</td>
<td>7.1%</td>
</tr>
<tr>
<td>Piwik</td>
<td>32</td>
<td>17</td>
<td>0</td>
<td>11</td>
<td>4</td>
<td>100%</td>
<td>8.14 MB (99.3%)</td>
<td>149 s</td>
<td>3.7%</td>
</tr>
<tr>
<td>Loomio</td>
<td>73</td>
<td>10</td>
<td>1</td>
<td>9</td>
<td>4</td>
<td>90%</td>
<td>2.83 MB (98.3%)</td>
<td>388 s</td>
<td>0.4%</td>
</tr>
<tr>
<td>Mailpile</td>
<td>37</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>100%</td>
<td>0.80 MB (91.8%)</td>
<td>214 s</td>
<td>2.3%</td>
</tr>
<tr>
<td>Firefox Debugger</td>
<td>17</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>100%</td>
<td>0.47 MB (98.2%)</td>
<td>243.8 s</td>
<td>3.4%</td>
</tr>
</tbody>
</table>

**Total / mean:** 35 67 3 59 13 96.8% 2.66 MB (93.7%) 243.8 s 3.4% 12.5% 31.4%

**Figure 9. BLeak precisely finds impactful memory leaks:** On average, BLeak finds these leaks with over 95% precision, and fixing them eliminates over 90% of all heap growth. 77% of these leaks would not be found with a staleness metric (§6.5).
### Growth Reduction for Top Leaks Fixed

<table>
<thead>
<tr>
<th>Program</th>
<th>Metric</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airbnb</td>
<td>LeakShare</td>
<td>0K</td>
<td>111K</td>
<td>462K</td>
</tr>
<tr>
<td></td>
<td>Retained Size</td>
<td>0K</td>
<td>0K</td>
<td>105K</td>
</tr>
<tr>
<td></td>
<td>Trans. Closure Size</td>
<td>0K</td>
<td>196K</td>
<td>393K</td>
</tr>
<tr>
<td>Loomio</td>
<td>LeakShare</td>
<td>0K</td>
<td>1083K</td>
<td>2878K</td>
</tr>
<tr>
<td></td>
<td>Retained Size</td>
<td>64K</td>
<td>186K</td>
<td>2898K</td>
</tr>
<tr>
<td></td>
<td>Trans. Closure Size</td>
<td>59K</td>
<td>67K</td>
<td>2398K</td>
</tr>
<tr>
<td>Mailpile</td>
<td>LeakShare</td>
<td>613K</td>
<td>817K</td>
<td>820K</td>
</tr>
<tr>
<td></td>
<td>Retained Size</td>
<td>613K</td>
<td>817K</td>
<td>820K</td>
</tr>
<tr>
<td></td>
<td>Trans. Closure Size</td>
<td>0K</td>
<td>0K</td>
<td>201K</td>
</tr>
<tr>
<td>Piwik</td>
<td>LeakShare</td>
<td>8003K</td>
<td>8104K</td>
<td>8306K</td>
</tr>
<tr>
<td></td>
<td>Retained Size</td>
<td>2073K</td>
<td>7969K</td>
<td>8235K</td>
</tr>
<tr>
<td></td>
<td>Trans. Closure Size</td>
<td>103K</td>
<td>110K</td>
<td>374K</td>
</tr>
</tbody>
</table>

**Figure 10. Performance of ranking metrics:** Growth reduction by metric after fixing quartiles of top ranked leaks. **Bold** indicates greatest reduction (±1%). We omit Firefox because it has only four leaks which must all be fixed (see §2). LeakShare generally outperforms or matches other metrics.
Summary

• This paper introduces B Leak (Browser Leak debugger), the first system for automatically debugging memory leaks in web applications.

• B Leak’s algorithms leverage the observation that sustained growth between round trips is a strong indicator of a memory leak.

• B Leak automatically generates a list of leaks found along with their root causes, ranked by return on investment.
Discussions (I)

• This paper shows very enough motivation
  • Why memory leaks in web applications are different
  • Why existing methods/tools are not effective

• This paper uses an illustrated example
  • It is used to explain `event listener` leaks
  • It is used to demonstrate existing tools
  • It is used as running example for new solution
Discussions (II)

• It heavily relies on developer-provided test scripts
  • Test script maintenance
  • Only could test part of the program

• It could be applied on GUI memory leak detections (also mentioned in the paper)
  • With automated event sequence generation