CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

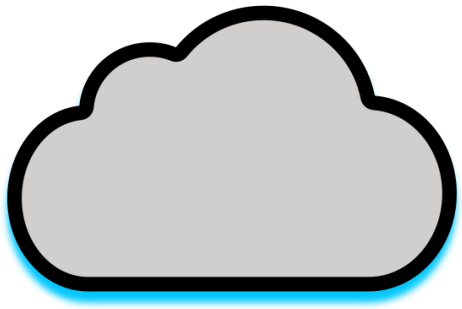
SandDriller

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

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

Motivation

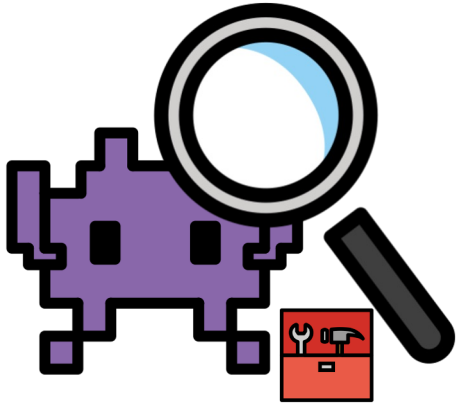
- JavaScript is widely-used in security-critical scenarios
- JavaScript isolation provides **cheap security solution**
- **Scarcity of testing** JavaScript sandboxes, especially the server-side



Cloud
Zscaler

IoT
NodeRED
Moddable XS

Blockchain
Embark



Malware Analysis:
Box-js
Intel owl

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**

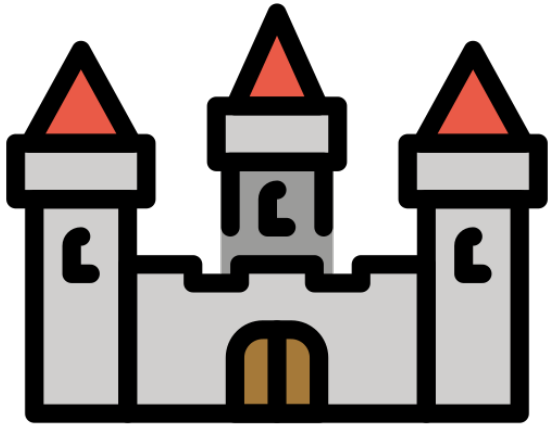
```
1:  const sandBox = require("realms-shim");
2:  const realm = sandBox.makeRootRealm();
3:  realm.evaluate(
4:        `log("foo");`
5:        , {log : console.log}
6:  );
```

Guest code

Endowments

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

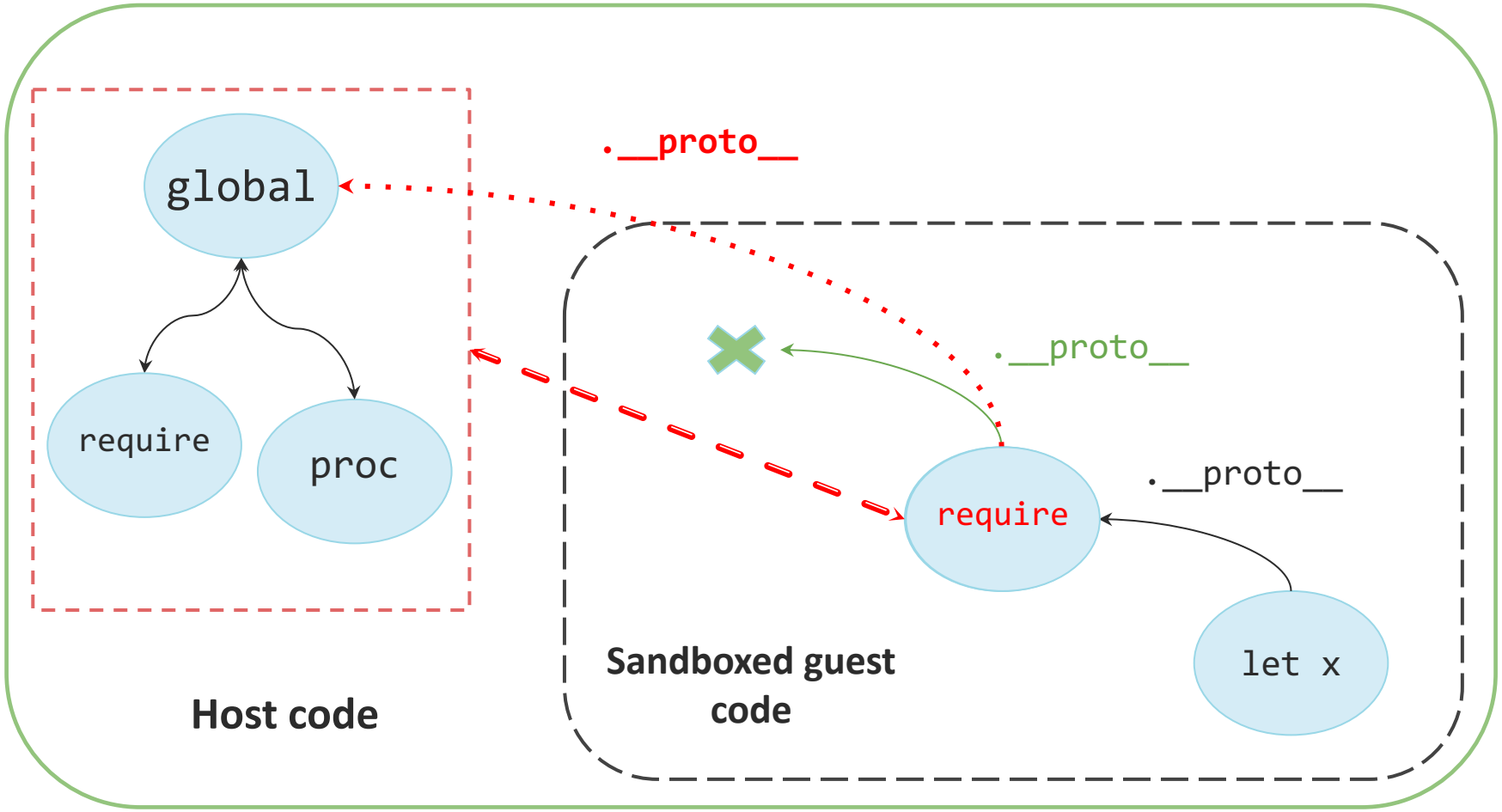
Study of Isolation Solutions for JavaScript (Cont.)

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

S: server-side sandbox
C: client-side sandbox

Threats to Language-Based Sandbox

Sandbox escape via prototype chain gives the access to *foreign references*



1st approach to test language-based sandboxes

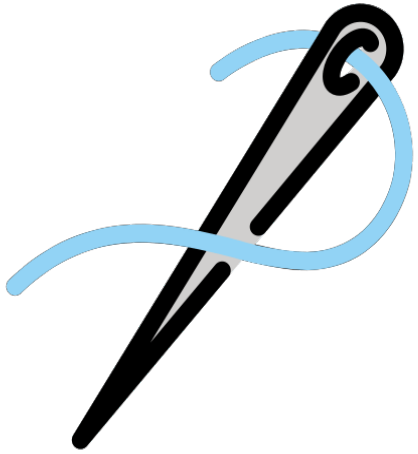
SandDriller

Detect security violations to security objectives

Uses dynamic analysis

SandDriller

Detect security violations to security objectives



Can be integrated in the development process of Language-Based Sandboxes

SandDriller

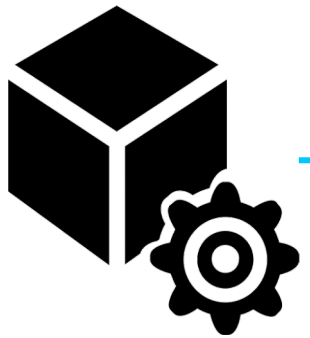
Detect security violations to security objectives

Methodology

Instrumentor: injects oracles to the seed code

1x instrumented
+
Nx variants

Status of the analysis



Seed Code

Instrument and Generate

Test Cases

Run on Sandbox

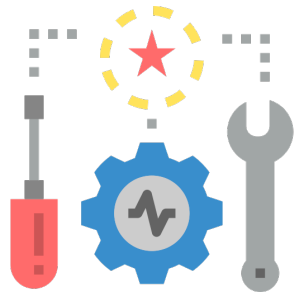
Results

Valid source code

Generator: generates new variants from the given seed code

Testing oracles search for foreign references.

Instrumentation



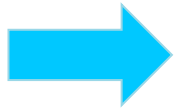
Injects code that tests all potential foreign references.

i.e., **global** scope, **function invocation**, **function entrance**, **try** .. **catch** clause, **throw**, **exceptions**, **this** ... etc.

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

Simplified

ECMA/language/expressions/dynamic-import/usage-from-eval.js



```
function analysis(){
...
}
analysis(global);
analysis(variables);
const p = analysis(eval("import('./foo');"));
analysis(p.then(imported => {
    analysis(this);
    analysis(arguments);
    analysis(foo(imported));
}));
```

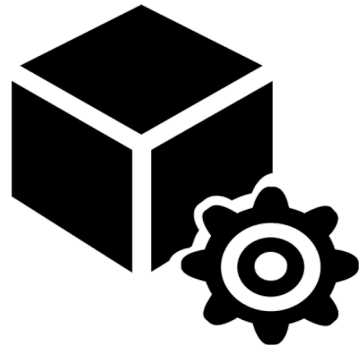
Analysis function(s)

Global object/variables

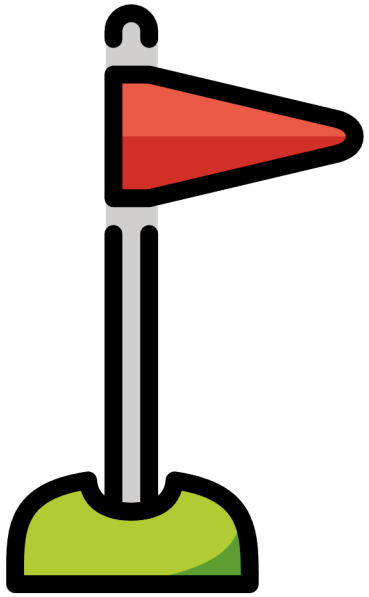
Function invocation

Function entrance

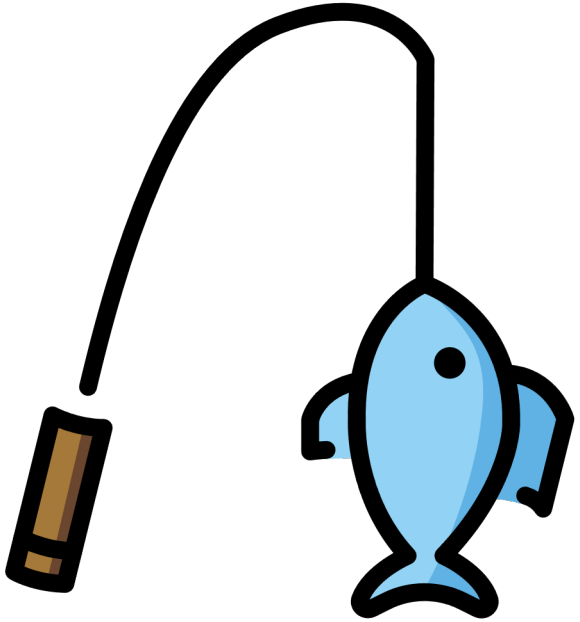
Oracle Checks



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

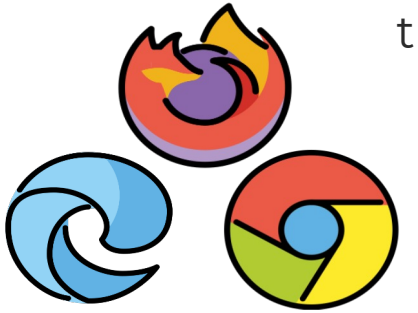
Experimental Setup

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

			Original corpus		Variants	
Node.js version	Sandbox	Data set	Security error	Hard crash	Security error	Hard crash
14.15	vm2	ECMA	1	2	6	4
		V8	7	1	11	1
	realms-shim	ECMA	4	0	8	0
		V8	12	0	4	0
	safe-eval	ECMA	3154	2	6214	4
		V8	812	1	2079	1
	ses	ECMA	0	0	0	0
		V8	0	0	0	0
	near-membrane	ECMA	2873	0	6877	0
		V8	793	1	2476	1
16.12	vm2	ECMA	1	2	13	4
		V8	8	0	12	0
	realms-shim	ECMA	17	0	15	0
		V8	18	0	5	0
	safe-eval	ECMA	3957	2	5531	4
		V8	475	0	1879	0
	ses	ECMA	0	0	0	0
		V8	0	0	0	0
	near-membrane	ECMA	3581	0	7160	0
		V8	626	0	2374	0

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

Qualitative Analysis Cont.: CVE-2021-23594

DEPRECATED

Realms-shim < 1.3.2

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

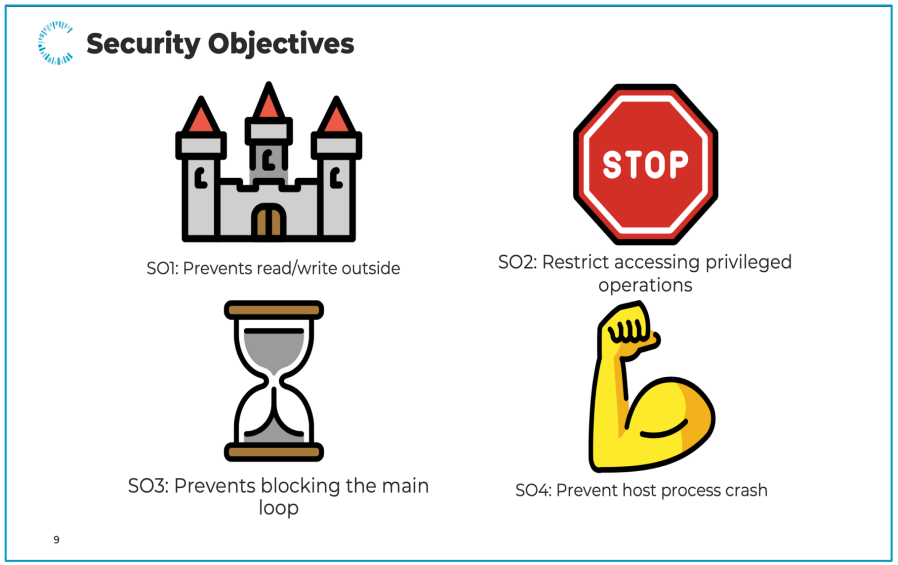
Overwrite

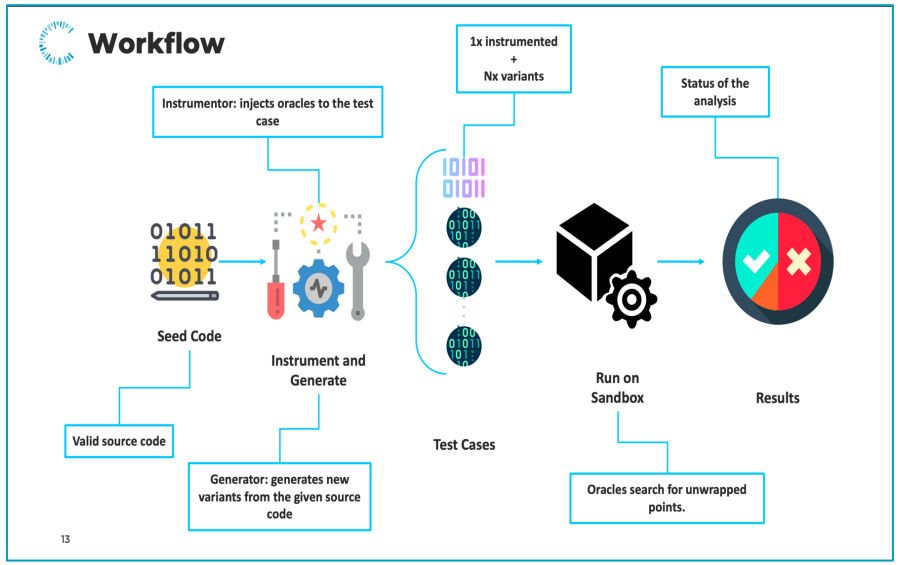
Escaping via foreign reference

'ERR_UNHANDLED_REJECTION'

cape

Conclusions





Sandbox	Vulnerability	Target SO	Date of fixing	Details about the payloads
`isolated-vm`	CVE-2021-21413	**SO1, SO2**	12th of February 2021	• capability leak
`vm2`	CVE-2021-23449	**SO1, SO2, SO3**	12th of October 2021	• `import` keyword • custom stack traces
`vm2`	CVE-2021-23555	**SO1, SO2**	8th of February 2022	• `vm`'s stack property issue
`vm2`	Issue #285	**SO1, SO2**	29th of April 2020	• custom `toString()` method on listener objects
`realms-shim`	CVE-2021-23594	**SO1, SO2**	n/a	• custom stack trace
`realms-shim`	CVE-2021-23543	**SO1, SO2**	n/a	• `vm`'s stack property issue
`SandTrap`	GHSA-xx7r-mw56-3q2h	**SO1, SO2, SO3**	11th of November 2021	• `import` keyword • `vm`'s stack property issue • custom stack traces
`jailed`	CVE-2022-23923	**SO2**	n/a	• direct access to powerful builtins
`notevil`	CVE-2021-23771	**SO1**	n/a	• bypass restriction on property names

SandDriller's source code

@AbdullahMHamdan
abdullah.alhamdan@cispa.de

Study of Isolation Solutions for JavaScript

Read sandboxes documentation

Study the sandboxes vulnerabilities/fixes

Understand the sandboxes' capabilities

Audit sandboxes' source code

Seed corpus

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

```
1: function foo() {
2:   try {
3: throw "bar";
4:   } finally {
5: return "baz";
6:   }
7: }
8: foo();
9: print({});
```

Function definition

Throw/Exceptions

try, catch, finally

Function invocation

Empty object

A v8 test

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

And more...

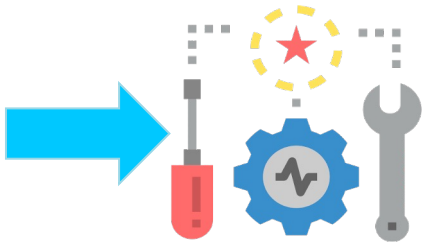
Variants Generation

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

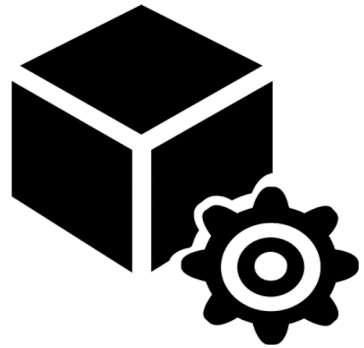


```
try {
function foo() {
    throw function thrower() {
        return () => {
            return this;
        };
    }();
    ...
    return "foo";
}
foo();
} catch (e) {
analysis(this);
analysis(e);
}
```

From existing exploits

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

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

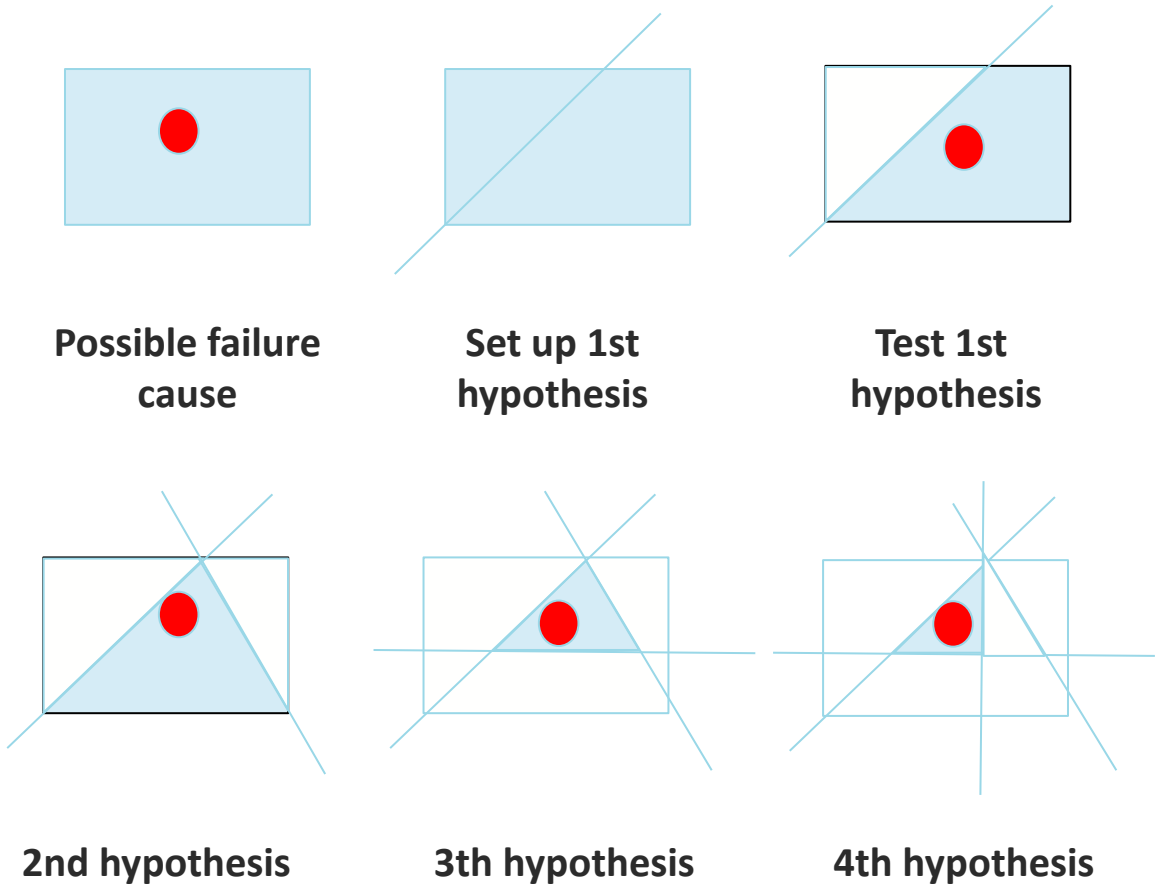
No Error	Runtime Error	Instrumentation Error	Security Error	Timeout	Hard crash	Memory Violation

Results and Logging

Seed Name	Tested Sandbox	Time Execution	Number of Oracle Checks	Instrumentation Error	Run Error	Security Error	Number of Variants

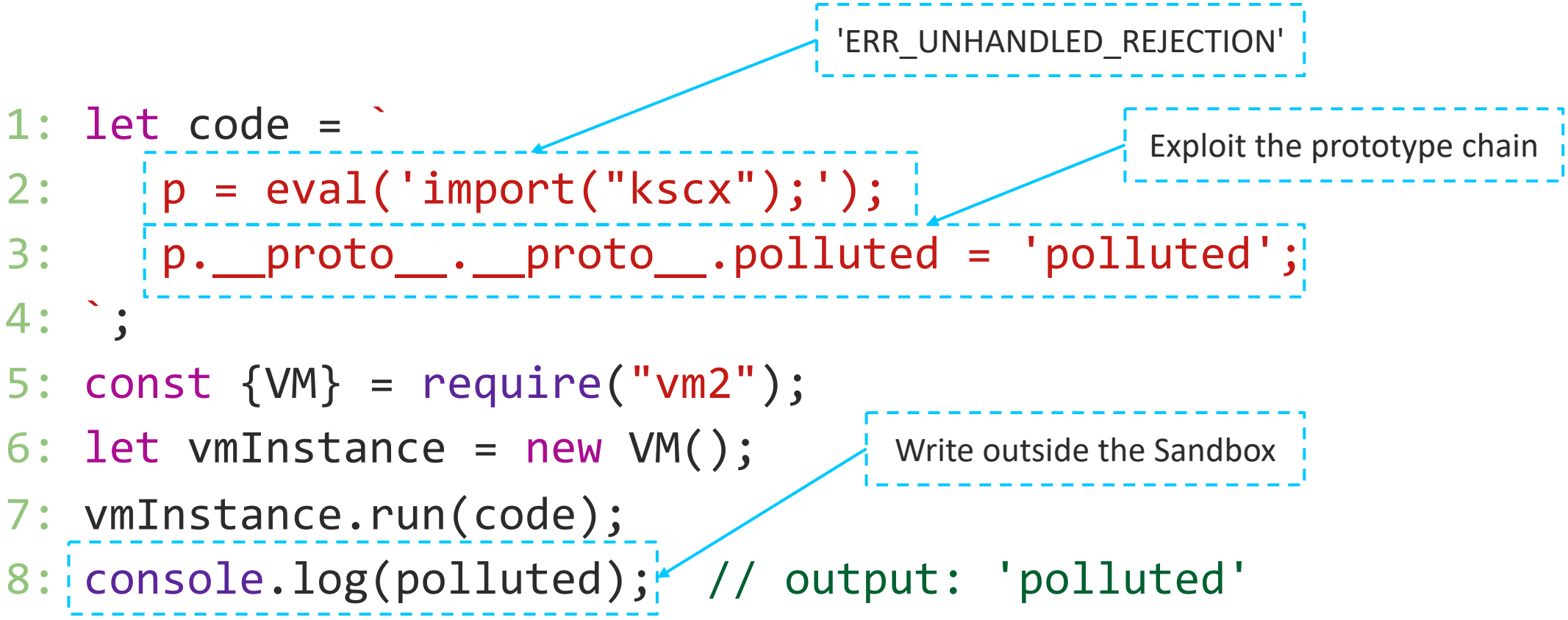
Exploit Minimization: Delta Debugging

- 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



Quantitative Results Cont.

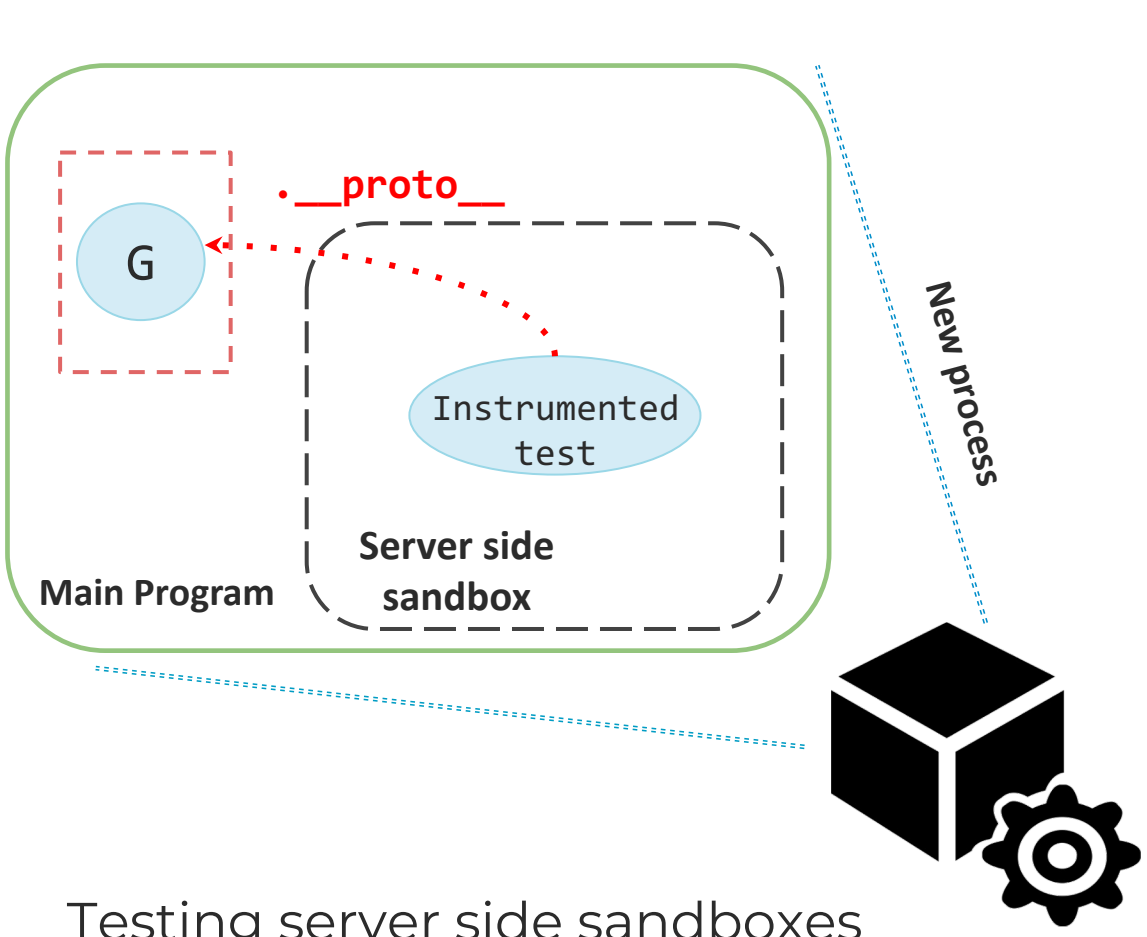
Sandbox	Data set	Without errors	Runtime error	Timeout	Security error	Hard crash	Memory corruption
vm2	ECMA	111742	117315	35311	26	18	6
	V8	15515	14904	6323	56	4	0
realms-shim	ECMA	115750	119408	45008	60	0	6
	V8	15085	15030	9710	55	0	0
safe-eval	ECM	[illegible]	[illegible]	3842	27482	18	6
	V8	943	14749	73[illegible]	80	4	0
ses	ECMA	[illegible]	[illegible]	[illegible]	0	0	6
	V8	15173	14986	9740	0	0	0
near-membrane	ECMA	85410	112964	37414	29926	0	6
	V8	8053	14706	7064	9644	4	0
ADSafe	ECMA	1999	20219	4419	44441	0	0
	V8	516	7536	6589	846	0	0

9X CVEs

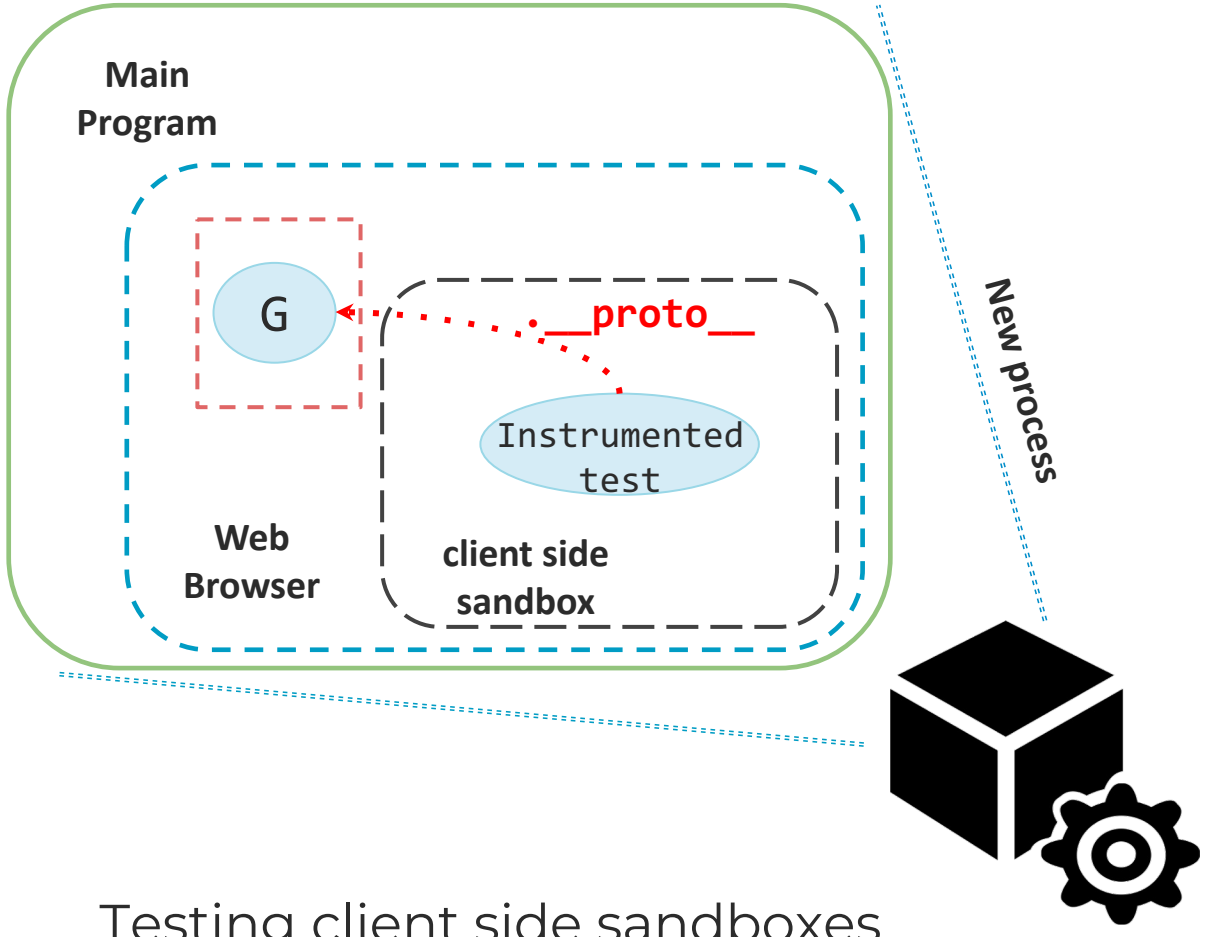
Results of testing the sandboxes using SANDDRILLER across the selected Node.js versions

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.



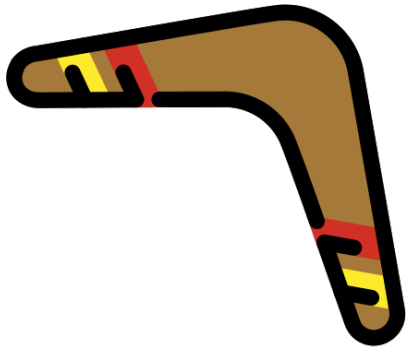
Testing server side sandboxes



Testing client side sandboxes

Variants Generation

Exceptions from functions internals **might** not be wrapped



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