Memory leaks:
How not to crash people's browsers

Stoyan Stefanov

@stoyanstefanov, @stoyan@indieweb.social

GDG@Wix Jan 17, 2023

The plan

1. Collecting crash data
2. Detect memory leaks
3. Explore common leakage patterns

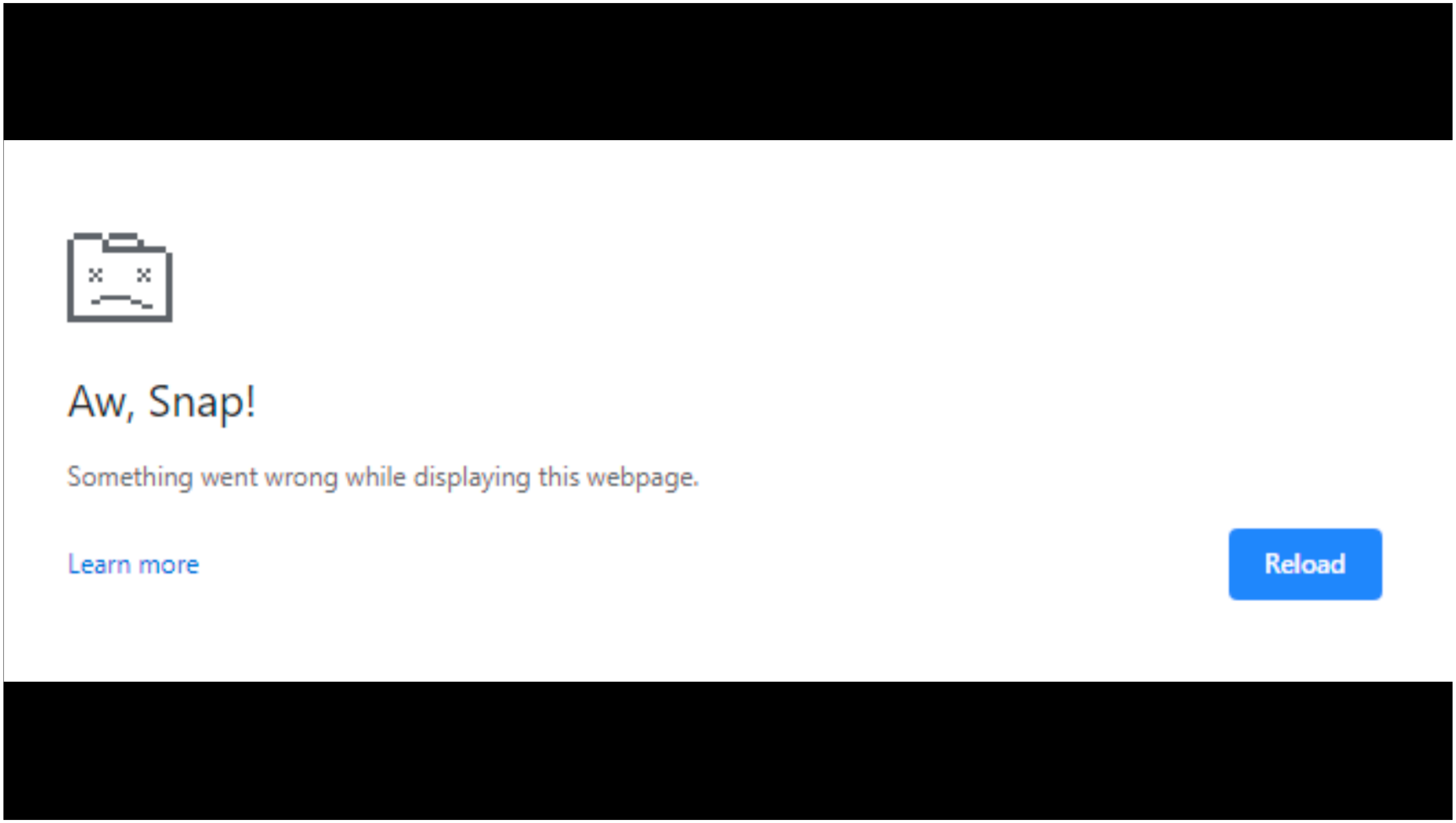
Audience

- React
- … or other SPAs

About me

- WebPageTest.org engineer
- ex-Yahoo
- ex-Facebook

Collecting data

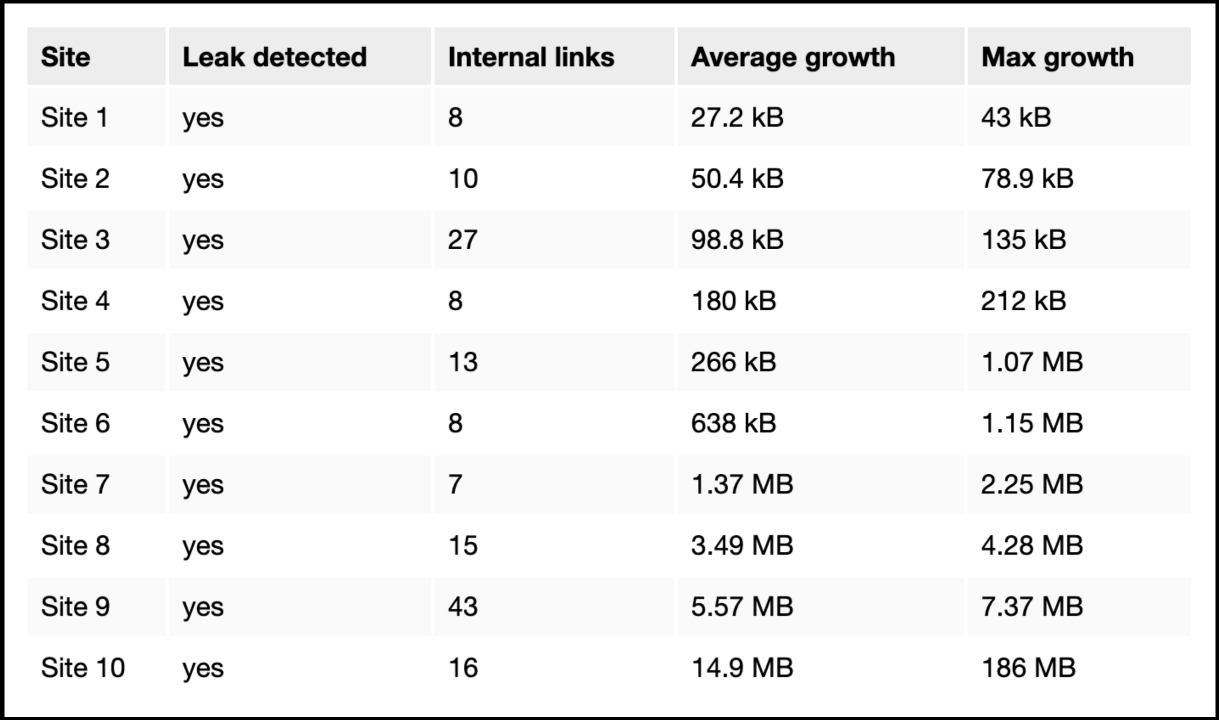

Aw, Snap!

Something went wrong while displaying this webpage.

Top 10 SPAs examined with the **`fuite`** tool

- 186MB after a *single* interaction?
- Everybody leaks
  - https://nolanlawson.com/2021/12/17/introducing-fuite-a-tool-for-finding-memory-leaks-in-web-apps/

Do you have a problem?



Site	Leak detected	Internal links	Average growth	Max growth
Site 1	yes	8	27.2 kB	43 kB
Site 2	yes	10	50.4 kB	78.9 kB
Site 3	yes	27	98.8 kB	135 kB
Site 4	yes	8	180 kB	212 kB
Site 5	yes	13	266 kB	1.07 MB
Site 6	yes	8	638 kB	1.15 MB
Site 7	yes	7	1.37 MB	2.25 MB
Site 8	yes	15	3.49 MB	4.28 MB
Site 9	yes	43	5.57 MB	7.37 MB
Site 10	yes	16	14.9 MB	186 MB

Do *you* have a problem?

- Is your app crashing the user’s browser?
- Reporting API
- Your app can beacon back ”oom” and “unresponsive” browser crashes

Reporting API

- Headers
  - `Reporting-Endpoints:`
  - `Report-To:`
- JavaScript
  - `ReportingObserver`

Reporting API

- Out of memory
- Crashes
- Security violations (CSP)
- Deprecated features
- …

Detecting leaks

How do you detect memory leaks?

Option A: phone a friend

How do you detect memory leaks?

Option B: take memory heap snapshots

1. Load a starting page, GC
2. “navigate” to the target page/interaction, GC
3. navigate back to square 1, GC
4. Diff the snapshots from steps #1 and #3.
   If not the same, memory may have leaked.

If only there was a tool

to just give me the results...

github.com/nolanlawson/fuite

github.com/facebook/memlab

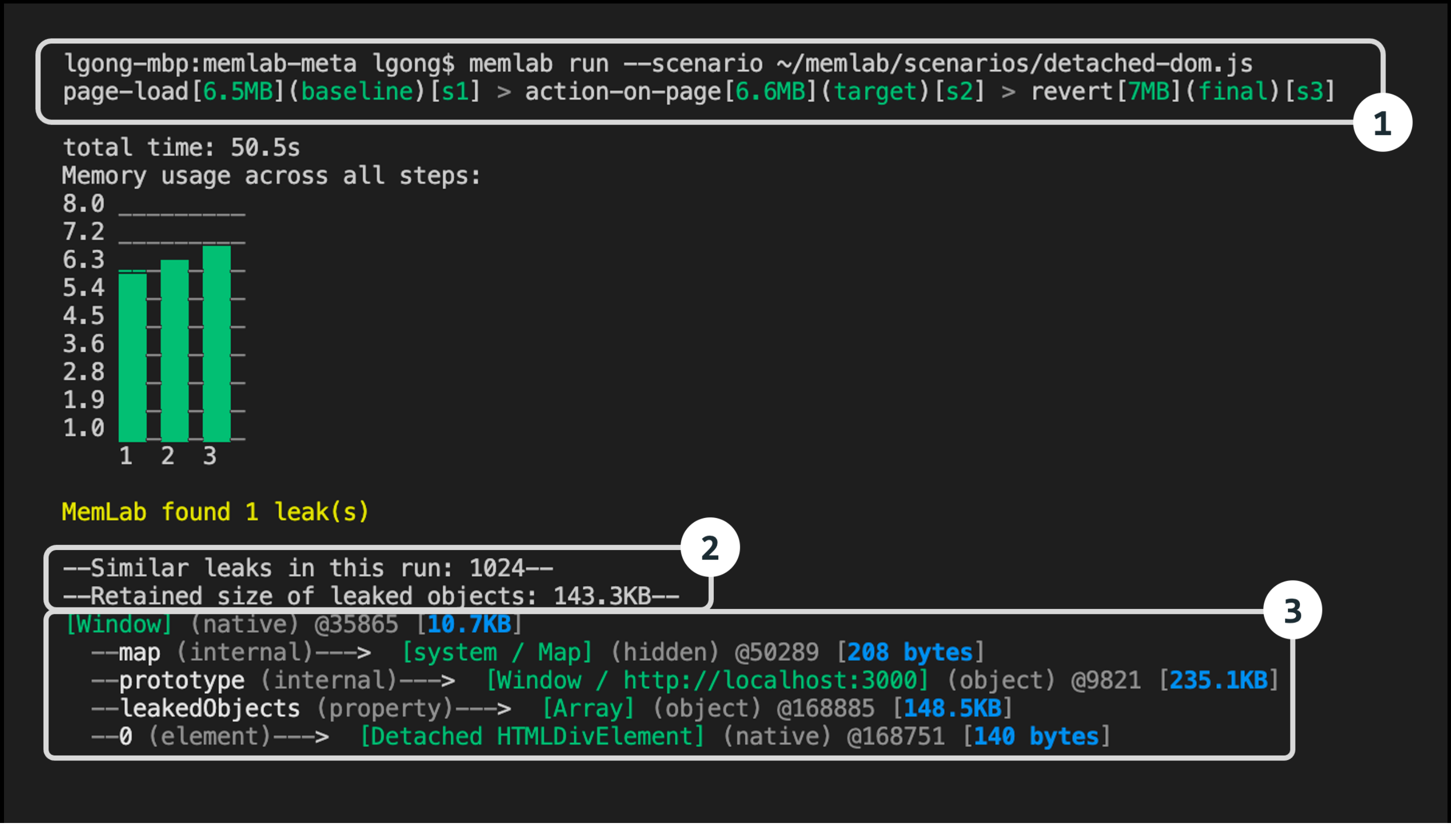
Memlab

- Command-line tool
- Uses Puppeteer to load a page and navigate forward-then-back
- Diffs heap snapshots

A scenario .js

```javascript
function url() {
  return 'https://example.org/';
}
async function action(page) {
  await page.click('button[id="Next"]');
}
async function back(page) {
 await page.click('button[id="Home"]');
}
module.exports = {action, back, url};
```

$ memlab run --scenario ~/scenario.js



```
lgong-mbp:memlab-meta lgong$ memlab run --scenario ~/memlab/scenarios/detached-dom.js
page-load[6.5MB](baseline)[s1] > action-on-page[6.6MB](target)[s2] > revert[7MB](final)[s3]

total time: 50.5s
Memory usage across all steps:
8.0 _________
7.2 _________
6.3 _________
5.4
4.5
3.6
2.8
1.9
1.0
     1  2  3

MemLab found 1 leak(s)

--Similar leaks in this run: 1024--
--Retained size of leaked objects: 143.3KB--
[Window] (native) @35865 [10.7KB]
  --map (internal)---> [system / Map] (hidden) @50289 [208 bytes]
  --prototype (internal)---> [Window / http://localhost:3000] (object) @9821 [235.1KB]
  --leakedObjects (property)---> [Array] (object) @168885 [148.5KB]
  --0 (element)---> [Detached HTMLDivElement] (native) @168751 [140 bytes]
```

The problem

```
window.leakedObjects = [];

for (let i = 0; i < 1024; i++) {
  window.leakedObjects.push(
    document.createElement('div’)
  );
}
```

Results!

- Can be overwhelming
- Grouping of finding and one sample
- Custom analyzers

Spot the leak

Results!

- Can be surprising
- Is this a memory leak?

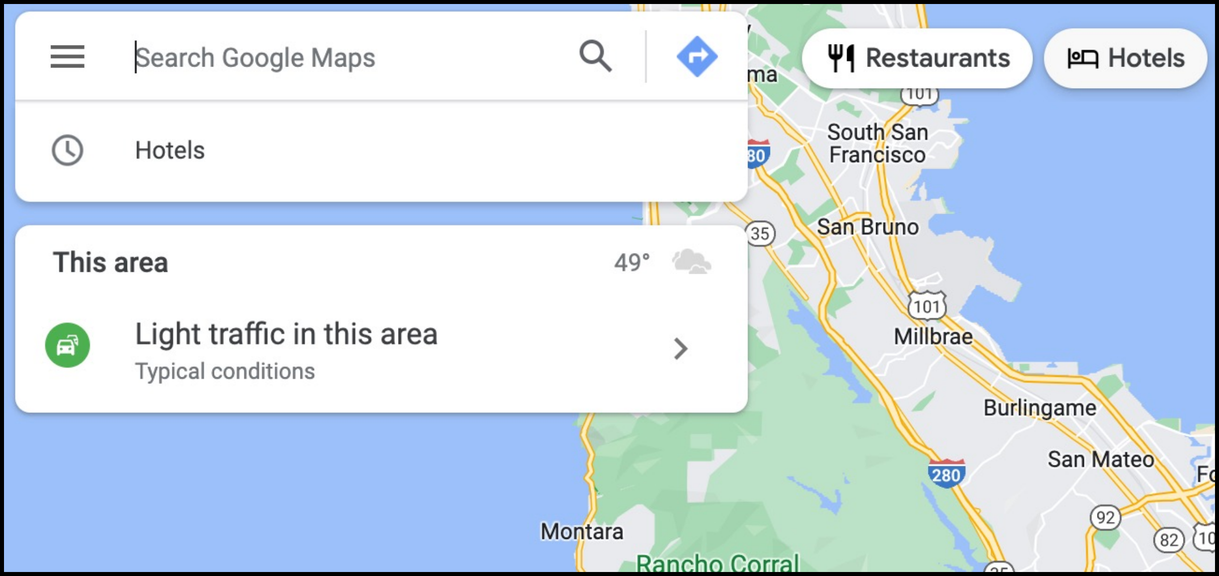
```
let obj = {};
console.log(obj);
obj = null;
```

Leaks everywhere

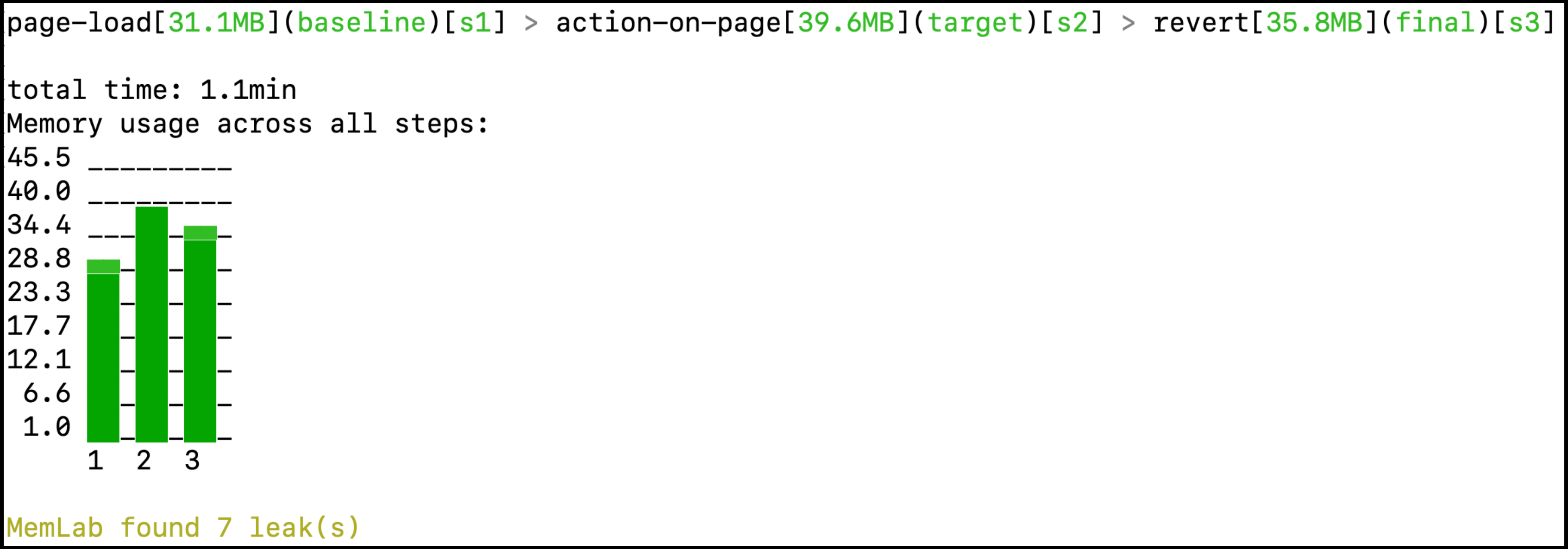
Google Maps scenario .js

```
function url() {
  return 'https://www.google.com/maps/@37.386427,-
122.0428214,11z';
}
async function action(page) {
  await page.click('button[aria-label="Hotels"]');
}
async function back(page) {
 await page.click('[aria-label="Clear search"]');
}
module.exports = {action, back, url};
```

Google Maps


Search Google Maps
Hotels
This area
49°
Light traffic in this area
Typical conditions
Restaurants
Hotels
South San Francisco
San Bruno
Millbrae
Burlingame
San Mateo
Montara
Rancho Corral
101
280
35
92
82

Google Maps



```
page-load[31.1MB](baseline)[s1] > action-on-page[39.6MB](target)[s2] > revert[35.8MB](final)[s3]

total time: 1.1min
Memory usage across all steps:
45.5 _________
40.0 _________
34.4 ___   ___
28.8
23.3
17.7
12.1
 6.6
 1.0
     1  2  3

MemLab found 7 leak(s)
```

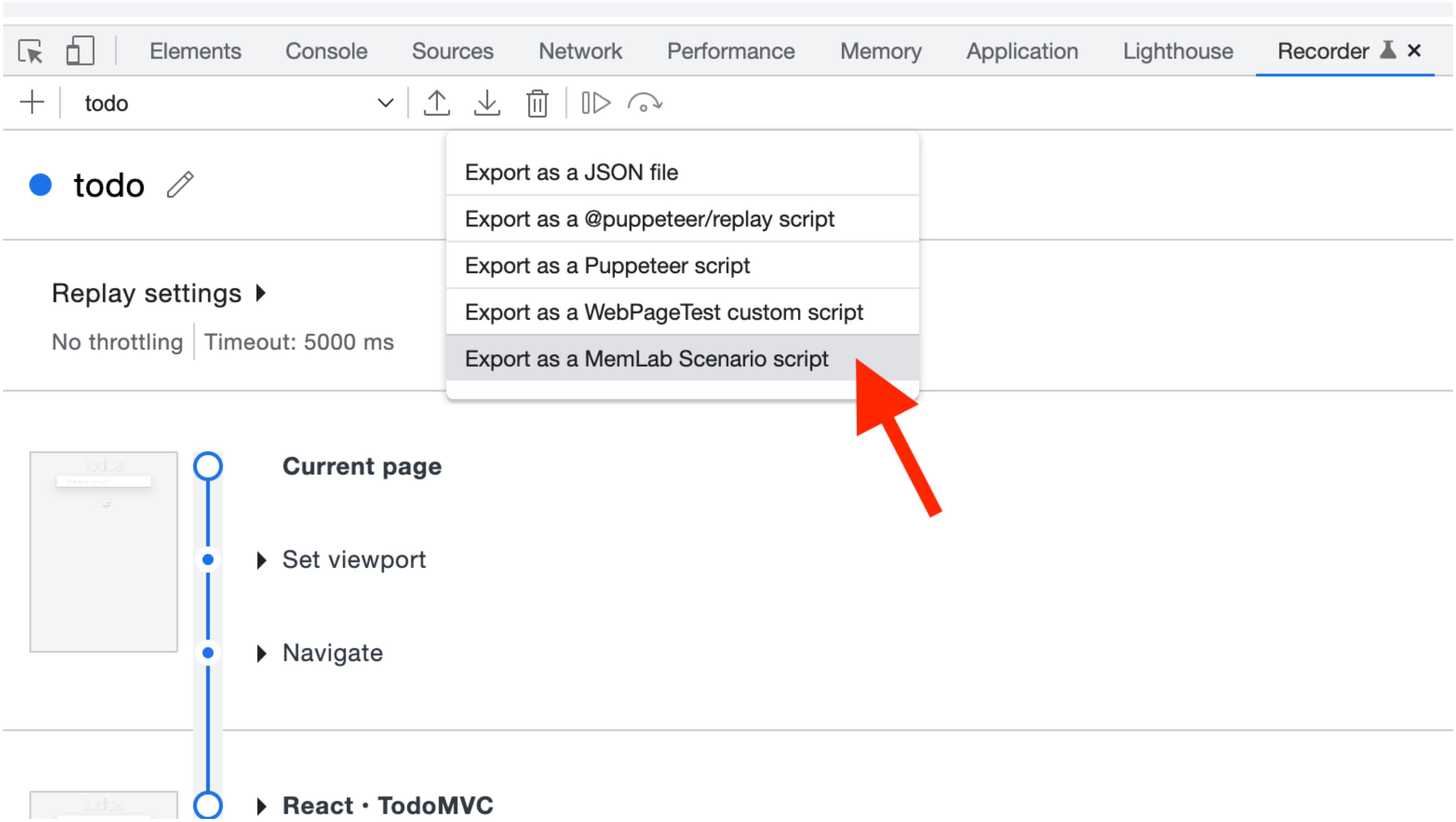
Scenario files

Scenario

- url()
- action()
- back()
- cookies()
- leakFilter()
- …

Memlab recorder

- A little help with Puppeteer
- Extension to the Recorder panel in DevTools
- https://github.com/stoyan/memlab-recorder


Elements
Console
Sources
Network
Performance
Memory
Application
Lighthouse
Recorder
todo
Export as a JSON file
Export as a @puppeteer/replay script
Export as a Puppeteer script
Export as a WebPageTest custom script
Export as a MemLab Scenario script
todo
Replay settings
No throttling
Timeout: 5000 ms
Current page
Set viewport
Navigate
React · TodoMVC

WebPageTest scenario .js

```javascript
// initial page load
function url() {
  return 'https://www.webpagetest.org/';
}

// action where we want to detect memory leaks
async function action(page /* Puppeteer page API */) {
  let el;
  el = await page.waitForSelector('#analytical-review > div:nth-child(3) > label');
  await el.evaluate(b => b.click());
}

// go back to the initial state
async function back(page /* Puppeteer page API */) {
  const el = await page.waitForSelector('#analytical-review > div:nth-child(2) > label');
  await el.evaluate(b => b.click());
}

module.exports = {action, back, url};
```

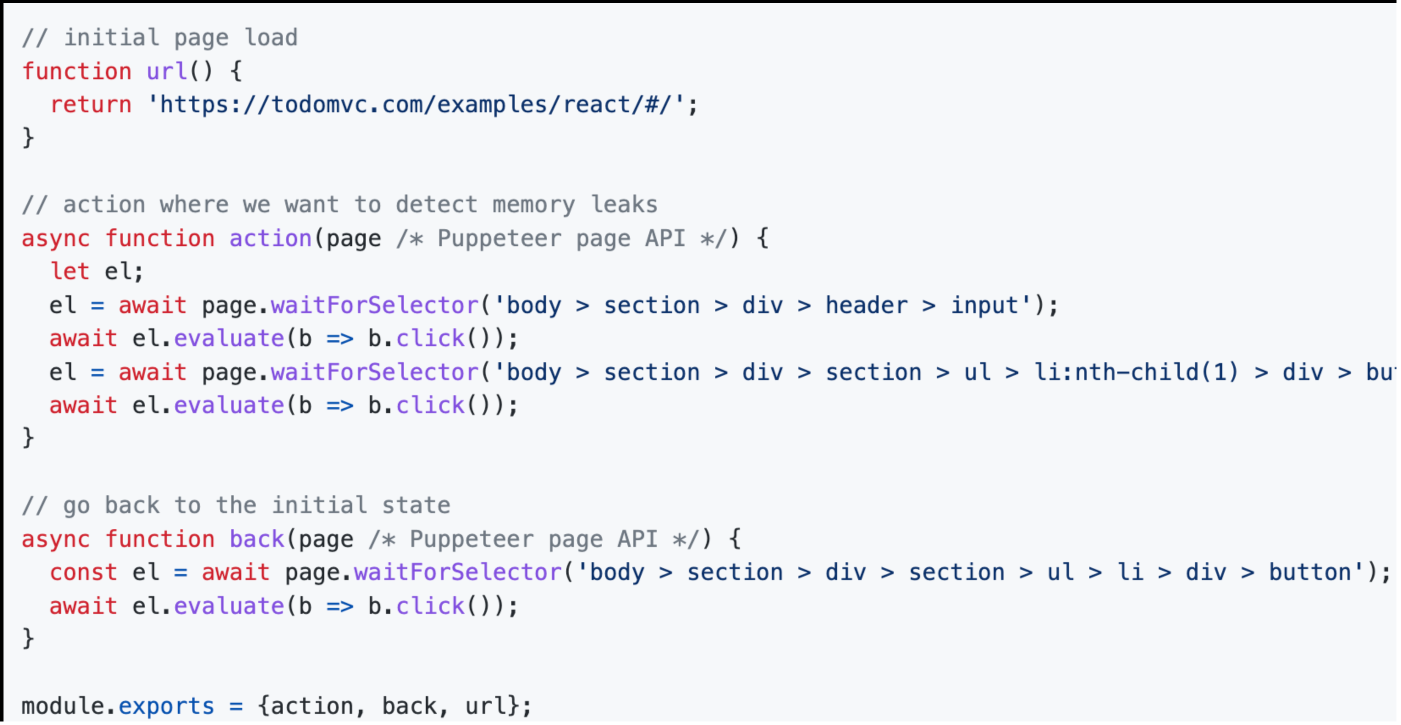
Memlab recorder

- Only initial navigation and clicks
- The last click is attributed to `back()`
- All others are `action()`
- It’s a start for you to tweak

React TODO scenario

Out of

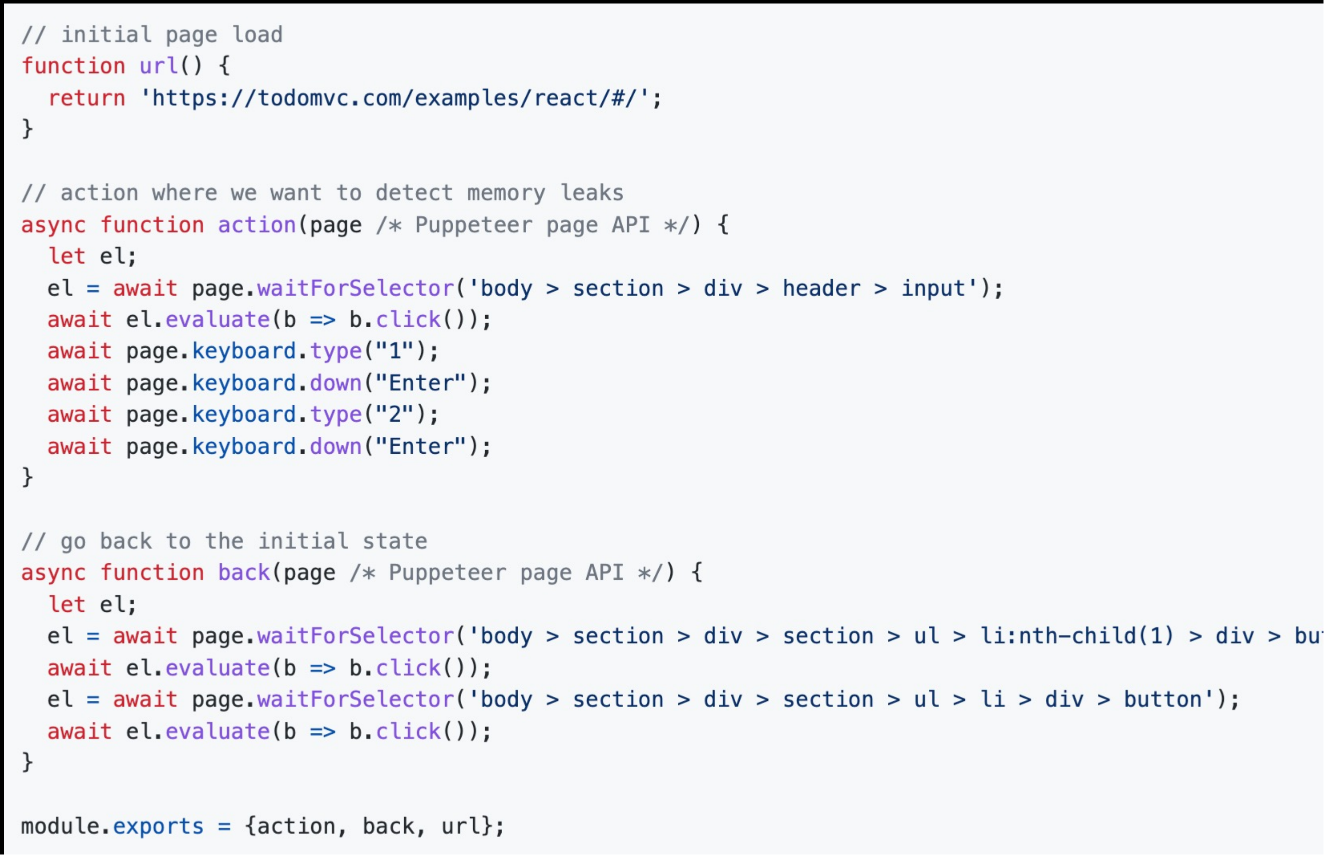
the box:



```
// initial page load
function url() {
  return 'https://todomvc.com/examples/react/#/';
}

// action where we want to detect memory leaks
async function action(page /* Puppeteer page API */) {
  let el;
  el = await page.waitForSelector('body > section > div > header > input');
  await el.evaluate(b => b.click());
  el = await page.waitForSelector('body > section > div > section > ul > li:nth-child(1) > div > bu
  await el.evaluate(b => b.click());
}

// go back to the initial state
async function back(page /* Puppeteer page API */) {
  const el = await page.waitForSelector('body > section > div > section > ul > li > div > button');
  await el.evaluate(b => b.click());
}

module.exports = {action, back, url};
```

React TODO scenario

Tweaked



```javascript
// initial page load
function url() {
  return 'https://todomvc.com/examples/react/#/';
}

// action where we want to detect memory leaks
async function action(page /* Puppeteer page API */) {
  let el;
  el = await page.waitForSelector('body > section > div > header > input');
  await el.evaluate(b => b.click());
  await page.keyboard.type("1");
  await page.keyboard.down("Enter");
  await page.keyboard.type("2");
  await page.keyboard.down("Enter");
}

// go back to the initial state
async function back(page /* Puppeteer page API */) {
  let el;
  el = await page.waitForSelector('body > section > div > section > ul > li:nth-child(1) > div > bu
  await el.evaluate(b => b.click());
  el = await page.waitForSelector('body > section > div > section > ul > li > div > button');
  await el.evaluate(b => b.click());
}

module.exports = {action, back, url};
```

github.com/stoyan/memlab-recorder

Spot the leak

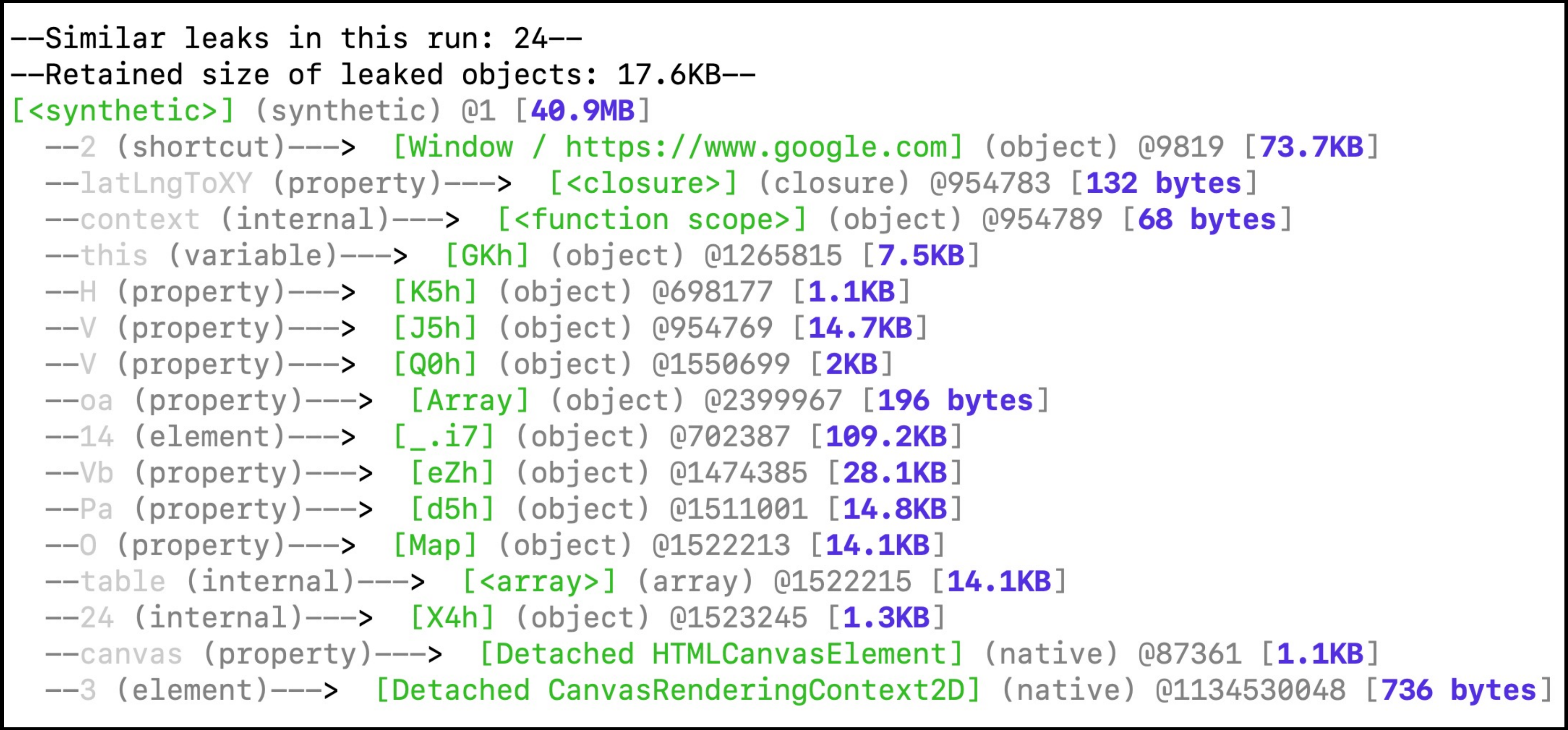
```
class Snoopy extends React.Component {
  constructor() { /* ... */ }

  componentDidMount() {
    document.addEventListener('keydown', e => {
      const keys = [...this.state.keys];
      keys.push(e.keyCode);
      this.setState({keys});
    });
  }

  render() {
    return (<p>/* ... */</p>);
  }
}
```

Spot the leak

```
class Snoopy extends React.Component {

  componentDidMount() {
    document.addEventListener('keydown', e => {
      const keys = [...this.state.keys];
      keys.push(e.keyCode);
      this.setState({keys});
    });
    setInterval(() => {
      const seconds = this.state.seconds + 1;
      this.setState({seconds});
    }, 1000);
  }

}
```

Detect

- https://phpied.com/files/snoopy/named/2.leak-interval.html
- Non-minified code



```
--Similar leaks in this run: 24--
--Retained size of leaked objects: 17.6KB--
[<synthetic>] (synthetic) @1 [40.9MB]
  --2 (shortcut)--->  [Window / https://www.google.com] (object) @9819 [73.7KB]
  --latLngToXY (property)--->  [<closure>] (closure) @954783 [132 bytes]
  --context (internal)--->  [<function scope>] (object) @954789 [68 bytes]
  --this (variable)--->  [GKh] (object) @1265815 [7.5KB]
  --H (property)--->  [K5h] (object) @698177 [1.1KB]
  --V (property)--->  [J5h] (object) @954769 [14.7KB]
  --V (property)--->  [Q0h] (object) @1550699 [2KB]
  --oa (property)--->  [Array] (object) @2399967 [196 bytes]
  --14 (element)--->  [_.i7] (object) @702387 [109.2KB]
  --Vb (property)--->  [eZh] (object) @1474385 [28.1KB]
  --Pa (property)--->  [d5h] (object) @1511001 [14.8KB]
  --O (property)--->  [Map] (object) @1522213 [14.1KB]
  --table (internal)--->  [<array>] (array) @1522215 [14.1KB]
  --24 (internal)--->  [X4h] (object) @1523245 [1.3KB]
  --canvas (property)--->  [Detached HTMLCanvasElement] (native) @87361 [1.1KB]
  --3 (element)--->  [Detached CanvasRenderingContext2D] (native) @1134530048 [736 bytes]
```

Detect

- https://phpied.com/files/snoopy/named/2.leak-interval.html
- Non-minified code
- Name our functions

```
class Snoopy extends React.Component {

    componentDidMount() {
        document.addEventListener('keydown', e => {
            const keys = [...this.state.keys];
            keys.push(e.keyCode);
            this.setState({keys});
        });
        setInterval(() => {
            const seconds = this.state.seconds + 1;
            this.setState({seconds});
        }, 1000);
    }

}
```

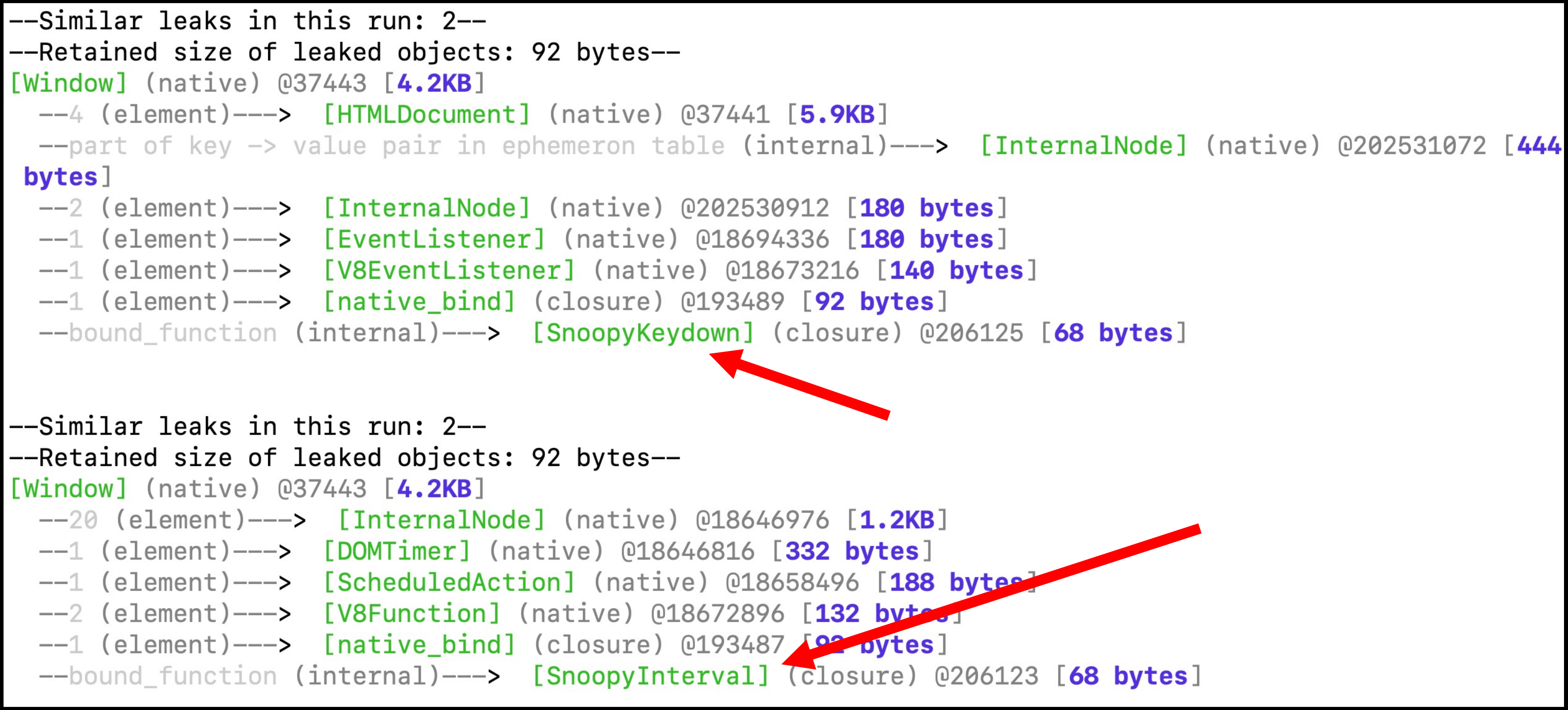
```
class Snoopy extends React.Component {

  componentDidMount() {
    document.addEventListener('keydown', function SnoopyKeydown(e) {
      const keys = [...this.state.keys];
      keys.push(e.keyCode);
      this.setState({keys});
    }.bind(this));
    setInterval(function SnoopyInterval() {
      const seconds = this.state.seconds + 1;
      this.setState({seconds});
    }.bind(this), 1000);
  }

}
```

Detect

- https://phpied.com/files/snoopy/named/2.leak-interval.html
- Non-minified code
- Name our functions
- Use Memlab recorder to create a scenario

Running Memlab

```
memlab run --scenario snoopy.js
           --verbose

memlab find-leaks --trace-all-objects
                  | grep Snoopy
```



```
--Similar leaks in this run: 2--
--Retained size of leaked objects: 92 bytes--
[Window] (native) @37443 [4.2KB]
  --4 (element)---> [HTMLDocument] (native) @37441 [5.9KB]
  --part of key -> value pair in ephemeron table (internal)---> [InternalNode] (native) @202531072 [444
 bytes]
  --2 (element)---> [InternalNode] (native) @202530912 [180 bytes]
  --1 (element)---> [EventListener] (native) @18694336 [180 bytes]
  --1 (element)---> [V8EventListener] (native) @18673216 [140 bytes]
  --1 (element)---> [native_bind] (closure) @193489 [92 bytes]
  --bound_function (internal)---> [SnoopyKeydown] (closure) @206125 [68 bytes]


--Similar leaks in this run: 2--
--Retained size of leaked objects: 92 bytes--
[Window] (native) @37443 [4.2KB]
  --20 (element)---> [InternalNode] (native) @18646976 [1.2KB]
  --1 (element)---> [DOMTimer] (native) @18646816 [332 bytes]
  --1 (element)---> [ScheduledAction] (native) @18658496 [188 bytes]
  --2 (element)---> [V8Function] (native) @18672896 [132 bytes]
  --1 (element)---> [native_bind] (closure) @193487 [92 bytes]
  --bound_function (internal)---> [SnoopyInterval] (closure) @206123 [68 bytes]
```

Plugging the leak

```
class Snoopy extends React.Component {

  componentDidMount() {
    document.addEventListener('keydown', this.snoopyKeydownHandler);
    this.intervalID = setInterval(function SnoopyInterval() {
      /* … */
    }.bind(this), 1000);
  }

  componentWillUnmount() {
    document.removeEventListener('keydown',
        this.snoopyKeydownHandler);
    clearInterval(this.intervalID);
}
```

Spot the leak

```
class Snoopy extends React.Component {

  componentDidMount() {
    document.addEventListener('keydown', this.snoopyKeydownHandler);
    this.intervalID = setInterval(function SnoopyInterval() {
      /* ... */
    }.bind(this), 1000);
  }

  componentWillUnMount() {
    document.removeEventListener('keydown',
        this.snoopyKeydownHandler);
    clearInterval(this.intervalID);
}
```

One more time, with hooks

Spot the leak

```
function Snoopy() {

  useEffect(()  => {
    function snoopyKeydownHandler() {/* … */}
    document.addEventListener('keydown', snoopyKeydownHandler);
  }, []);


  useEffect(()  => {
    setInterval(function SnoopyInterval() {
      /* … */
    }, 1000);
 }, []);
}
```

Plugging the leak

```
function Snoopy() {

  useEffect(()  => {
    function snoopyKeydownHandler() {/* … */}
    document.addEventListener('keydown', snoopyKeydownHandler);
    return () => document.removeEventListener('keydown’,
                    snoopyKeydownHandler);
  }, []);
  useEffect(()  => {
    const intervalID = setInterval(/* … */, 1000);
    return () => clearInterval(intervalID);
 }, []);
}
```

Wrapping up

// todo

- Integrate Reporting API
- Check `componentWillUnmount()`
- Check return values in `useEffect()/useLayoutEffect()`
- `null` discarded nodes and other objects
- Try Memlab and Memlab-recorder
- Leaks are hard to find, easy to plug

Big thank-yous!

- Benoit Girard @b56girard
- Liang Gong

Thank you!

@stoyanstefanov

@stoyan@indieweb.social

https://phpied.com