

FuzzJIT: Oracle-enhanced Fuzzing for JavaScript Engine JIT Compiler

Junjie Wang⁺, Zhiyi Zhang^{*}, Shuang Liu⁺, Xiaoning Du[^], Junjie Chen⁺

Tianjin University⁺

Qi An Xin Group Corp.^{*}

Monash University[^]

Browser is vital in our daily life



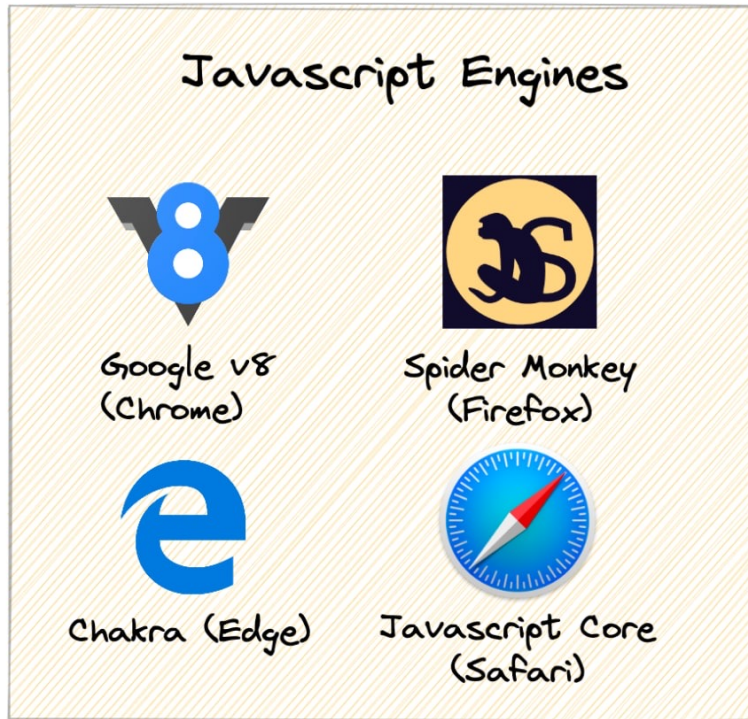
- Web browsing
- Social media
- Online shopping
- Online banking
- Online collaboration
- ...

Browser can get compromised



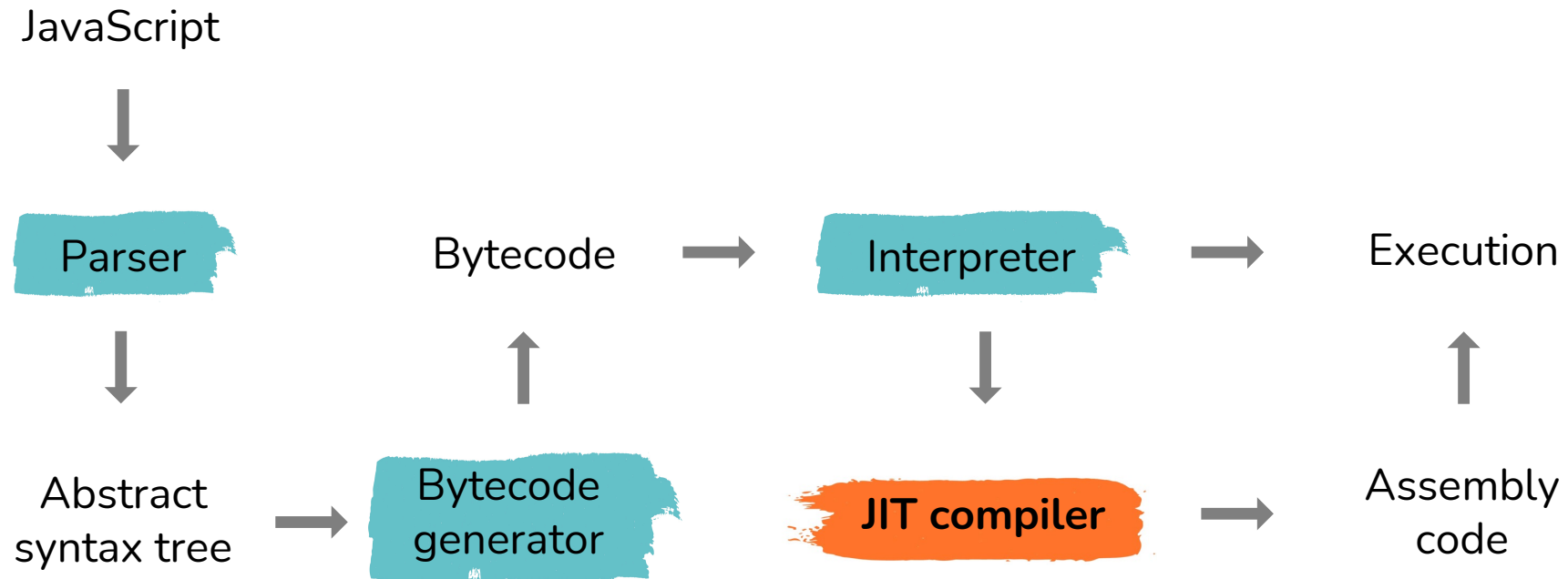
- At Pwn2Own 2022, Manfred Paul successfully demonstrated 2 bugs on Mozilla Firefox, earning him \$100,000.
- Manfred Paul successfully scored his second win on Apple Safari, earning him \$50,000.

JavaScript engine powers browser



- Parse and validate JavaScript
- Execute JavaScript
- JIT compile and optimize JavaScript

Architecture of JavaScript engine



JIT compiler do lots of optimization

```
var c = a + b;
```

```
var d = a + b;
```



```
var c = a + b;
```

```
var d = c;
```

- Bound check elimination
- Constant folding
- Dead code elimination
- Common subexpression elimination
- Redundancy elimination
- ...

JIT compiler is error-prone

```
var a = 1234;  
a = "zhunki";  
a = {};
```

- JavaScript is a weakly and dynamically typed language.

JIT compiler is error-prone

```
var a = 1234;  
a = "zhunki";  
a = {};
```

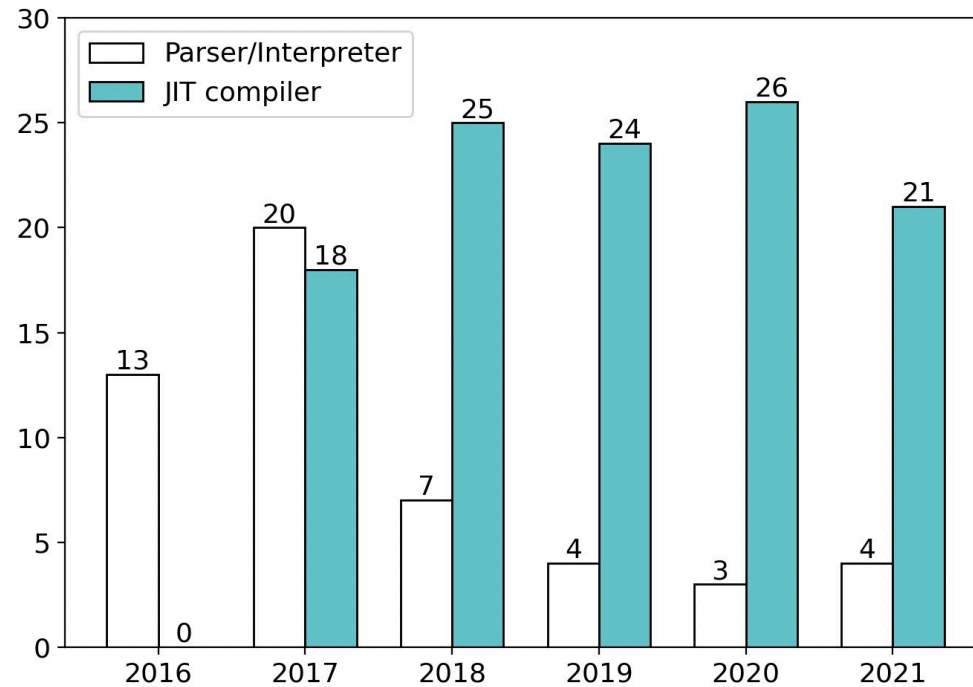
- JavaScript is a weakly and dynamically typed language.
- A direct optimization is not realistic due to the potential ambiguity of variable types.

JIT compiler is error-prone

```
var a = 1234;  
a = "zhunki";  
a = {};
```

- JavaScript is a weakly and dynamically typed language.
- A direct optimization is not realistic due to the potential ambiguity of variable types.
- JIT compiler profiles variable types with runtime information to make optimization decisions.

JIT compiler is error-prone



- The number of JIT compiler bugs is around four times that of the parser/interpreter bugs during the past four years.

JIT compiler is error-prone

CVE-2021-21220 (JIT)

CVE-2020-9805 (JIT)

CVE-2019-9813 (JIT)

CVE-2019-6217 (JIT)

CVE-2019-6216 (JIT)

- Among 8 successful Pwn2Own demonstrations in 2019 to 2021, 6 of them exploit 5 JIT compiler bugs.

How to detect JIT compiler bugs?



- Current existing JavaScript engine fuzzors:
 - Mainly using **crash** as the oracle

How to detect JIT compiler bugs?



- Current existing JavaScript engine fuzzors:
 - Mainly using **crash** as the oracle
 - Is it enough?

880207: Math.expm1 typing bug

```
function foo(){  
  return Object.is(Math.expm1(-0),-0);  
}
```

- $\text{Math.expm1}(x) = e^x - 1$

880207: Math.expm1 typing bug

```
function foo(){  
  return Object.is(Math.expm1(-0),-0);  
}
```

- $\text{Math.expm1}(x) = e^x - 1$
- `Object.is` determines whether two values are the same value.

880207: Math.expm1 typing bug

```
function foo(){  
  return Object.is(Math.expm1(-0),-0);  
}  
  
console.log(foo()); // true
```

- $\text{Math.expm1}(x) = e^x - 1$
- `Object.is` determines whether two values are the same value.

880207: Math.expm1 typing bug

```
function foo(){  
  return Object.is(Math.expm1(-0),-0);  
}
```

```
console.log(foo()); // true  
%OptimizeFunctionOnNextCall(foo);
```

- $\text{Math.expm1}(x) = e^x - 1$
- `Object.is` determines whether two values are the same value.

880207: Math.expm1 typing bug

```
function foo(){  
  return Object.is(Math.expm1(-0),-0);  
}
```

```
console.log(foo()); // true  
%OptimizeFunctionOnNextCall(foo);  
console.log(foo()); // false
```

- $\text{Math.expm1}(x) = e^x - 1$
- `Object.is` determines whether two values are the same value.

880207: Math.expm1 typing bug

```
function foo(){  
  return Object.is(Math.expm1(-0),-0);  
}  
  
console.log(foo()); // true  
%OptimizeFunctionOnNextCall(foo);  
console.log(foo()); // false
```

- $\text{Math.expm1}(x) = e^x - 1$
- `Object.is` determines whether two values are the same value.
- What harm can the subtle difference between `-0` and `0` cause?

880207: Math.expm1 typing bug

```
function foo(){  
  return Object.is(Math.expm1(-0),-0);  
}
```

```
console.log(foo()); // true  
%OptimizeFunctionOnNextCall(foo);  
console.log(foo()); // false
```

- $\text{Math.expm1}(x) = e^x - 1$
- `Object.is` determines whether two values are the same value.
- What harm can the subtle difference between `-0` and `0` cause?
- [Exploiting the Math.expm1 typing bug in V8](#)

Remark



- There are many other JIT compiler bugs:
 - only cause subtle difference before/after optimization rather than crash
 - but could be exploitable.

Insight



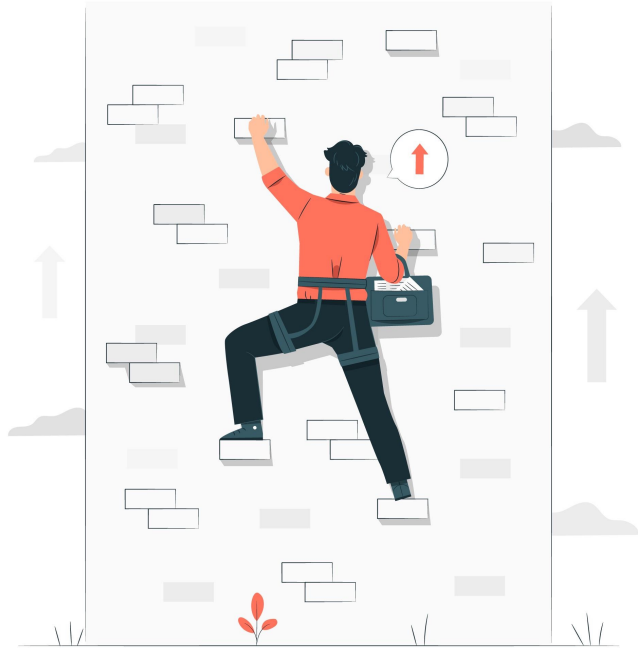
- JIT compiler shall only speed up but never change the output.

How to detect JIT compiler bugs?



- Current existing JavaScript engine fuzzors:
 - Mainly using **crash** as the oracle
 - Is it enough?
- We need an **enhanced oracle** to detect both crash and non-crash JIT compiler bugs.

Our approach




1. Activating JIT compiler for each test case.
2. Precisely capturing discrepancy caused by JIT compiler.
3. Mutation strategy to reveal JIT compiler bugs.

1. Activating JIT compiler

- JIT compiler can be activated when certain JavaScript code becomes hot, i.e., being executed enough times.

1. Activating JIT compiler

- 
- JIT compiler can be activated when certain JavaScript code becomes hot, i.e., being executed enough times.
 - We wrap the testing content into a function (opt) and invoke it inside for loops.

1. Activating JIT compiler

```
function opt(){  
  ...  
}  
  
for(var i=0; i<1000; i++)  
  opt();
```

- JIT compiler can be activated when certain JavaScript code becomes hot, i.e., being executed enough times.
- We wrap the testing content into a function (opt) and invoke it inside for loops.

1. Activating JIT compiler

```
function opt(){  
  ...  
}  
  
for(var i=0; i<1000; i++)  
  opt();
```

- JIT compiler can be activated when certain JavaScript code becomes hot, i.e., being executed enough times.
- We wrap the testing content into a function (opt) and invoke it inside for loops.
- The number and times of for loops are determined by optimization conditions of each JavaScript engine.

2. Capturing discrepancy

```
function opt(){
```

```
  ...
```

```
}
```

```
for(var i=0; i<1000; i++)
```

```
  opt();
```

- Compare if the return value of optimized function before JIT and after JIT **deeply** equals.

2. Capturing discrepancy

```
function opt(){  
  ...  
}  
  
var beforeJIT = opt();  
for(var i=0; i<1000; i++)  
  opt();  
  
var afterJIT = opt();  
if(!deepEquals(beforeJIT, afterJIT))  
  crash();
```

- Compare if the return value of optimized function before JIT and after JIT **deeply** equals.

2. Capturing discrepancy

```
function opt(){  
  ...  
}  
var beforeJIT = opt();  
for(var i=0; i<1000; i++)  
  opt();  
var afterJIT = opt();  
if(!deepEquals(beforeJIT, afterJIT))  
  crash();
```

- Compare if the return value of optimized function before JIT and after JIT **deeply** equals.
- To eliminate false alarms, we forbid the generation of some APIs:
 - Math.random()
 - Date.now()
 - ...

3. Mutation strategies

```
function opt(){  
  ...  
}  
var beforeJIT = opt();  
for(var i=0; i<1000; i++)  
  opt();  
var afterJIT = opt();  
if(!deepEquals(beforeJIT, afterJIT))  
  crash();
```

- Increasing the probability of generating JIT bug related elements:
 - Arrays
 - Objects
 - Interesting numbers
 - ...

3. Mutation strategies

```
function opt(){  
    arrays, objects, interesting numbers...  
}  
var beforeJIT = opt();  
for(var i=0; i<1000; i++)  
    opt();  
var afterJIT = opt();  
if(!deepEquals(beforeJIT, afterJIT))  
    crash();
```

- Increasing the probability of generating JIT bug related elements:
 - Arrays
 - Objects
 - Interesting numbers
 - ...

FuzzJIT implementation

```
function opt(){  
  arrays, objects, interesting numbers...  
}  
var beforeJIT = opt();  
for(var i=0; i<1000; i++)  
  opt();  
var afterJIT = opt();  
if(!deepEquals(beforeJIT, afterJIT))  
  crash();
```

- One template + Fuzzilli

FuzzJIT implementation

```
function opt(){  
  arrays, objects, interesting numbers...  
}  
var beforeJIT = opt();  
for(var i=0; i<1000; i++)  
  opt();  
var afterJIT = opt();  
if(!deepEquals(beforeJIT, afterJIT))  
  crash();
```

- One template + Fuzzilli
- Fuzzilli is a coverage-guided fuzzer for JavaScript engines based on a custom intermediate language (FuzzIL).
- Fuzzilli provides:
 - Coverage guidance
 - Fuzzing queue organization
 - Test case execution
 - Fault detection
 - ...

1-month evaluation: found new bugs

- JavaScriptCore (10)

233353: undefined/NaN

239757: undefined/NaN

239758: -Infinity/Infinity

228068: True/False

232866: -NaN/NaN

233118: crash

232869: 1/-1

-: -Infinity/Infinity

-: 255/0

-: crash

- V8 (5)

1224283: undefined/123

12471: 14951/14955

11977: True/False

1276923: crash

12495: opt()/11

1-month evaluation: found new bugs

- SpiderMonkey (2)

- 1747013: opt()/NaN

- 1747777: crash

- ChakraCore (16)

- 6783: True/False

- 059706: crash

- 6762: crash

- 6763: crash

- 6764: crash

- 6765: crash

- 6766: crash

- ...

1-month evaluation: coverage

- FuzzJIT outperforms state-of-the-art fuzzers
 - Superior: +30.04%
 - DIE: +3.48%
 - Fuzzilli: +16.47%

Thank you!

Q & A

Contact us: junjie.wang@tju.edu.cn