SANDDRILLER
A Fully-Automated Approach for Testing Language-Based JavaScript Sandboxes

Abdullah AlHamdan, Cristian-Alexandru Staicu | USENIX‘23 | AUG 2023
Motivation

- JavaScript is widely-used in security-critical scenarios
- JavaScript isolation provides **cheap security solution**
- **Scarcity of testing** JavaScript sandboxes, especially the server-side
What is a JavaScript Sandbox?

- It **limits** the **capabilities** of the third-party or untrusted code:
  - **Limits the access** to the global object and the built in functions
  - Offers a way to **share references** between the guest code and host code via **endowments**

```javascript
1:   const sandBox = require("realms-shim");
2:   const realm = sandBox.makeRootRealm();
3:   realm.evaluate(
4:     `log("foo");`,
5:     {log: console.log}
6:   );
```

Running guest code with endowments on a sandbox.
Security Objectives

SO1: Prevent read/write outside

SO2: Restrict accessing privileged operations

SO3: Prevent blocking the main loop

SO4: Prevent host process crash
<table>
<thead>
<tr>
<th>Sandbox</th>
<th>Type</th>
<th>Vulns.</th>
<th>$SO_1$</th>
<th>$SO_2$</th>
<th>$SO_3$</th>
<th>$SO_4$</th>
<th>Sandboxing strategy</th>
<th>Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>TreeHouse</td>
<td>C</td>
<td>0/0</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>worker threads with post messages</td>
<td>n/a</td>
</tr>
<tr>
<td>BreakApp</td>
<td>S</td>
<td>0/0</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>OS processes with IPC</td>
<td>n/a</td>
</tr>
<tr>
<td>jailed</td>
<td>C+S</td>
<td>0/0</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>OS processes with IPC</td>
<td>62</td>
</tr>
<tr>
<td>deno-vm</td>
<td>S</td>
<td>0/0</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>worker threads with Deno</td>
<td>173</td>
</tr>
<tr>
<td>isolated-vm</td>
<td>S</td>
<td>0/0</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>expose V8’s Isolate API</td>
<td>12,097</td>
</tr>
<tr>
<td>vm2</td>
<td>S</td>
<td>0/15</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>vm module and membranes</td>
<td>3,547,348</td>
</tr>
<tr>
<td>realms-shim</td>
<td>C+S</td>
<td>0/2</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>vm module and membranes</td>
<td>405</td>
</tr>
<tr>
<td>ses</td>
<td>C+S</td>
<td>0/0</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>membranes and frozen primordials</td>
<td>17,550</td>
</tr>
<tr>
<td>safe-eval</td>
<td>S</td>
<td>3/7</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>vm module and mutating the context</td>
<td>37,090</td>
</tr>
<tr>
<td>notevil</td>
<td>C+S</td>
<td>0/3</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>meta-circular interpreter</td>
<td>4,387</td>
</tr>
<tr>
<td>SandTrap</td>
<td>S</td>
<td>0/0</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>vm module and membranes</td>
<td>n/a</td>
</tr>
<tr>
<td>MIR</td>
<td>C+S</td>
<td>0/0</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>shadow builtins with wrappers</td>
<td>6</td>
</tr>
<tr>
<td>near-membrane</td>
<td>C+S</td>
<td>0/0</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>vm module and membranes</td>
<td>42</td>
</tr>
<tr>
<td>AdSafe</td>
<td>C</td>
<td>0/3</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>static checks and wrappers</td>
<td>n/a</td>
</tr>
<tr>
<td>Caja</td>
<td>C</td>
<td>0/2</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>code rewrite and frozen primordials</td>
<td>n/a</td>
</tr>
</tbody>
</table>

S: server-side sandbox  
C: client-side sandbox
Threats to Language-Based Sandbox

Sandbox escape via prototype chain gives the access to foreign references.
Detect security violations to security objectives

**SandDriller**

1st approach to test language-based sandboxes
Uses dynamic analysis

**SANDDRILLER**

Detect security violations to security objectives
Can be integrated in the development process of Language-Based Sandboxes

SandDriller

Detect security violations to security objectives
Methodology

Seed Code

Instrumentor: injects oracles to the seed code

Valid source code

Instrument and Generate

Generator: generates new variants from the given seed code

Test Cases

1x instrumented + Nx variants

Run on Sandbox

Testing oracles search for foreign references.

Results

Status of the analysis
function analysis(){
  ...
  analysis(global);
  analysis(variables);
  const p = analysis(eval("import('./foo');"));
  analysis(p.then(imported => {
    foo(imported)
  }));
}

const p = eval("import('./foo');");
p.then(imported => {
  foo(imported)
});
Oracle Checks

To test two main objectives of language-based isolation:

- Read/Write secret flag (SO1)
- Access/Call powerful API (SO2)
- Detect hard crash (SO3)
Experimental Setup
Experimental Setup

3X Node.js versions
14.15, 15.12, 16.12

2X Seed corpus
ECMAScript (41034)
v8 (5572)

2X Test rounds
With and without the variants generator

Server-side
vm2, realms-shim, ses safe-eval, near-membrane

Client-side
AdSafe
## Quantitative Results

Results of testing the sandboxes using SANDDRILLER across the selected Node.js versions

<table>
<thead>
<tr>
<th>Node.js version</th>
<th>Sandbox</th>
<th>Data set</th>
<th>Security error</th>
<th>Hard crash</th>
<th>Security error</th>
<th>Hard crash</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.15</td>
<td>vm2</td>
<td>ECMA</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V8</td>
<td>7</td>
<td>1</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>realms-shim</td>
<td>ECMA</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V8</td>
<td>12</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>safe-eval</td>
<td>ECMA</td>
<td>3154</td>
<td>2</td>
<td>6214</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V8</td>
<td>812</td>
<td>1</td>
<td>2079</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>ses</td>
<td>ECMA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>near-membrane</td>
<td>ECMA</td>
<td>2873</td>
<td>0</td>
<td>6877</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V8</td>
<td>793</td>
<td>1</td>
<td>2476</td>
<td>1</td>
</tr>
<tr>
<td>16.12</td>
<td>vm2</td>
<td>ECMA</td>
<td>1</td>
<td>2</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V8</td>
<td>8</td>
<td>0</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>realms-shim</td>
<td>ECMA</td>
<td>17</td>
<td>0</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V8</td>
<td>18</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>safe-eval</td>
<td>ECMA</td>
<td>3957</td>
<td>2</td>
<td>5531</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V8</td>
<td>475</td>
<td>0</td>
<td>1879</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ses</td>
<td>ECMA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>near-membrane</td>
<td>ECMA</td>
<td>3581</td>
<td>0</td>
<td>7160</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V8</td>
<td>626</td>
<td>0</td>
<td>2374</td>
<td>0</td>
</tr>
</tbody>
</table>
Qualitative Analysis Cont.: CVE-2021-23594

Realms-shim < 1.2.2

1: let realm = Realm.makeRootRealm();
2: try {
3:   realm.evaluate(`
4:   Error.prepareStackTrace().function(error, stackTrace) {
5:     stackTrace.__proto__.__proto__.polluted = "success";
6:   };
7: } catch(e) {
8:   'ERR_UNHANDLED_REJECTION' ape
9: } console.log(polluted); // output: 'success'
## Conclusions

### Security Objectives

- **SO1**: Prevents read/write outside
- **SO2**: Restricts accessing privileged operations
- **SO3**: Prevents blocking the main loop
- **SO4**: Prevents host process crash

### Workflows

- **Instrument**: Objects areki test case
- **Valid source code**: Generates new variants from the given source code
- **Test Cases**: Run on Sandbox
- **Results**: Checks for unwrapped points

### Sandbox Vulnerabilities

<table>
<thead>
<tr>
<th>Sandbox</th>
<th>Vulnerability</th>
<th>Target SO</th>
<th>Date of fixing</th>
<th>Details about the payloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>isolated-vm</td>
<td>CVE-2021-21413</td>
<td>SO1, SO2</td>
<td>12(^{th}) of February 2021</td>
<td>• capability leak</td>
</tr>
<tr>
<td></td>
<td>vm</td>
<td></td>
<td></td>
<td>• import keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12(^{th}) of October 2021</td>
<td>• custom stack traces</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• vm’s stack property issue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8(^{th}) of February 2022</td>
<td>• custom to(String()) method on listener objects</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>29(^{th}) of April 2020</td>
<td>• custom stack trace</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• import keyword</td>
</tr>
<tr>
<td>realms-shim</td>
<td>CVE-2021-23594</td>
<td>SO1, SO2</td>
<td>n/a</td>
<td>• custom stack trace</td>
</tr>
<tr>
<td>realms-shim</td>
<td>CVE-2021-23543</td>
<td>SO1, SO2</td>
<td>n/a</td>
<td>• vm’s stack property issue</td>
</tr>
<tr>
<td>SandTrap</td>
<td>GHSA-xx7r-mw56-3q2h</td>
<td>SO1, SO2, SO3</td>
<td>11(^{th}) of November 2021</td>
<td>• custom stack traces</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• import keyword</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• vm’s stack property issue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• bypass restriction on property names</td>
</tr>
<tr>
<td>jailed</td>
<td>CVE-2022-23923</td>
<td>SO2</td>
<td>n/a</td>
<td>• bypass restriction on powerful builtins</td>
</tr>
<tr>
<td>notevil</td>
<td>CVE-2021-23771</td>
<td>SO1</td>
<td>n/a</td>
<td>• bypass restriction on property names</td>
</tr>
</tbody>
</table>

---

**SandDriller’s source code**

@AbdullahMHamdan

abdullah.alhamdan@cispa.de
Study of Isolation Solutions for JavaScript

- Read sandboxes documentation
- Understand the sandboxes’ capabilities
- Study the sandboxes’ vulnerabilities/fixes
- Audit sandboxes’ source code
Seed corpus

- Contains self-contained JavaScript code,
- It should cover most of the language’s features.

```javascript
function foo() {
    try {
        throw "bar";
    } finally {
        return "baz";
    }
}

foo();
print({});
```

Can be found in:
`v8/api-call-after-bypassed-exception.js`

And more...
Variants Generation

Injects **hard-to-handle patterns** from previous sandbox breakouts function entrance.

#variations = #function definitions

```
function foo() {
    ...
    return "foo";
}
function bar() {
    ...
    return "bar";
}
foo();
bar();
```

---

Seed file

From existing exploits
**Sandbox runner**

- Runs multiple tests at once using a multi-process architecture,
- A fresh process for each test seed in pool of processes,
- Respawns processes when a hard crash is detected.

**Errors types**

<table>
<thead>
<tr>
<th>No Error</th>
<th>Runtime Error</th>
<th>Instrumentation Error</th>
<th>Security Error</th>
<th>Timeout</th>
<th>Hard crash</th>
<th>Memory Violation</th>
</tr>
</thead>
</table>

**Results and Logging**

<table>
<thead>
<tr>
<th>Seed Name</th>
<th>Tested Sandbox</th>
<th>Time Execution</th>
<th>Number of Oracle Checks</th>
<th>Instrumentation Error</th>
<th>Run Error</th>
<th>Security Error</th>
<th>Number of Variants</th>
</tr>
</thead>
</table>
Exploit Minimization: Delta Debugging

- Failure-inducing circumstances
  - Inputs, code, execution

- Minimal lines-of-code which triggers the same problem!
  - Vulnerability sink!
Qualitative Analysis: CVE-2021-23449

VM2 < 3.9.4

1: let code = ` 
2: p = eval('import("kscx");'); 
3: p.__proto__.proto.polluted = 'polluted'; 
4: `; 
5: const {VM} = require("vm2"); 
6: let vmInstance = new VM(); 
7: vmInstance.run(code); 
8: console.log(polluted); // output: 'polluted'

'ERR_UNHANDLED_REJECTION'

Exploit the prototype chain

Write outside the Sandbox
### Quantitative Results Cont.

<table>
<thead>
<tr>
<th>Sandbox</th>
<th>Data set</th>
<th>Without errors</th>
<th>Runtime error</th>
<th>Timeout</th>
<th>Security error</th>
<th>Hard crash</th>
<th>Memory corruption</th>
</tr>
</thead>
<tbody>
<tr>
<td>vm2</td>
<td>ECMA</td>
<td>111742</td>
<td>117315</td>
<td>35311</td>
<td>26</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>V8</td>
<td>15515</td>
<td>14904</td>
<td>6323</td>
<td>56</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ECMA</td>
<td>115750</td>
<td>119408</td>
<td>45008</td>
<td>60</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>V8</td>
<td>15085</td>
<td>15030</td>
<td>9710</td>
<td>55</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>realms-shim</td>
<td>ECMA</td>
<td>39821</td>
<td>115103</td>
<td>3842</td>
<td>27482</td>
<td>18</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>V8</td>
<td>3043</td>
<td>14749</td>
<td>738</td>
<td>80</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ECMA</td>
<td>13994</td>
<td>119364</td>
<td>999</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>V8</td>
<td>15173</td>
<td>14986</td>
<td>9740</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>safe-eval</td>
<td>ECMA</td>
<td>85410</td>
<td>112964</td>
<td>37414</td>
<td>29926</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>V8</td>
<td>8053</td>
<td>14706</td>
<td>7064</td>
<td>9644</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ECMA</td>
<td>1999</td>
<td>20219</td>
<td>4419</td>
<td>44441</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>V8</td>
<td>516</td>
<td>7536</td>
<td>6589</td>
<td>846</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ses</td>
<td>ECMA</td>
<td>113834</td>
<td>119364</td>
<td>999</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>V8</td>
<td>15173</td>
<td>14986</td>
<td>9740</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>near-membrane</td>
<td>ECMA</td>
<td>85410</td>
<td>112964</td>
<td>37414</td>
<td>29926</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>V8</td>
<td>8053</td>
<td>14706</td>
<td>7064</td>
<td>9644</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>ADSafe</td>
<td>ECMA</td>
<td>1999</td>
<td>20219</td>
<td>4419</td>
<td>44441</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>V8</td>
<td>516</td>
<td>7536</td>
<td>6589</td>
<td>846</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Results of testing the sandboxes using SANDDRILLER across the selected Node.js versions.
Sandbox runner

- Runs multiple tests at once using a multi-process architecture
- A fresh process for each test seed in pool of processes
- Respawns processes when a hard crash is detected.
Exceptions from functions internals **might** not be wrapped

```javascript
try {
    function foo() {
        throw function thrower() {
            return () => {
                return this;
            };
        }();
        ...
        return "foo";
    }
    foo();
    catch (e) {
        analysis(this);
        analysis(e);
    }
} catch (e) {
    analysis(e);
}
```