Token-Level Fuzzing

Chris Salls, Chani Jindal, Jake Corina
Christopher Kruegel, Giovanni Vigna

SECLAB

UCSB

Motivation - Bugs in JS Engines

CVE-2017-8729

```
function f() {
  ({
     a: {
        b = 0x1111
        c = 0x2222,
     }.c = 0x3333
   } = {});
}

f();
```

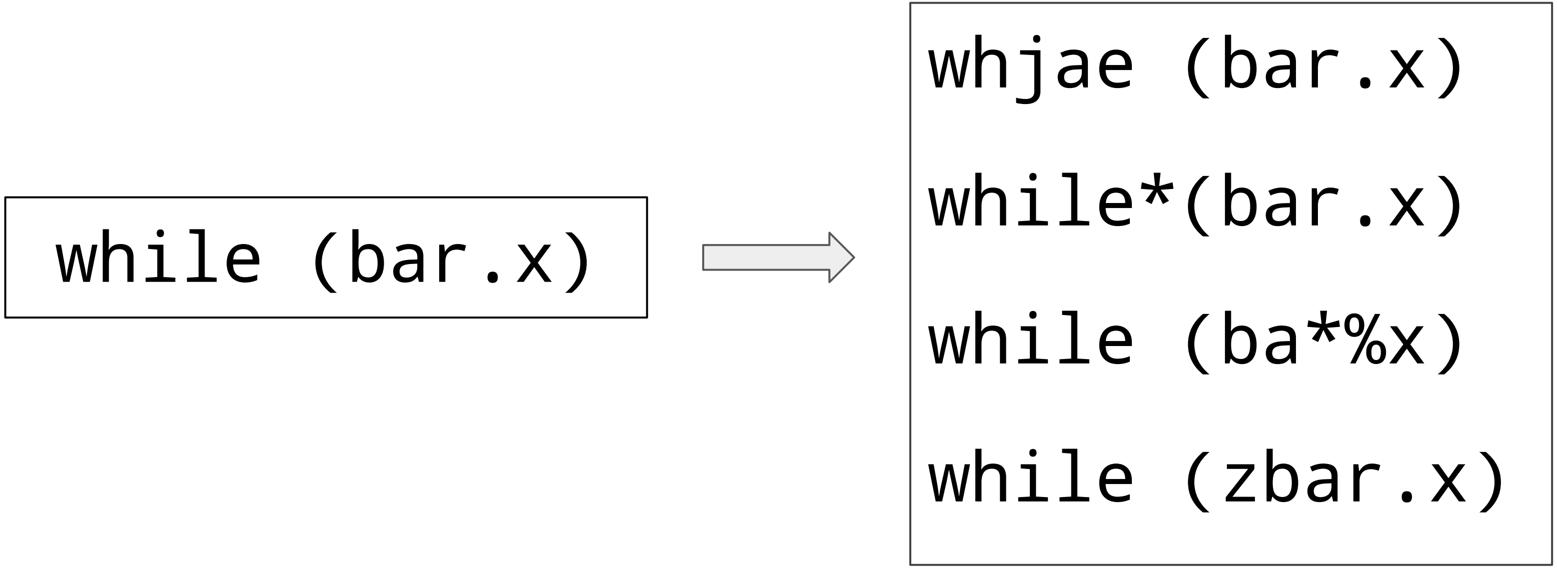
CVE-2018-17463

```
function hax(o) {
 o.a;
 Object.create(o);
 return o.b;
}
for (i = 0;i<100000;i++) {
 let o = {a: 42};
 o.b = 43;
 hax(o);
}
```

“Byte-Level Fuzzing”

- Traditional Fuzzers apply mutations on bytes
- What AFL/LibFuzzer and many other fuzzers do

“Byte-Level Fuzzing”



```
while (bar.x)
```

```
whjae (bar.x)

while*(bar.x)

while (ba*%x)

while (zbar.x)
```

“Byte-Level Fuzzing”

- Most mutations result in simple syntax errors
- Fails to generate much coverage
- “Dictionaries” can improve results, but still suffer from the same issues

Grammar-Based Fuzzing

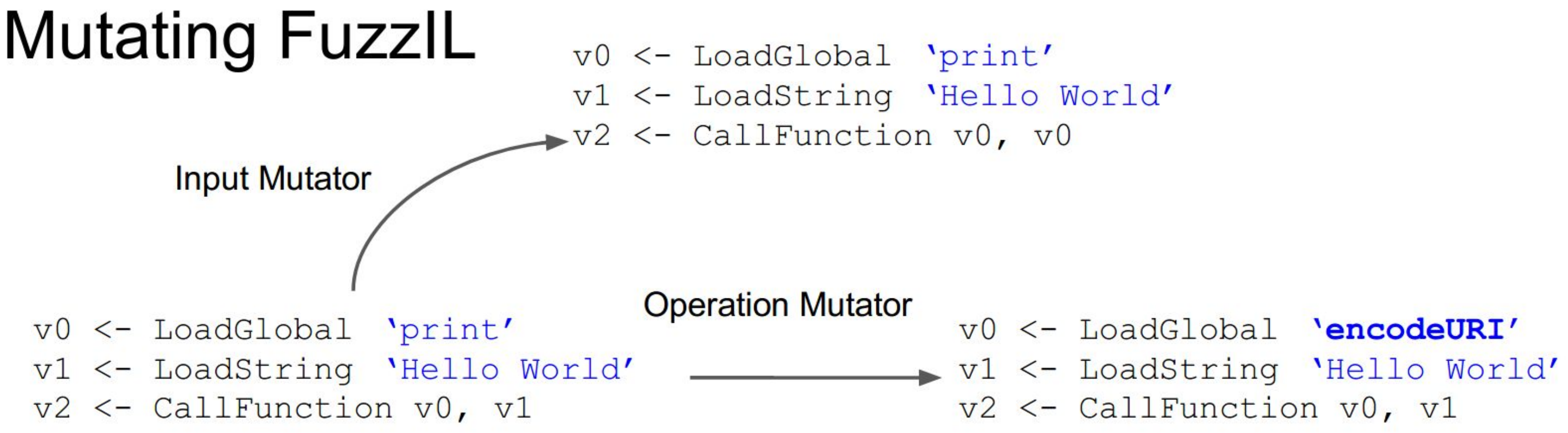
- Most common way to fuzz interpreters
- Specify a “grammar” and apply mutations following the grammar

Grammar-Based Fuzzing - Example

```
; Example FuzzIL program
v0 <- LoadInt ‘0’
v1 <- LoadInt ‘10’
v2 <- LoadInt ‘1’
v3 <- Phi v0
BeginFor v0, ‘<’, v1, ‘+’, v2 -> v4
  v6 <- BinaryOperation v3, ‘+’, v4
  Copy v3, v6
EndFor
v7 <- LoadString ‘Result: ‘
```

* Taken from https://saelo.github.io/presentations/offensivecon_19_fuzzilli.pdf

Grammar-Based Fuzzing - Example



Mutating FuzzIL

```
v0 <- LoadGlobal  'print'
v1 <- LoadString  'Hello World'
v2 <- CallFunction v0, v1
```

Input Mutator

```
v0 <- LoadGlobal  'print'
v1 <- LoadString  'Hello World'
v2 <- CallFunction v0, v0
```

Operation Mutator

```
v0 <- LoadGlobal  'encodeURI'
v1 <- LoadString  'Hello World'
v2 <- CallFunction v0, v1
```

* Taken from https://saelo.github.io/presentations/offensivecon_19_fuzzilli.pdf

Limitations Grammar-Based Fuzzing

- Limiting to a grammar limits the bugs you can find
- E.g. FuzzIL never assigns to a variable more than once
    - -> Can never find bugs that require assigning to a variable multiple times
- Does not generate inputs with invalid syntax
- And bugs like these exist!

Limitations Grammar-Based (old bugs)

Chromium Issue 800032 - Semantic error leads to OOB Write

```
class Sub extends RegExp {
  constructor(a) {
    const a = 1; // semantic error
  }
}

let o = Reflect.construct(RegExp,[],Sub);
// OOB write
o.lastIndex = 0x1234;
```

Limitations Grammar-Based (old bugs)

CVE-2017-8729 - Syntax error leads to type confusion

```
function f() {
  ({
     a: {
        b = 0x1111, // invalid assignment
        c = 0x2222,
     }.c = 0x3333
   } = {});
}

f();
```

So what are we missing?

Grammar Based Fuzzing

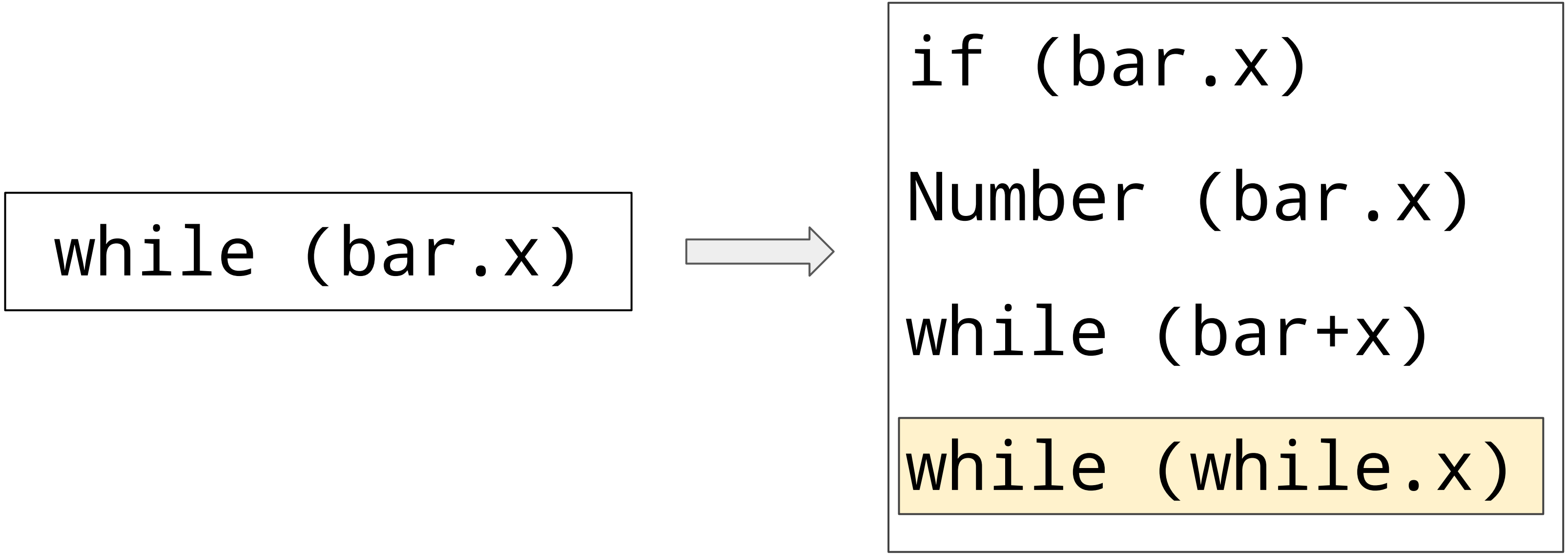
Token Level Fuzzing

Byte Level Fuzzing

“Token-Level Fuzzing”

- Mutations applied on individual tokens
- Allows fuzzer to make more useful mutations

“Token-Level Fuzzing”



```
while (bar.x)
```

```
if (bar.x)

Number (bar.x)

while (bar+x)

while (while.x)
```

Implementation

Input seeds

```
function foo() {
    try {
    } catch (x) {
        var x = 18;
    }
    print(x);
}
```

Step 1) Rewrite

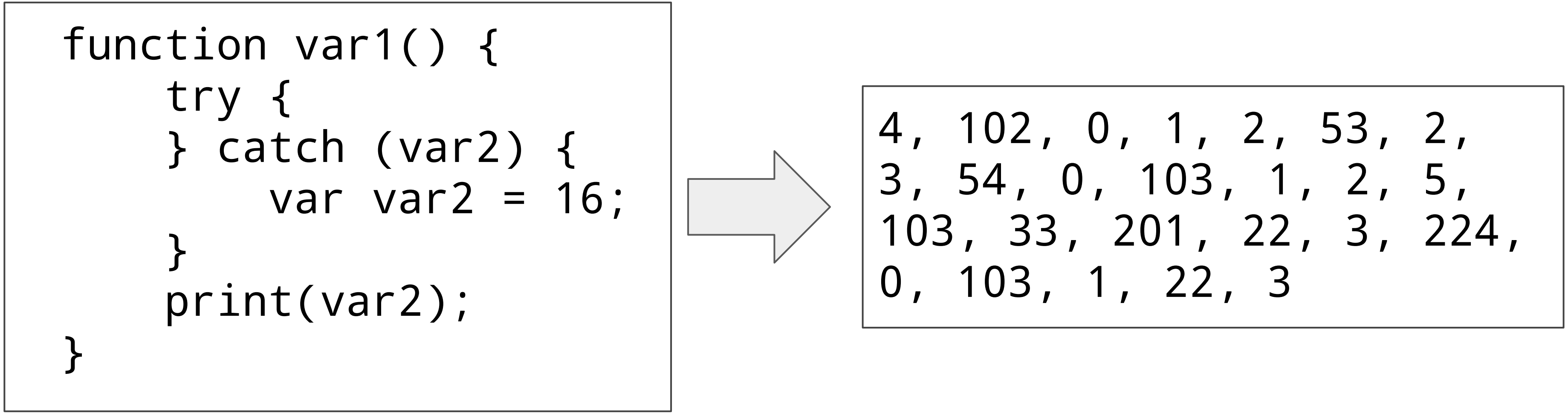
```
function var1() {
    try {
    } catch (var2) {
        var var2 = 16;
    }
    print(var2);
}
```

Step 2) Identify tokens assign unique numbers

```
function, var1, (, ), {,
try, }, catch, var2, var,
...
```

```
0: (
1: )
2: {
3: }
4: function
5: var
6: Math
.
.
.
```

Step 3) Encode



```
function var1() {
    try {
    } catch (var2) {
        var var2 = 16;
    }
    print(var2);
}
```

```
4, 102, 0, 1, 2, 53, 2,
3, 54, 0, 103, 1, 2, 5,
103, 33, 201, 22, 3, 224,
0, 103, 1, 22, 3
```

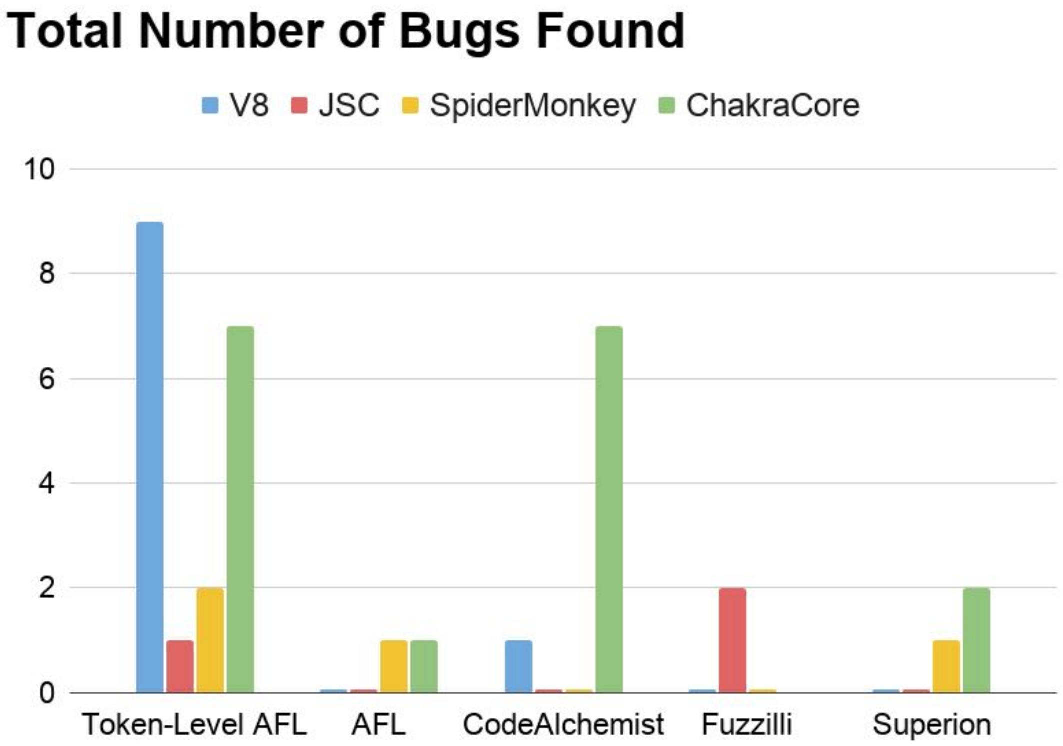
Inputs become a list of numbers

- Mutations can be applied directly on the numbers
- Before executing an input, decode the list back into Javascript Tokens!
- Only requires small modifications to AFL

Experiments

- Compared against other state-of-the-art Fuzzers
    - (3 days X 30 cores)
    - AFL, CodeAlchemist, Fuzzilli
- Ran Token-Level AFL on latest JavaScript Engines
    - (60 days X 30 cores)
    - V8, JavaScriptCore, SpiderMonkey, ChakraCore

Results - 3 day X 30 core runs



Results - 60 Days X 30 Cores

Token-Level Fuzzing Found:

- 16 V8 Bugs
- 4 JSC Bugs
- 3 SpiderMonkey Bugs
- 6 ChakraCore Bugs

Case Study 1

```
class var6 extends Object {
   constructor ( a,b,c) {
      super (1.1 ) 1 ;
   }
};

new var6();
```

Case Study 2

```
function f () {
   var14=[1,2,3,4,5,6,7,8];
   var15=var14;
   var14.length = 0x100 ;
   var14.__defineGetter__(/./, function(){
      var14.unshift ( 0x20 ) ;
      var14.shift();
      var var3=new Uint32Array(var14);
      Object.entries(var14).toString();
   } ) ;
   print(Object.entries(var14).toString());
}
f();
```

Conclusion

- Token-Level Fuzzing is a promising new technique
- Can make use of existing mutation fuzzers such as AFL
- Finds different bugs than other state of the art fuzzers