

Advancements in JavaScript Engine Fuzzing

Finding cool bugs with little compute - OffensiveCon'23

Carl Smith, Samuel Groß - V8 Security

Fuzzilli

A (coverage-)guided fuzzer for dynamic language interpreters based on a custom intermediate language ("FuzzIL") which can be mutated and translated to JavaScript.

 fuzzilli.io

Fuzzilli Recap

```
v0 <- BeginPlainFunction -> v1
  v2 <- CreateArray [v1, v1, v1]
  v3 <- LoadInt '1'
  v4 <- CallMethod 'slice', v2, [v3]
  Return v4
EndPlainFunction
v5 <- LoadFloat '13.37'
v6 <- CallFunction v0, [v5]
```

Fuzzilli Recap

```
v0 <- BeginPlainFunction -> v1
  v2 <- CreateArray [v1, v1, v1]
  v3 <- LoadInt '1'
  v4 <- CallMethod 'slice', v2, [v3]
  Return v4
EndPlainFunction
v5 <- LoadFloat '13.37'
v6 <- CallFunction v0, [v5]
```



```
v0 <- BeginPlainFunction -> v1
  v2 <- CreateArray [v1, v1, v1]
  v4 <- LoadInt '100'
  SetProperty v2, 'length', v4
  v5 <- CallMethod 'slice', v2, [v1]
  Return v5
EndPlainFunction
v6 <- LoadFloat '42.0'
v7 <- CallFunction v0, [v5]
```

Splicing

Program 1

```
...  
v21 <- BeginPlainFunction -> v22, v23  
...  
  
EndPlainFunction  
...
```

Program 2

```
v0 <- BeginPlainFunction -> v1  
v2 <- CreateArray [v1, v1, v1]  
v3 <- LoadInt '1'  
v4 <- CallMethod 'slice', v2, [v3]  
Return v4  
EndPlainFunction  
v5 <- LoadFloat '13.37'  
v6 <- CallFunction v0, [v5]
```

Splicing

Program 1

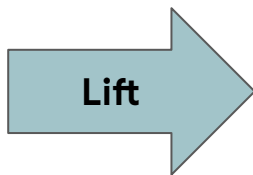
```
...  
v21 <- BeginPlainFunction -> v22, v23  
...  
v35 <- CreateArray [v23, v23, v23]  
v36 <- LoadInt '1'  
v37 <- CallMethod 'slice' v35, [v36]  
Return v37  
EndPlainFunction  
...
```

Program 2

```
v0 <- BeginPlainFunction -> v1  
v2 <- CreateArray [v1, v1, v1]  
v3 <- LoadInt '1'  
v4 <- CallMethod 'slice', v2, [v3]  
Return v4  
EndPlainFunction  
v5 <- LoadFloat '13.37'  
V6 <- CallFunction v0, [v5]
```

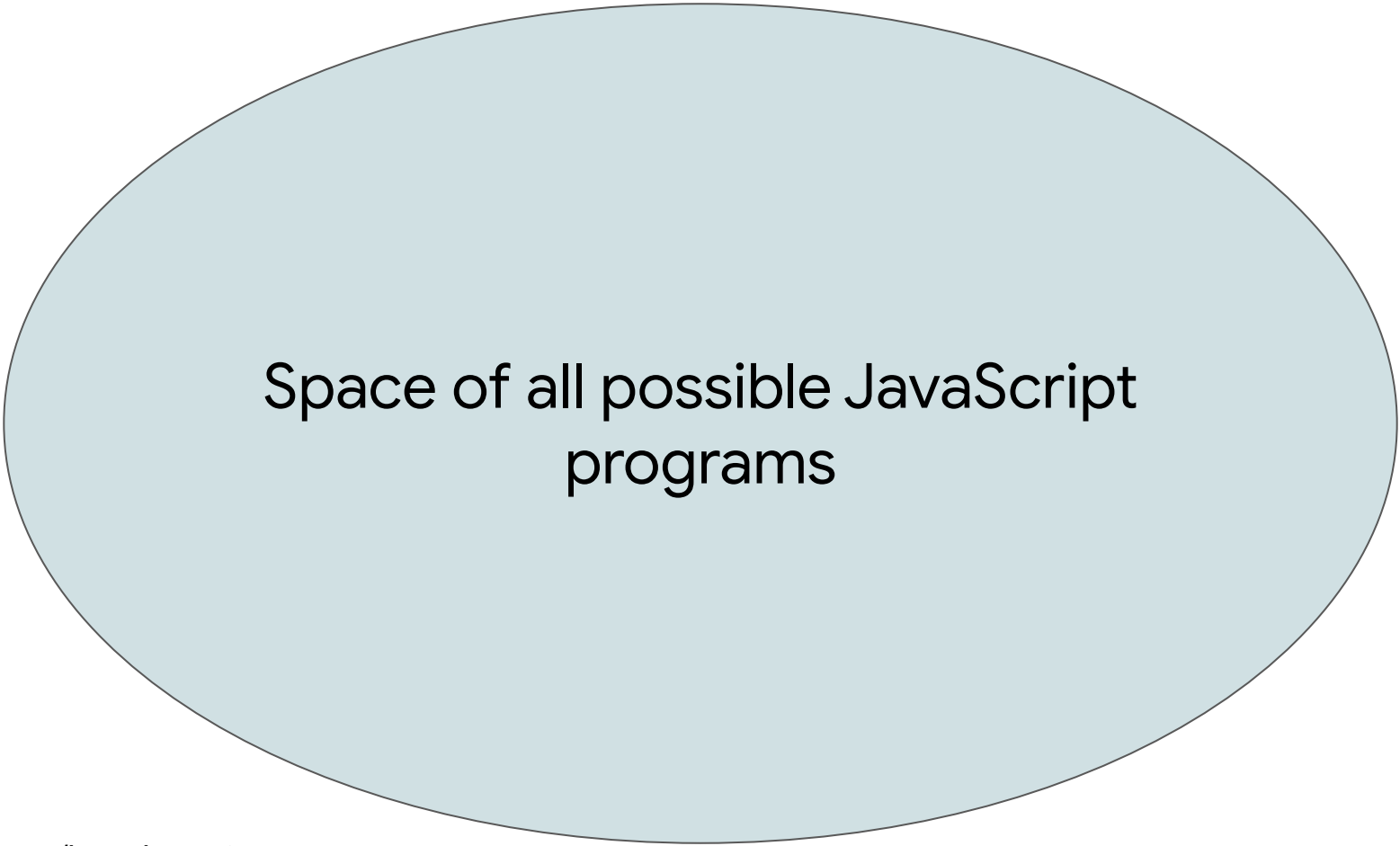
Fuzzilli Recap

```
v0 <- BeginPlainFunction -> v1
  v2 <- CreateArray [v1, v1, v1]
  v3 <- LoadInt '1'
  v4 <- CallMethod 'slice', v2, [v3]
  Return v4
EndPlainFunction
v5 <- LoadFloat '13.37'
v6 <- CallFunction v0, [v5]
```

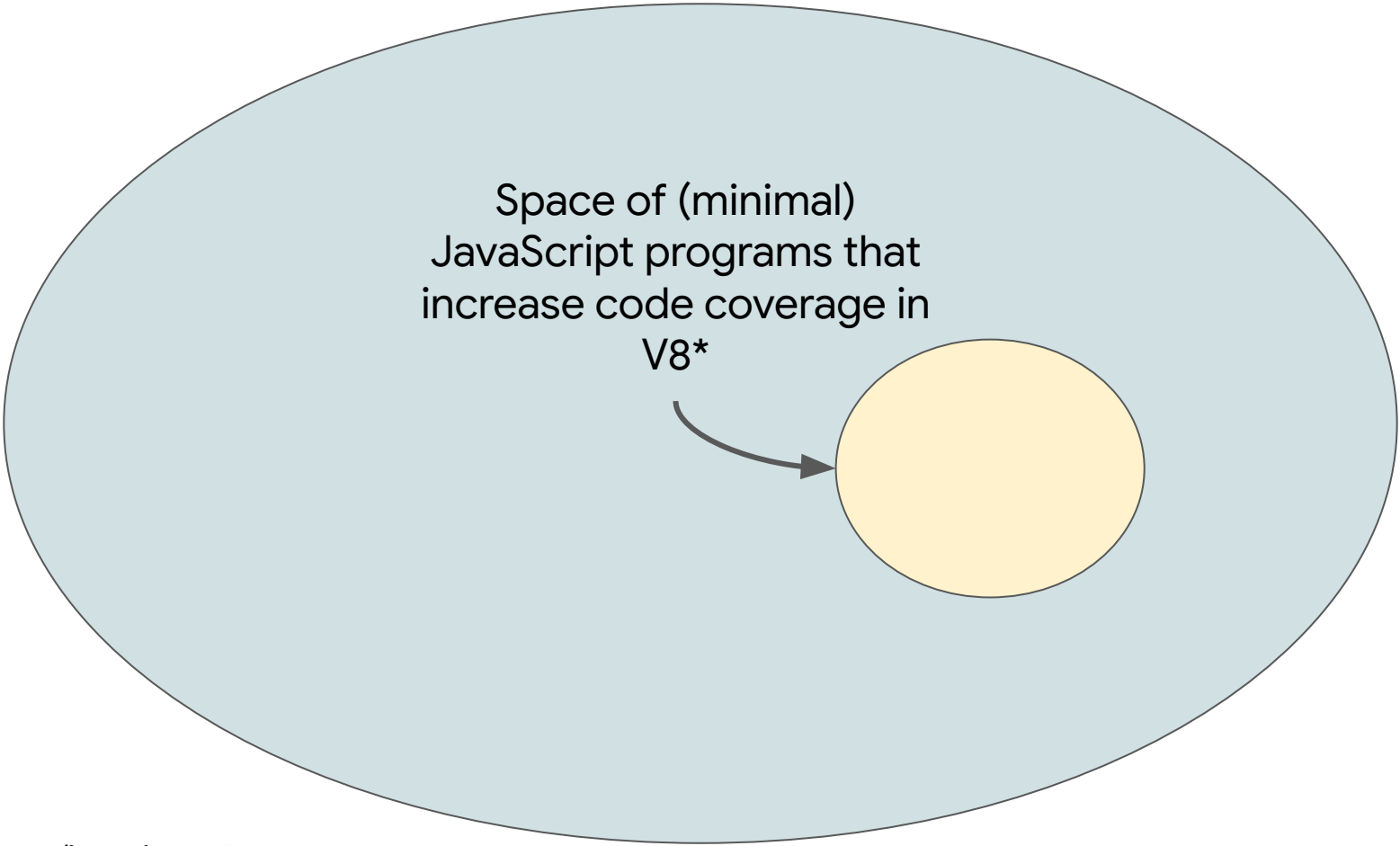


```
function f0(v1) {
  const v2 = [v1, v1, v1];
  return v2.slice(1);
}
f0(13.37);
```

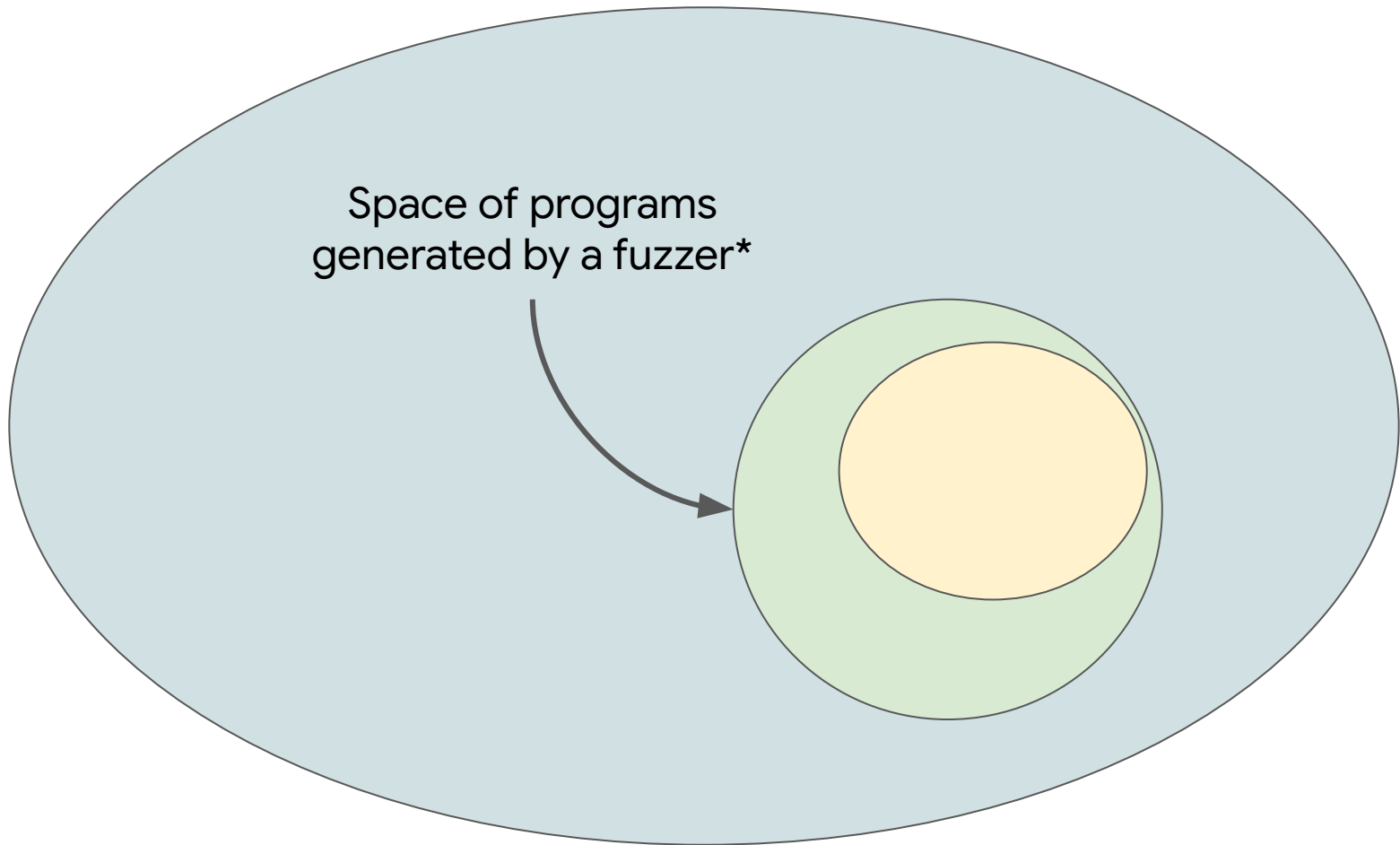
This finds bugs, but not enough...

A large, light blue oval with a thin black border, centered on a white background. Inside the oval, the text "Space of all possible JavaScript programs" is written in a black, sans-serif font, centered horizontally and vertically.

Space of all possible JavaScript
programs

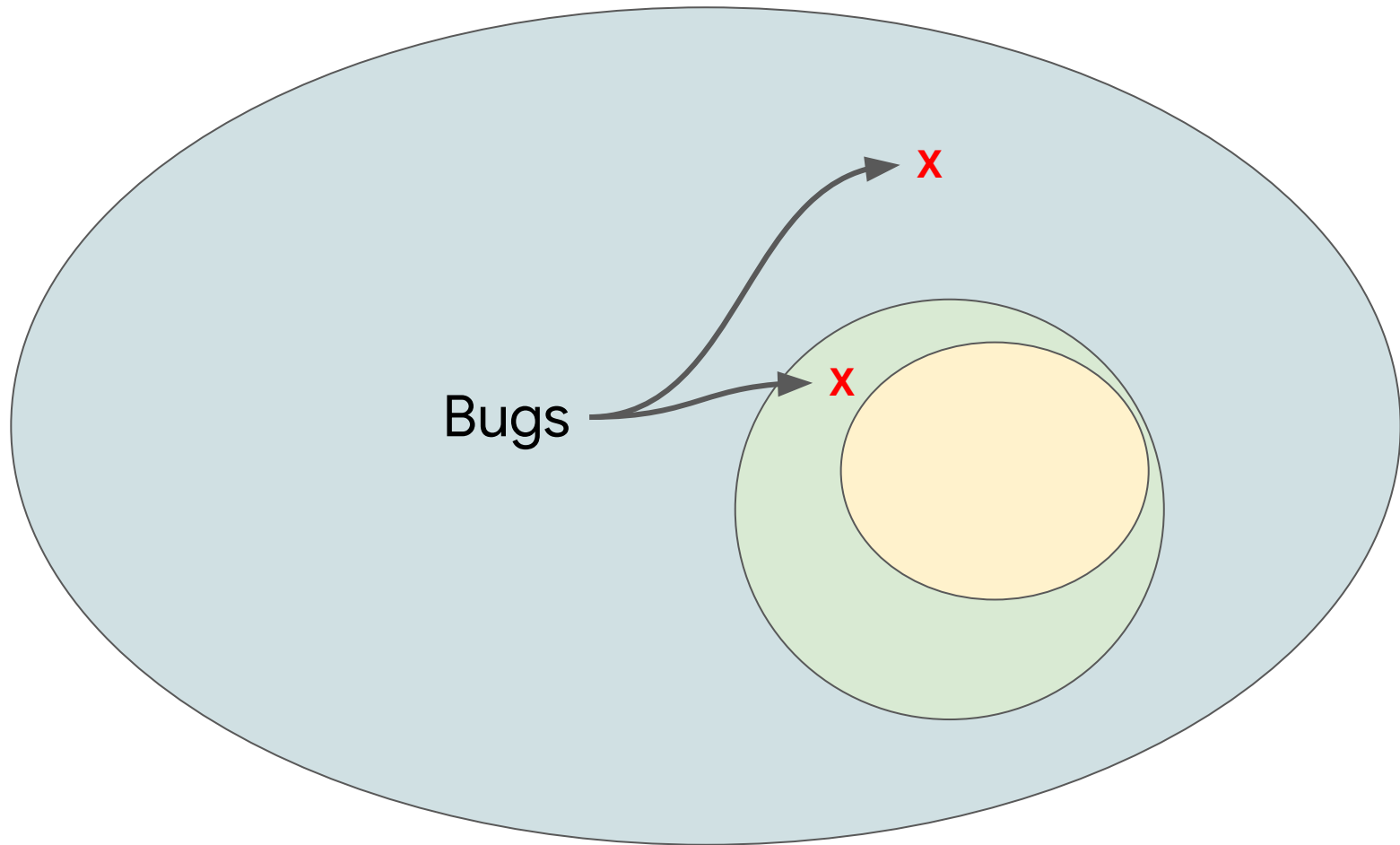


Space of (minimal)
JavaScript programs that
increase code coverage in
V8*

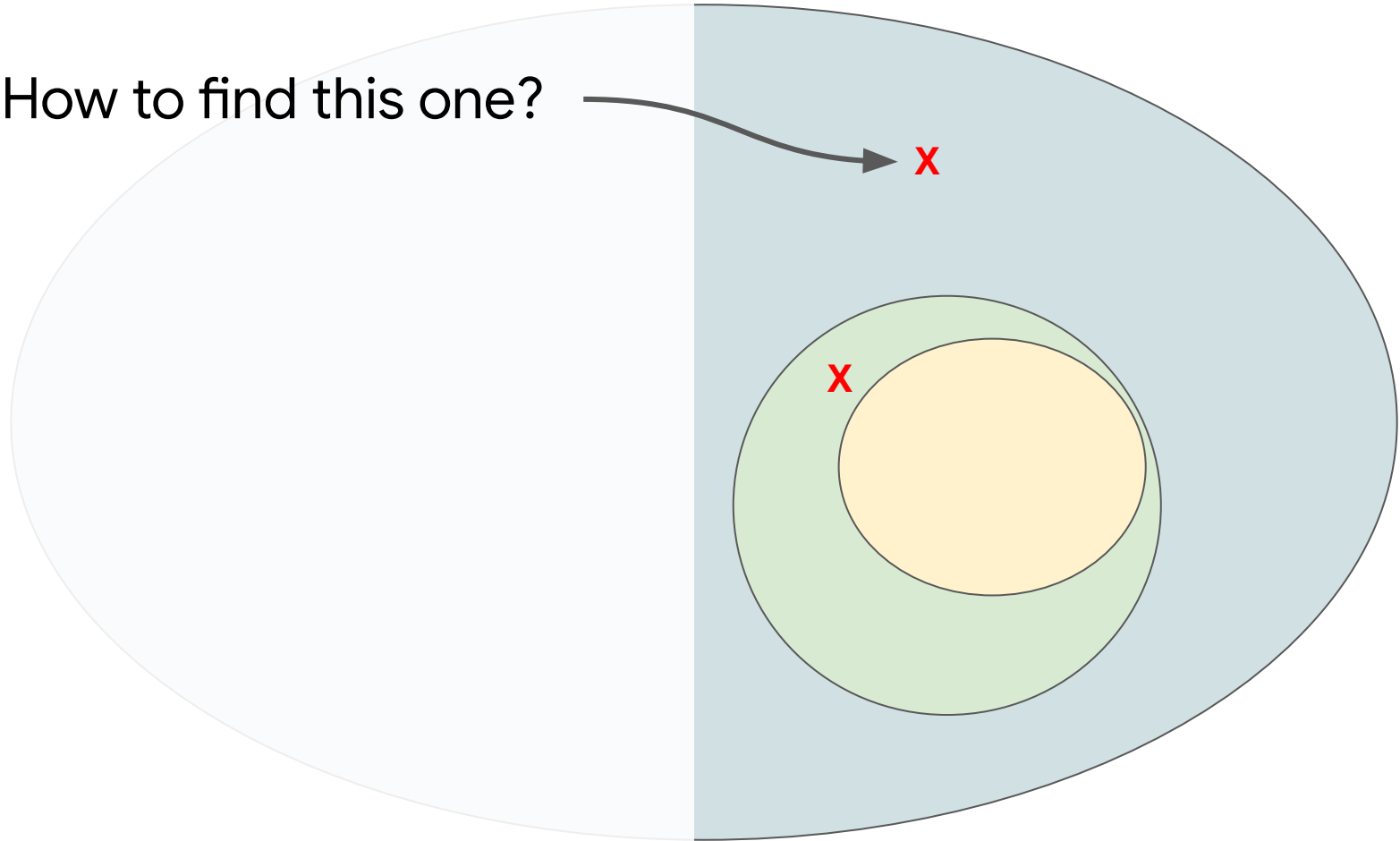


<https://t.me/learningnets>

* Basically, one mutation away from the corpus



How to find this one?



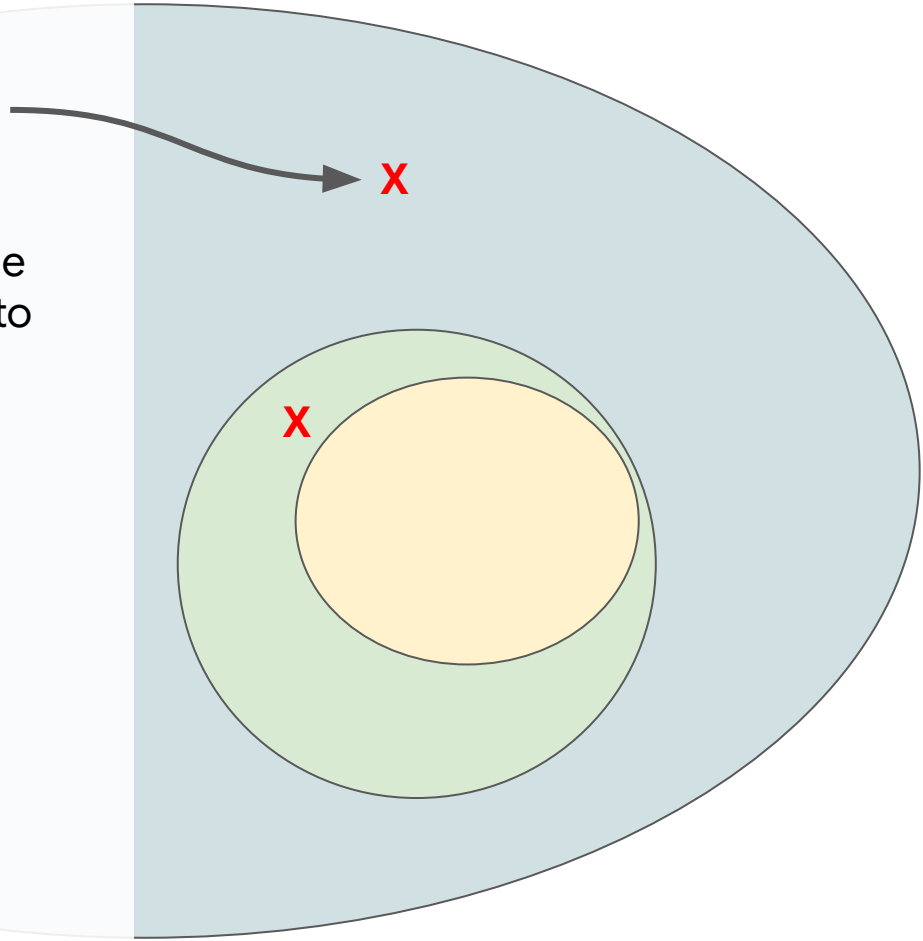
How to find this one?

- Import existing JavaScript code for mutation, hope it's "close" to the bug
 - Now possible with new JavaScript -> FuzzIL compiler!



How to find this one?

- Import existing JavaScript code for mutation, hope it's "close" to the bug
 - Now possible with new JavaScript -> FuzzIL compiler!
- Use different feedback
 - Future research topic?



How to find this one?

- Import existing JavaScript code for mutation, hope it's "close" to the bug
 - Now possible with new JavaScript -> FuzzIL compiler!
- Use different feedback
 - Future research topic?
- Use specialized mutators
 - To "hint" fuzzer towards known bug patterns



CVE-2016-4622

```
let a = [];  
for (let i = 0; i < 100; i++) a.push(i + 0.123);  
let evil = { valueOf() {  
    a.length = 0; return 10;  
}};  
  
// Triggers valueOf callback of evil and unexpectedly  
// shrinks the Array, leading to an OOB access  
let b = a.slice(0, evil);  
  
// b = [0.123,1.123,2.12199579146e-313,0,0,0,0,0,0,0]
```

CVE-2016-4622

```
let a = [];  
for (let i = 0; i < 100; i++) a.push(i + 0.123);  
let evil = { valueOf() {  
    a.length = 0; return 10;  
}};  
  
// Triggers valueOf callback of evil and unexpectedly  
// shrinks the Array, leading to an OOB access  
let b = a.slice(0, evil);  
  
// b = [0.123, 1.123, 2.12199579146e-313, 0, 0, 0, 0, 0, 0, 0]
```

CVE-2016-4622

```
let a = [];  
for (let i = 0; i < 100; i++) a.push(i + 0.123);  
let evil = { valueOf() {  
    a.length = 0; return 10;  
}};
```

Fuzzer is rewarded for finding these *individually*, but **not for combining** them!

```
// Triggers valueOf callback of evil and unexpectedly  
// shrinks the Array, leading to an OOB access  
let b = a.slice(0, evil);  
  
// b = [0.123,1.123,2.12199579146e-313,0,0,0,0,0,0,0]
```

Probing Mutator

Probing Mutator

```
let v1 = {};
```

```
// How is v1 being used?
```

```
builtin_func(v1);
```

Probing Part 1: Intermediate Program

```
let v1 = {};  
  
// How is v1 being used?  
  
// Let's find out!  
  
probe(v1);  
  
builtin_func(v1);
```

Probing Part 1: Intermediate Program

```
let v1 = {};
```

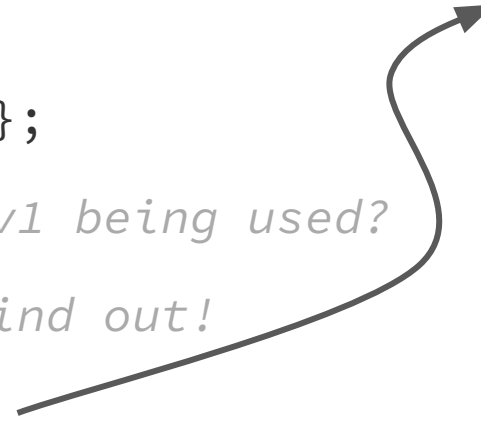
```
// How is v1 being used?
```

```
// Let's find out!
```

```
probe(v1);
```

```
builtin_func(v1);
```

```
function probe(v) {  
    // Turn |v| into a JS Proxy that  
    // records all property loads  
    // (and more), then sends that  
    // information back to Fuzzilli.  
}
```

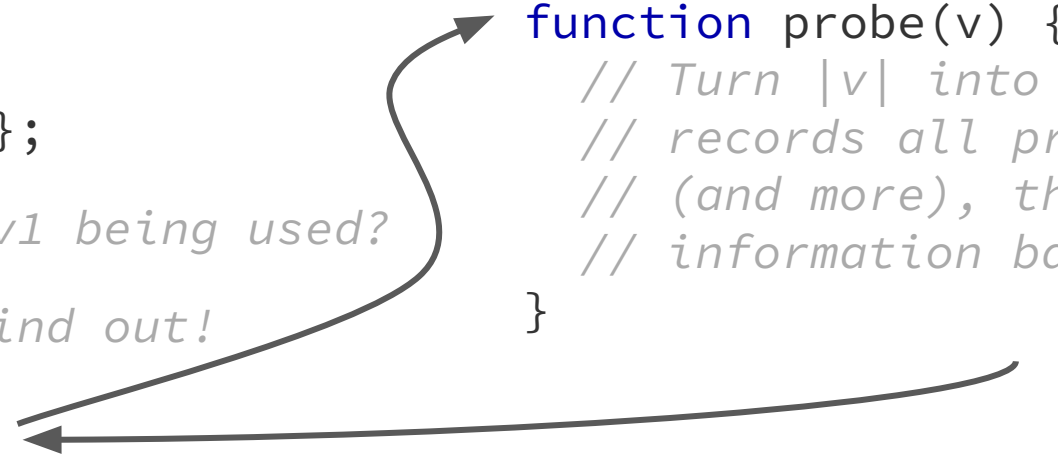


Probing Part 1: Intermediate Program

```
let v1 = {};  
// How is v1 being used?  
// Let's find out!  
probe(v1);  
builtin_func(v1);
```

```
function probe(v) {  
    // Turn |v| into a JS Proxy that  
    // records all property loads  
    // (and more), then sends that  
    // information back to Fuzzilli.  
}
```

Load .valueOf from v1

A diagram consisting of two curved arrows. One arrow starts at the `probe(v1);` line in the left column and points to the opening curly brace of the `function probe(v) {` definition in the right column. A second arrow starts from the text `Load .valueOf from v1` and points to the `probe(v1);` line.

Probing Part 2: Final Program

```
let v1 = {};  
function v2() {  
    ...;  
}  
v1.valueOf = v2;  
builtin_func(v1);
```

crbug.com/1381064 (and a couple other, similar bugs)

```
const v8 = new ArrayBuffer(1050, {"maxLength":6623679});
```

```
const v10 = new Uint8ClampedArray(v8);
```

```
function v11() {
```

```
    const v15 = v8.resize();
```

```
}
```

```
v10[Symbol.toPrimitive] = v11;
```

```
// Triggers toPrimitive conversion and unexpectedly shrinks the
```

```
// ArrayBuffer, leading to a (harmless) OOB access.
```

```
v10[916] = v10;
```

<https://t.me/learningnets>

crbug.com/1381064 (and a couple other, similar bugs)

```
const v8 = new ArrayBuffer(1050, {"maxLength":6623679});
```

```
const v10 = new Uint8ClampedArray(v8);
```

```
function v11() {
```

```
    const v15 = v8.resize();
```

```
}
```

```
v10[Symbol.toPrimitive] = v11;
```

```
// Triggers toPrimitive conversion and unexpectedly shrinks the
```

```
// ArrayBuffer, leading to a (harmless) OOB access.
```

```
v10[916] = v10;
```

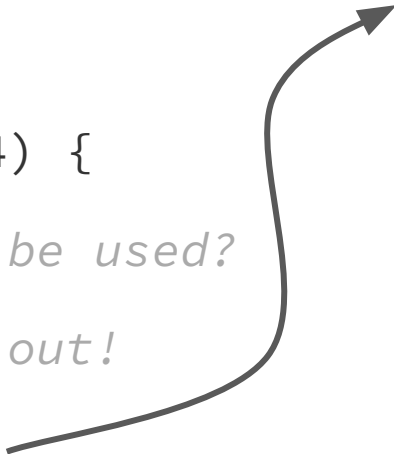
<https://t.me/learningnets>

Exploration

```
function f2(v3, v4) {  
    // How can v3 be used?  
}
```

Exploration (Step 1)

```
function f2(v3, v4) {  
    // How can v3 be used?  
    // Let's find out!  
    explore(v3);  
}
```



```
function explore(v) {  
    // Determine type of |v|  
    // using the typeof operator  
    // and enumerate all fields  
    // and methods, then pick a  
    // random "action", e.g. a  
    // property load, to perform.  
}
```

Exploration (Step 1)

```
function f2(v3, v4) {  
    // How can v3 be used?  
    // Let's find out!  
    explore(v3);  
}
```

```
function explore(v) {  
    // Determine type of |v|  
    // using the typeof operator  
    // and enumerate all fields  
    // and methods, then pick a  
    // random "action", e.g. a  
    // property load, to perform.  
}
```

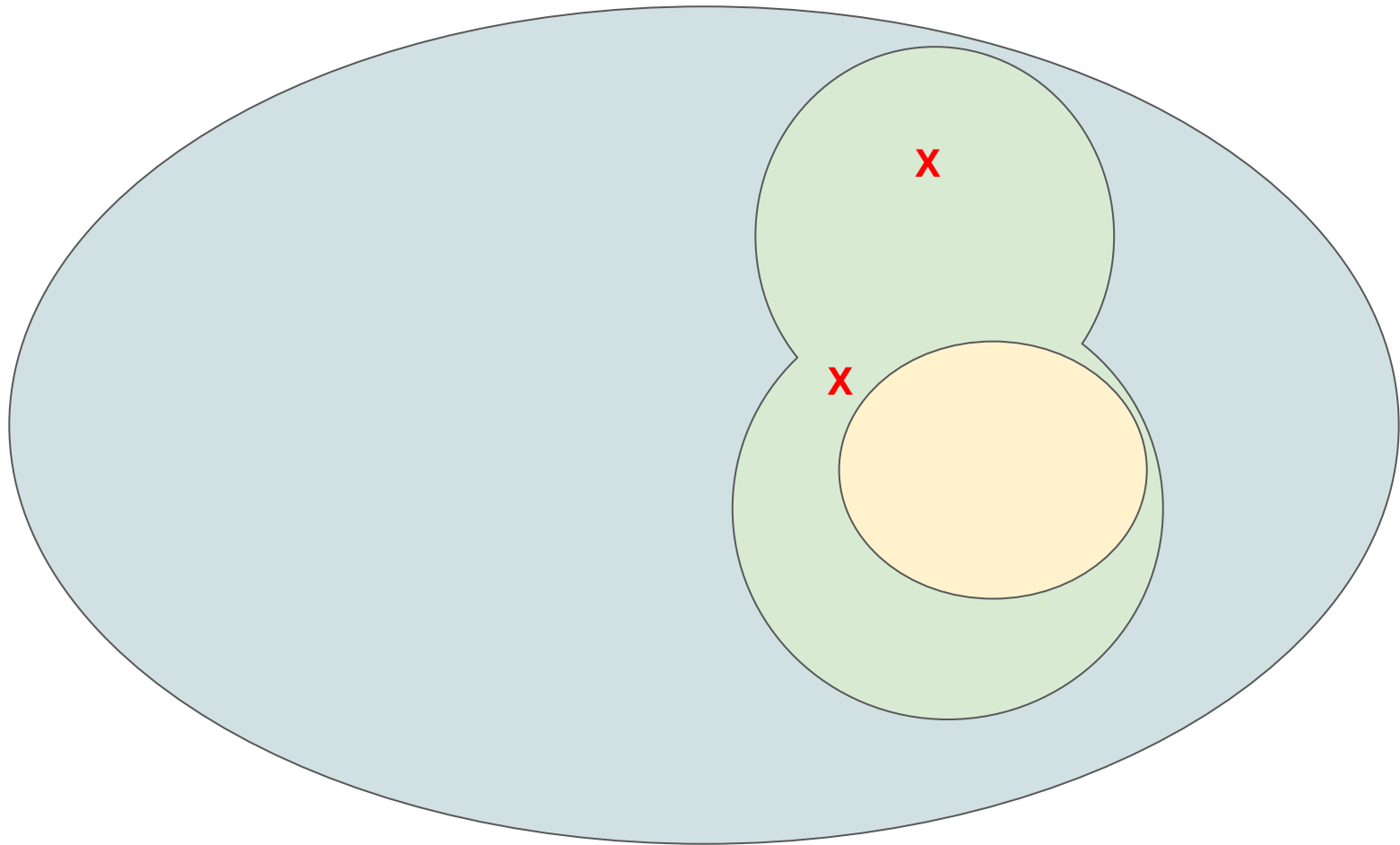
Call method "foobar" with arg 42.

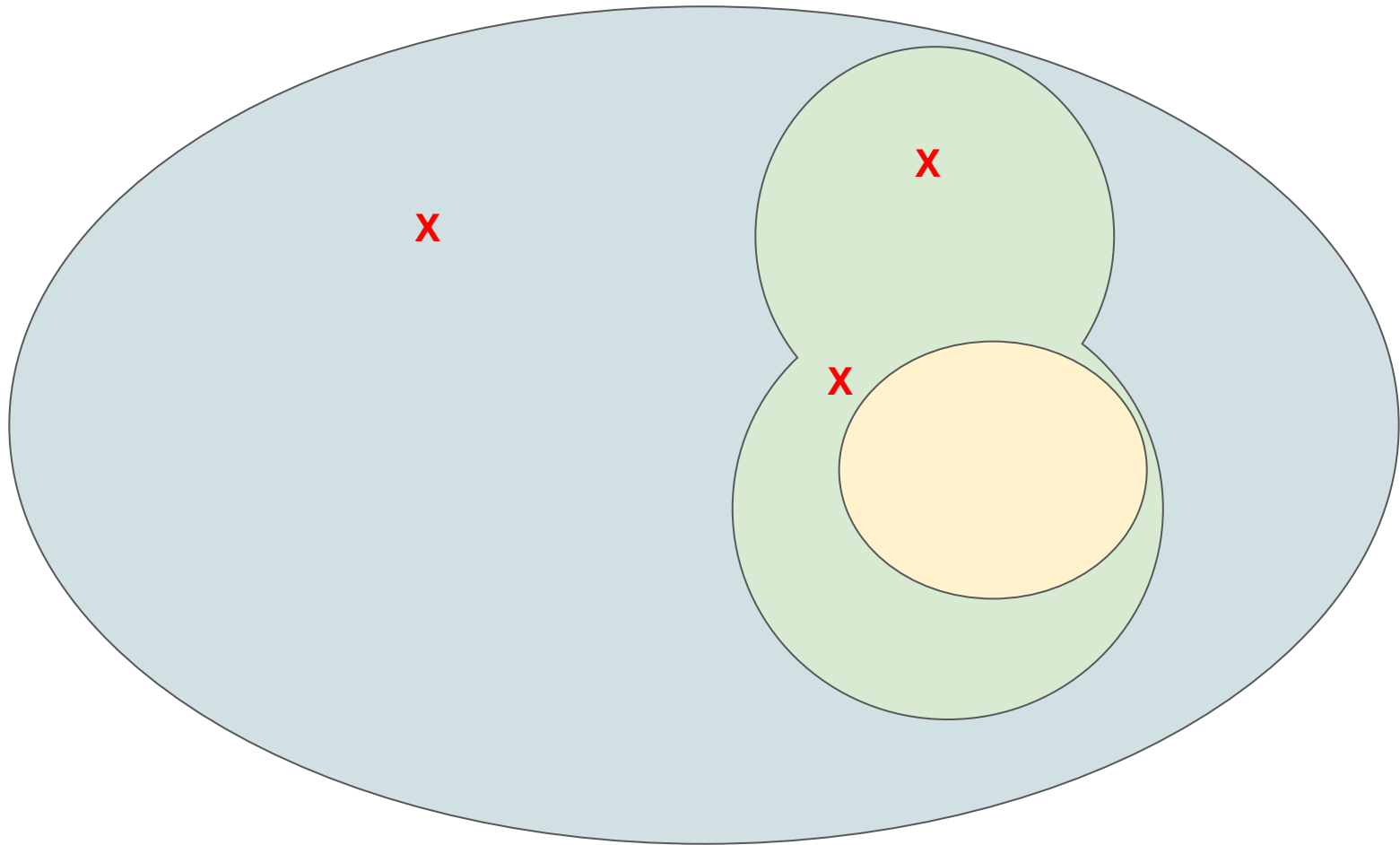
Exploration (Step 2)

```
function f2(v3, v4) {  
    v3.foobar(42);  
}
```

Example bug: crbug.com/1377775

```
const v19 = {};  
v19.a = 42;  
const v20 = [v19];  
function v21(v23) {  
    const v26 = v23.shift();  
    const v27 = v23.at(10000000);    // .at is inlined by Turbofan but the type check  
}                                     // is faulty, leading to a type confusion when  
v19.__proto__ = v20;                 // v23 is not a JSArray.  
for (let v39 = 0; v39 < 100; v39++) {  
    const v43 = v21(v19);  
    const v45 = v21(v20);  
}
```





Example bug: crbug.com/1377775

```
const v19 = {}; // Step 1
v19.a = 42; // Step 2
const v20 = [v19]; // Step 3
function v21(v23) {
    const v26 = v23.shift(); // Step 4
    const v27 = v23.at(1000000); // Step 5
}
v19.__proto__ = v20; // Step 6
for (let v39 = 0; v39 < 10000; v39++) { // Step 7
    const v43 = v21(v19); // Step 8
    const v45 = v21(v20); // Step 9
}
```

CVE-2022-3723 (V8 ITW)

```
function setInnerProperty(o) {
  o.inner.foo = {};
}
function makeObject() {
  var o = {
    inner: {
      ['foo']: 0
    }
  };
  setInnerProperty(o, ...arguments);
  return o;
}
makeObject();
gc();
makeObject();
gc();

let o = makeObject();
%HeapObjectVerify(o.inner);
```

<https://t.me/learningnets>

CVE-2022-3723 (V8 ITW)

```
function setInnerProperty(o) { // Step 1
  o.inner.foo = {}; // Step 2-3
}
function makeObject() { // Step 4
  var o = { // Step 5
    inner: { // Step 6
      ['foo']: 0 // Step 7
    }
  };
  setInnerProperty(o, ...arguments); // Step 8
  return o;
}
makeObject(); // Step 9
gc(); // Step 10
makeObject(); // Step 11
gc(); // Step 12

let o = makeObject(); // Step 13
%HeapObjectVerify(o.inner); // Step 14
```

<https://t.me/learningnets>

CVE-2022-3723 (V8 ITW)

```
function setInnerProperty(o) { // Step 1
  o.inner.foo = {}; // Step 2-3
}
function makeObject() { // Step 4
  var o = { // Step 5
    inner: { // Step 6
      ['foo']: 0 // Step 7
    }
  };
  setInnerProperty(o, ...arguments); // Step 8
  return o;
}
makeObject(); // Step 9
gc(); // Step 10
makeObject(); // Step 11
gc(); // Step 12

let o = makeObject(); // Step 13
%HeapObjectVerify(o.inner); // Step 14
```

<https://t.me/learningnets>

CVE-2022-3723 (V8 ITW)

```
function setProperty(o) { // Step 1
  o.foo = {}; // Step 2
}
function makeObject() { // Step 3
  var o = { // Step 4
    ['foo']: 0 // Step 5
  };
  setProperty(o, ...arguments); // Step 6
  return o;
}
makeObject(); // Step 7
%GetObjectIntoInterestingState(o); // Step 8
makeObject(); // Step 9
%HeapObjectVerify(o.inner); // Step 10
```

How to find this one?



- Import existing JavaScript code for mutation, hope it's "close" to the bug
 - Now possible with new JavaScript -> FuzzIL compiler!
- Use different feedback
 - Future research topic?
- Use specialized mutators
 - To "hint" fuzzer towards known bug patterns
- **Use "human-assisted" fuzzing**
 - Let the researcher guide the fuzzer

Code Shape ITW

```
function f1(...) {  
    ...  
}
```

```
function f2(...) {  
    ...  
}
```

...

```
%GetObjectIntoInterestingState(...);
```

...

```
for(let i = 0; i < 100; i++) {  
    ...  
}
```

<https://t.me/learningnets>

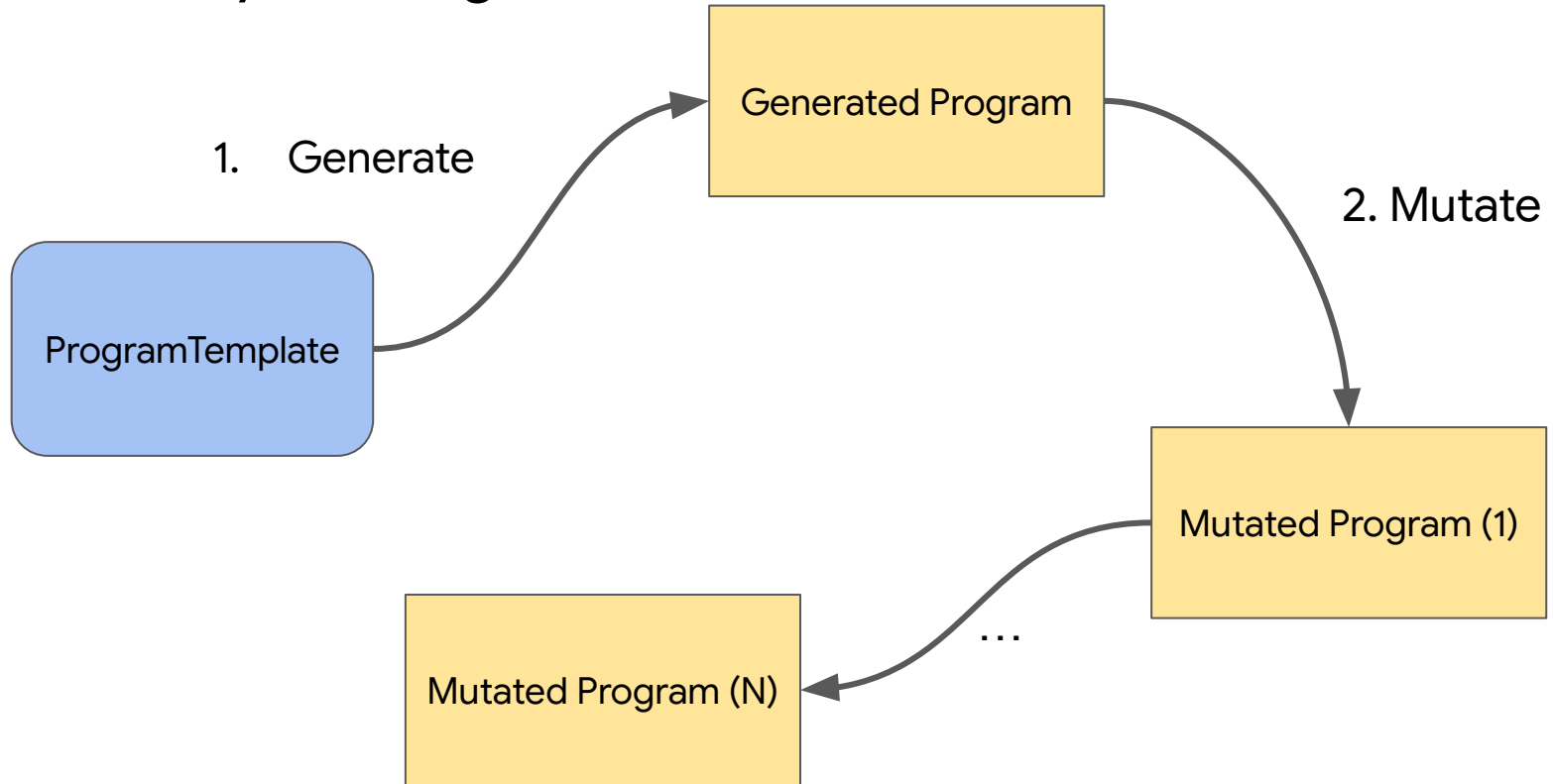
“Hybrid” Fuzzing with ProgramTemplates

```
let f1 = b.buildPlainFunction(with: b.randomParameters()) {  
    b.build(50)  
}
```

```
b.getObjectIntoInterestingState(b.randomArguments())
```

```
b.buildRepeatLoop(n: 100) {  
    let f = b.randomFunction()  
    let args = b.randomArguments(forCalling: f)  
    b.callFunction(, withArgs: args)  
}
```

Fuzzilli's HybridEngine



RegExp Fuzzer in Fuzzilli

```
ProgramTemplate("RegExpFuzzerTemplate") { b in  
    let f = b.buildPlainFunction(with: .parameters(n: 0)) {  
        let pattern = probability(0.5) ? chooseUniform(from: b.fuzzer.environment.interestingRegExps) : b.randomString()  
        let regExpVar = b.loadRegExp(pattern, RegExpFlags.random())  
  
        let subjectVar: b.loadString(b.randomString())  
  
        let symbol = b.loadBuiltin("Symbol")  
        let resultVar = b.callMethod("exec", on: regExpVar, withArgs: [subjectVar])  
  
        b.build(n: 7)  
    }  
    b.doReturn(resultVar)  
}  
  
b.callFunction(f)  
b.callFunction(f)  
  
b.build(n: 15)  
}
```

RegExp Fuzzer in Fuzzilli

crbug.com/1439691:

```
function f0() {  
}  
/(?!(a))\1/gudyi[Symbol.replace]("f\uD83D\uDCA9ba\u2603", f0);
```

Serializer API Fuzzer

```
// Serialize a random object
let content = b.callMethod("serialize", on: serializer, withArgs: [b.randomVariable()])

// Mutate the contents
b.mutate(content)

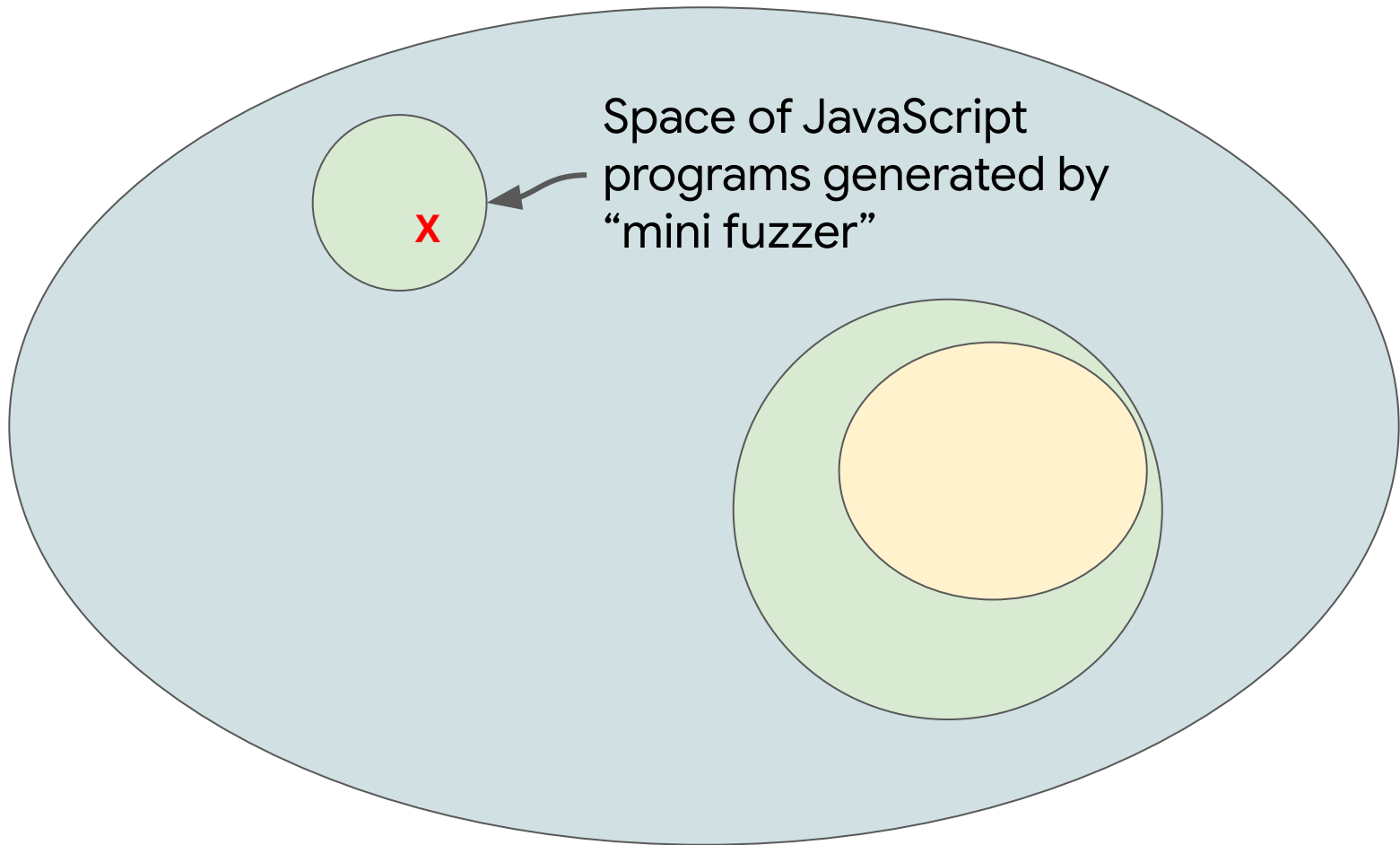
// Deserialize the resulting buffer
let _ = b.callMethod("deserialize", on: serializer, withArgs: [content])

// Deserialized object is available in a variable now and can be used by following code
```

Serializer API Fuzzer

crbug.com/1364974:

```
const v4 = d8.serializer.serialize(-2147483648n);  
const v5 = new Uint8Array(v4);  
v5[3] = 1;  
const v9 = d8.serializer.deserialize(v4);  
~v9;
```



Summary

- Key challenge: JavaScript “search space” is extremely large
- Coverage guided fuzzing only gets you so far
 - Can find some types of bugs, but will struggle with others
- Specific mutators can be used to target certain bug types
 - Fuzzilli’s new Probe- and Explore mutators have each found new bugs
- “Human-guided fuzzing” to target areas that researcher deems interesting
 - Can build “mini fuzzers” on top of Fuzzilli’s HybridEngine