OptFuzz: Optimization Path Guided Fuzzing for JavaScript JIT Compilers

Jiming Wang[1], Yan Kang[1], Chenggang Wu[1], Yuhao Hu[1], Yue Sun[1], Jikai Ren[1], Yuanming Lai[1], Mengyao Xie[1], Charles Zhang[2], Tao Li[3], Zhe Wang[1]

[1]Institute of Computing Technology, Chinese Academy of Sciences
[2]Tsinghua University
[3]Nankai University

JavaScript Engine

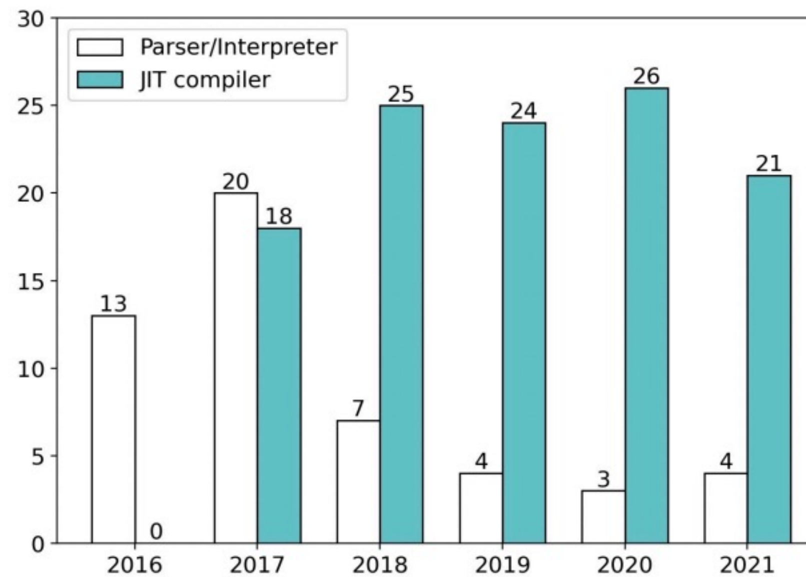
- JavaScript engines are widely used in various applications
  - Web browsers
  - PDF readers
  - React Native applications
  - …

JIT Compiler

- JIT compiler is complex and incorporates various optimizations
  - Common Subexpression Elimination (CSE)
  - Loop Invariant Code Motion (LICM)
  - Constant Folding
  - Dead Code Elimination (DCE)
  - Strength Reduction
  - Bound Check Elimination (BCE)
  - CFG Simplification
  - …

JIT Compiler

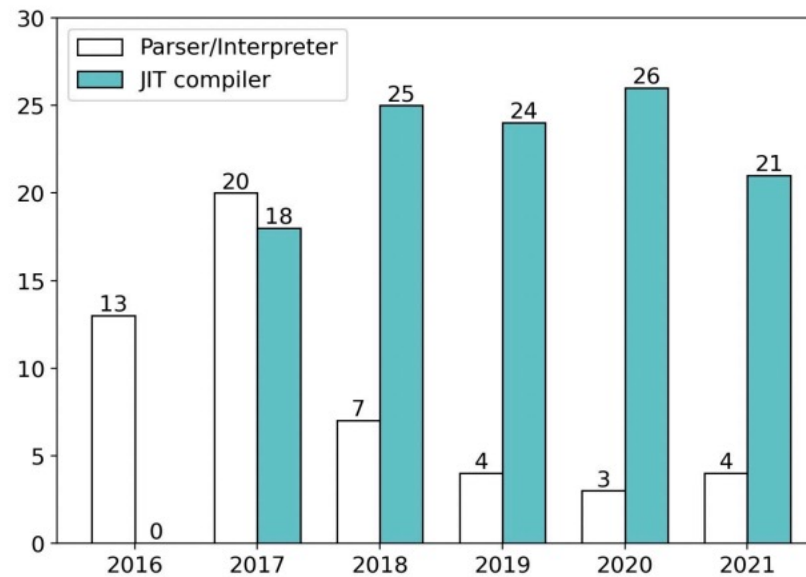
- JIT compiler is complex and incorporates various optimizations
- Potential hotspot for vulnerabilities within JavaScript engines
  - CVE-2019-5857
  - CVE-2019-8518
  - CVE-2020-9802
  - CVE-2021-30599
  - CVE-2021-21230
  - CVE-2022-46691
  - CVE-2023-32439
  - …



From FuzzJIT (Usenix Security 2023)

JIT Compiler

- JIT compiler is complex and incorporates various optimizations
- Potential hotspot for vulnerabilities within JavaScript engines
  - CVE-2019-5857
  - CVE-2019-8518
  - CVE-2020-9802
  - CVE-2021-30599
  - CVE-2021-21230
  - CVE-2022-46691
  - CVE-2023-32439
  - …



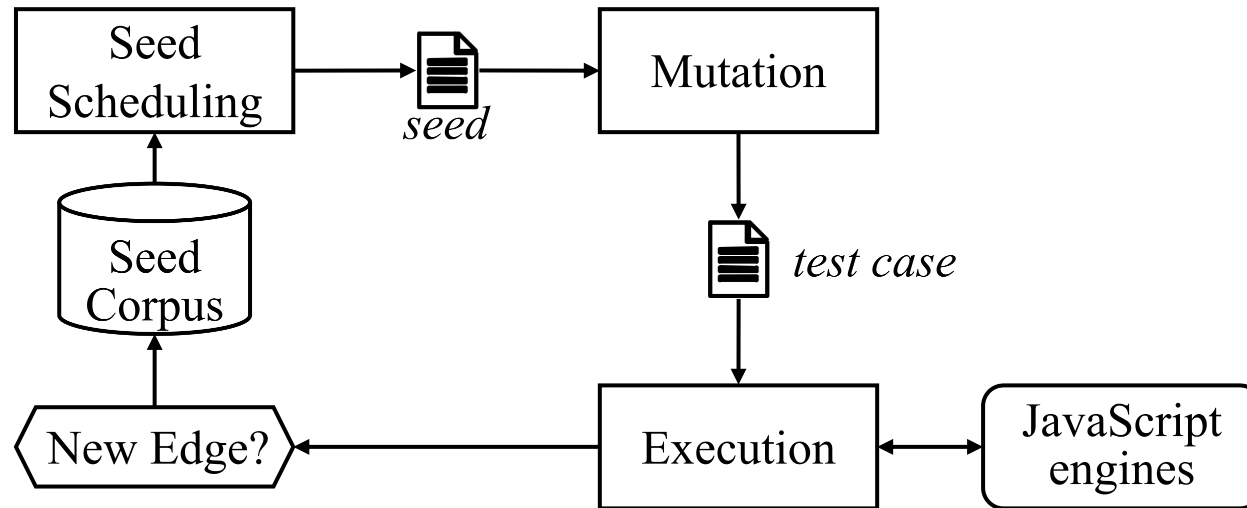
From FuzzJIT (Usenix Security 2023)

*The optimization is **triggered** and there is an **error** in the optimization.*

中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

How to find JIT optimization bugs?

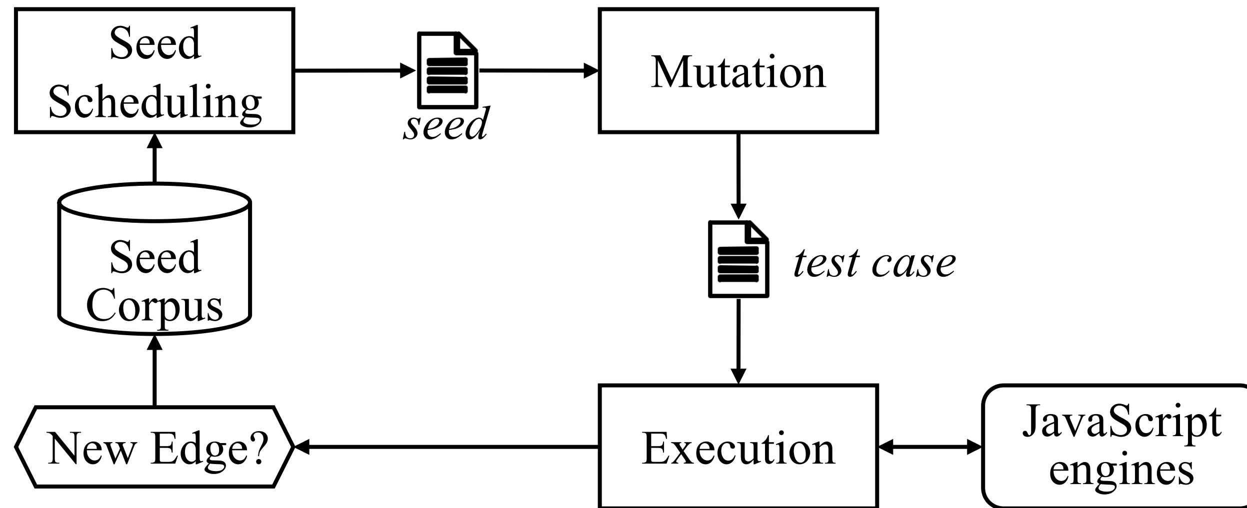
- Edge coverage-guided fuzzing ?



中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

How to find JIT optimization bugs?

- Edge coverage-guided fuzzing ?



Merely exploring new code is insufficient for finding bugs in the JIT optimization passes. ***Why?***

Analysis on JIT Optimization

- Optimization Path
- Three Observations:
  - Enter Optimization ≠ Tigger Optimization
  - Overlooked Optimization Path
  - Imbalanced Testing for Optimization Path

Optimization Path

```
1  void convertToJump( BasicBlock* BB, BasicBlock* targetBB ) {
2      if ( CanMergeBlocks( BB, targetBB ) )
3          mergeBlocks( BB, targetBB ); //optimization
4      else
5          BB->replaceTerminal( Jump ); //optimization
6  }
7  void run( ) {
8      for ( BasicBlock* BB : graph ) {
9          if ( !BB )
10             continue;
11         switch ( BB>terminal( ) ) {
12             case Branch:
13                 …
14                 if ( BB->successor( 0 ) == BB->successor( 1 ) ) {
15                     convertToJump( BB, BB->successor( 0 ) );
16                     break;
17                 }
18             case Switch:
19                 for ( int i = 0; i < cases.size( ); i++ ) {
20                     // Analyze the target of cases
21                 }
22                 if ( cases.isEmpty( ) ) {
23                     convertToJump( BB, BB->fallThrough( ) );
24                     break;
25                 }
26         }
27     }
28 }
```

- A large outer loop
- Traverse the intermediate representation (IR) of the optimized JavaScript code.
- The optimization is triggered if a specific condition is met. ➡ *Optimization path*
- The functionality of the optimization pass is manifested in these optimization paths.

Optimization Path: 9→11→14→15→2→3

Enter Optimization ≠ Tigger Optimization

- During JIT compilation, each optimization pass is executed.
- Multiple conditions must be met for triggering the optimization.
- Fuzzing on JavaScriptCore

Fuzzer	Total Testcase	Enter LICM	Tigger LICM
DIE	1370090	721074	263908 (36.6%)
Fuzzilli	17406775	8266477	3660644 (44.3%)

```
1  void convertToJump( BasicBlock* BB, BasicBlock* targetBB ) {
2      if ( CanMergeBlocks( BB, targetBB ) )
3          mergeBlocks( BB, targetBB ); //optimization
4      else
5          BB->replaceTerminal( Jump ); //optimization
6  }
7  void run( ) {
8      for ( BasicBlock* BB : graph ) {
9          if ( !BB )
10             continue;
11         switch ( BB>terminal( ) ) {
12             case Branch:
13                 …
14                 if ( BB->successor( 0 ) == BB->successor( 1 ) ) {
15                     convertToJump( BB, BB->successor( 0 ) );
16                     break;
17                 }
18             case Switch:
19                 for ( int i = 0; i < cases.size( ); i++ ) {
20                     // Analyze the target of cases
21                 }
22                 if ( cases.isEmpty( ) ) {
23                     convertToJump( BB, BB->fallThrough( ) );
24                     break;
25                 }
26         }
27     }
28 }
```

Overlooked Optimization Path

- The test case which triggers a new optimization path may be missed in edge coverage-guided fuzzing.
- 4 path executed, 3 seeds preserved
  - Path 9 →11→14→15→2→3
  - Path 9 →11→14→15→2→4→5
  - Path 9 →11→22→23→2→3
  - Path 9 →11→22→23→2→4→5 (missed)
- Overlook underlying vulnerabilities

```
1  void convertToJump( BasicBlock* BB, BasicBlock* targetBB ) {
2      if ( CanMergeBlocks( BB, targetBB ) )
3          mergeBlocks( BB, targetBB ); //optimization
4      else
5          BB->replaceTerminal( Jump ); //optimization
6  }
7  void run( ) {
8      for ( BasicBlock* BB : graph ) {
9          if ( !BB )
10             continue;
11         switch ( BB>terminal( ) ) {
12             case Branch:
13                 …
14                 if ( BB->successor( 0 ) == BB->successor( 1 ) ) {
15                     convertToJump( BB, BB->successor( 0 ) );
16                     break;
17                 }
18             case Switch:
19                 for ( int i = 0; i < cases.size( ); i++ ) {
20                     // Analyze the target of cases
21                 }
22                 if ( cases.isEmpty( ) ) {
23                     convertToJump( BB, BB->fallThrough( ) );
24                     break;
25                 }
26         }
27     }
28 }
```

Imbalanced Testing for Optimization Path

- Optimization paths with easily satisfied conditions undergo more testing, while those with challenging conditions are tested less.

- Number of times 4 paths were tested
  - Path 9 →11→14→15→2→3 (11)
  - Path 9 →11→14→15→2→4→5 (1)
  - Path 9 →11→22→23→2→3 (83)
  - Path 9 →11→22→23→2→4→5 (0)

```
1  void convertToJump( BasicBlock* BB, BasicBlock* targetBB ) {
2      if ( CanMergeBlocks( BB, targetBB ) )
3          mergeBlocks( BB, targetBB ); //optimization
4      else
5          BB->replaceTerminal( Jump ); //optimization
6  }
7  void run( ) {
8      for ( BasicBlock* BB : graph ) {
9          if ( !BB )
10             continue;
11         switch ( BB>terminal( ) ) {
12             case Branch:
13                 ...
14                 if ( BB->successor( 0 ) == BB->successor( 1 ) ) {
15                     convertToJump( BB, BB->successor( 0 ) );
16                     break;
17                 }
18             case Switch:
19                 for ( int i = 0; i < cases.size( ); i++ ) {
20                     // Analyze the target of cases
21                 }
22                 if ( cases.isEmpty( ) ) {
23                     convertToJump( BB, BB->fallThrough( ) );
24                     break;
25                 }
26         }
27     }
28 }
```

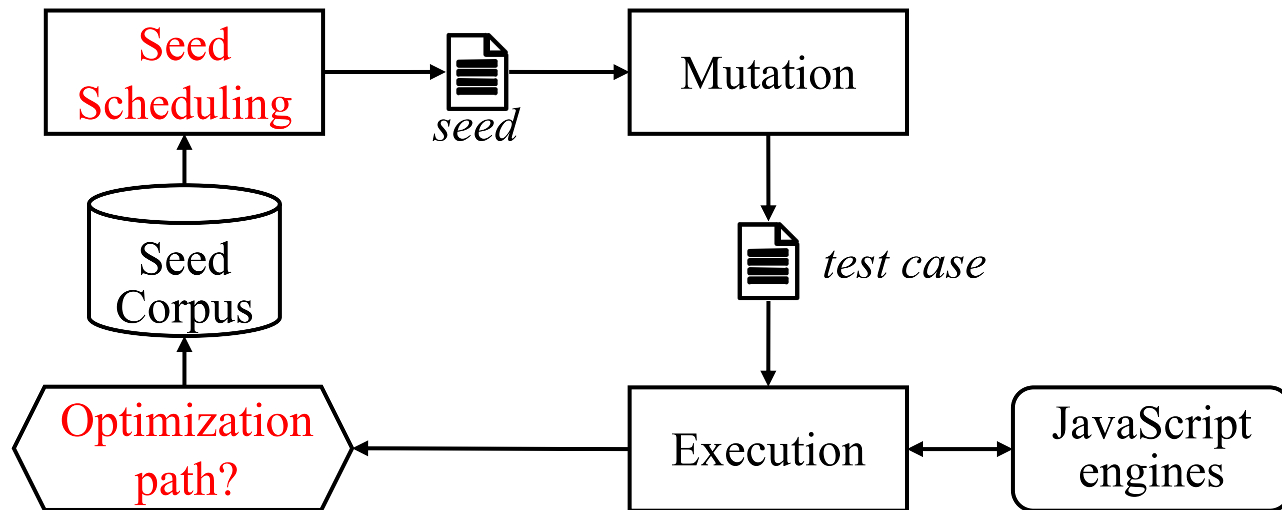
中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

Design

It is crucial to not only test up the optimization pass, but also to comprehensively test the optimization paths within that pass.

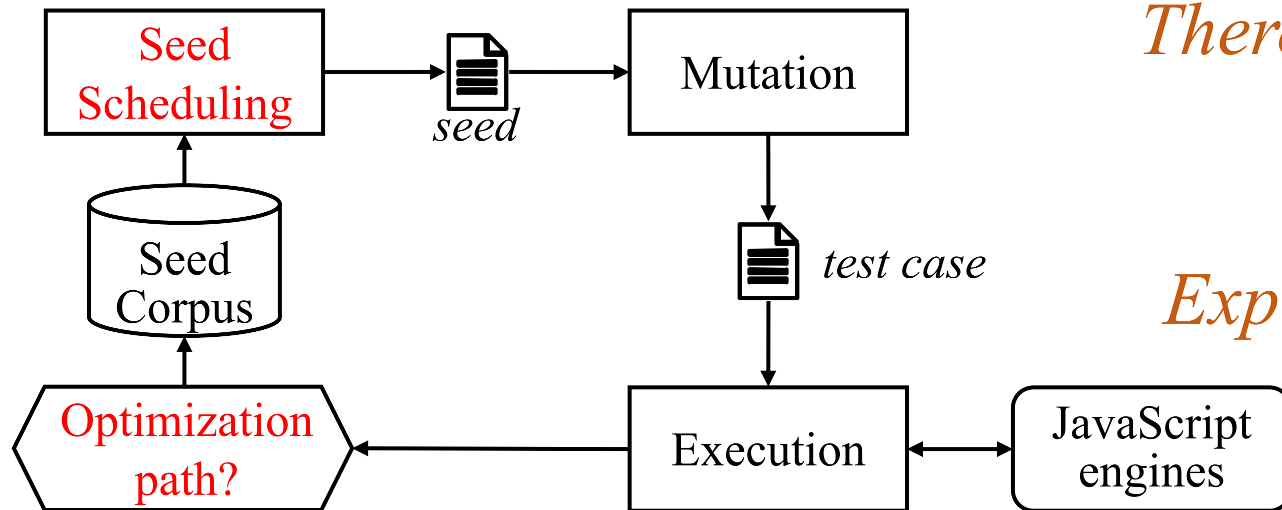
Optimization Trunk path

- Optimization path-guided fuzzing?
  - Enter Optimization ≠ Tigger Optimization
  - Overlooked Optimization Path
  - Imbalanced Testing for Optimization Path



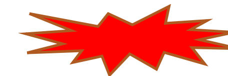
Optimization Trunk path

- Optimization path-guided fuzzing?
  - Enter Optimization ≠ Tigger Optimization
  - Overlooked Optimization Path
  - Imbalanced Testing for Optimization Path



There are inner nested loops in the optimization path!

Explosion in the number of seeds

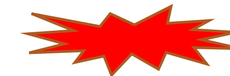


Optimization Trunk path

```
1  void convertToJump( BasicBlock* BB, BasicBlock* targetBB ) {
2      if ( CanMergeBlocks( BB, targetBB ) )
3          mergeBlocks( BB, targetBB ); //optimization
4      else
5          BB->replaceTerminal( Jump ); //optimization
6  }
7  void run( ) {
8      for ( BasicBlock* BB : graph ) {
9          if ( !BB )
10             continue;
11         switch ( BB>terminal( ) ) {
12             case Branch:
13                 …
14                 if ( BB->successor( 0 ) == BB->successor( 1 ) ) {
15                     convertToJump( BB, BB->successor( 0 ) );
16                     break;
17                 }
18             case Switch:
19                 for ( int i = 0; i < cases.size( ); i++ ) {
20                     // Analyze the target of cases
21                 }
22                 if ( cases.isEmpty( ) ) {
23                     convertToJump( BB, BB->fallThrough( ) );
24                     break;
25                 }
26         }
27     }
28 }
```

There are inner nested loops in the optimization path!

Explosion in the number of seeds



Optimization Trunk path

```
1  void convertToJump( BasicBlock* BB, BasicBlock* targetBB ) {
2      if ( CanMergeBlocks( BB, targetBB ) )
3          mergeBlocks( BB, targetBB ); //optimization
4      else
5          BB->replaceTerminal( Jump ); //optimization
6  }
7  void run( ) {
8      for ( BasicBlock* BB : graph ) {
9          if ( !BB )
10             continue;
11         switch ( BB>terminal( ) ) {
12             case Branch:
13                 …
14                 if ( BB->successor( 0 ) == BB->successor( 1 ) ) {
15                     convertToJump( BB, BB->successor( 0 ) );
16                     break;
17                 }
18             case Switch:
19                 for ( int i = 0; i < cases.size( ); i++ ) {
20                     // Analyze the target of cases
21                 }
22                 if ( cases.isEmpty( ) ) {
23                     convertToJump( BB, BB->fallThrough( ) );
24                     break;
25                 }
26         }
27     }
28 }
```

```
1  void convertToJump( BasicBlock* BB, BasicBlock* targetBB ) {
2      if ( CanMergeBlocks( BB, targetBB ) )
3          mergeBlocks( BB, targetBB ); //optimization
4      else
5          BB->replaceTerminal( Jump ); //optimization
6  }
7  void run( ) {
8      for ( BasicBlock* BB : graph ) {
9          if ( !BB )
10             continue;
11         switch ( BB>terminal( ) ) {
12             case Branch:
13                 …
14                 if ( BB->successor( 0 ) == BB->successor( 1 ) ) {
15                     convertToJump( BB, BB->successor( 0 ) );
16                     break;
17                 }
18             case Switch:
19
20                     Ignore the nested inner loops
21
22                 if ( cases.isEmpty( ) ) {
23                     convertToJump( BB, BB->fallThrough( ) );
24                     break;
25                 }
26         }
27     }
28 }
```

Optimization Trunk path (OptPath)

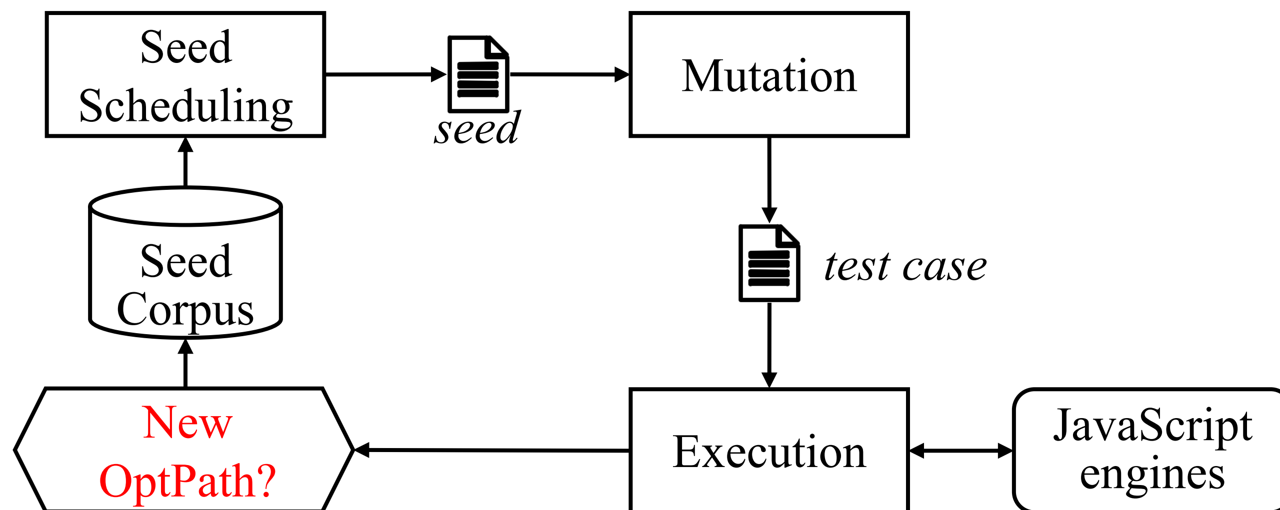
- The OptPath represents a path within the outer loop, which ignores the nested inner loop.

- OptPath
  - 9→11→12→14→15→2→3→26→27
  - 9→10→27 *early exit*

```
1  void convertToJump( BasicBlock* BB, BasicBlock* targetBB ) {
2      if ( CanMergeBlocks( BB, targetBB ) )
3          mergeBlocks( BB, targetBB ); //optimization
4      else
5          BB->replaceTerminal( Jump ); //optimization
6  }
7  void run( ) {
8      for ( BasicBlock* BB : graph ) {
9          if ( !BB )
10             continue;
11         switch ( BB>terminal( ) ) {
12             case Branch:
13                 …
14                 if ( BB->successor( 0 ) == BB->successor( 1 ) ) {
15                     convertToJump( BB, BB->successor( 0 ) );
16                     break;
17                 }
18             case Switch:
19
20                 Ignore the nested inner loops
21
22                 if ( cases.isEmpty( ) ) {
23                     convertToJump( BB, BB->fallThrough( ) );
24                     break;
25                 }
26         }
27     }
28 }
```

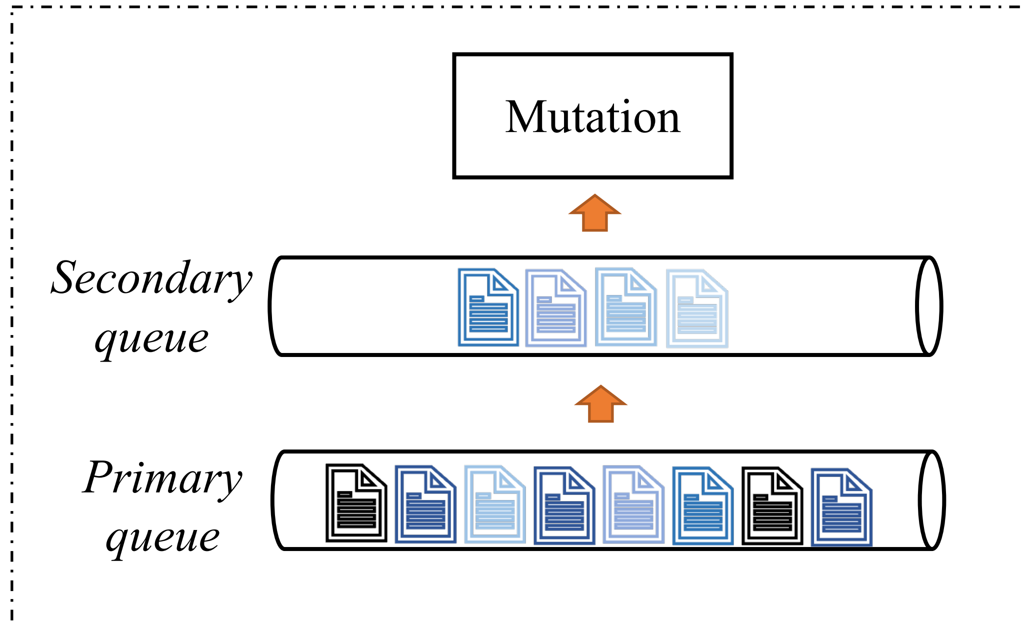
Seed Preservation

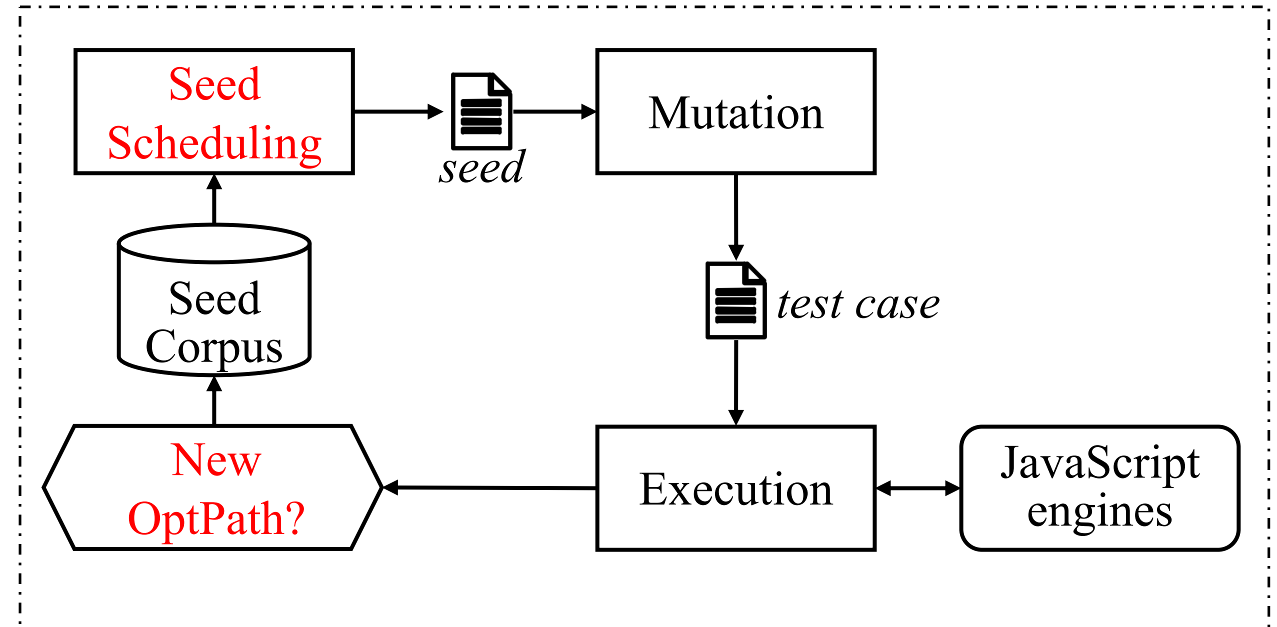
- If a test case triggers a new OptPath, it is preserved as a seed.



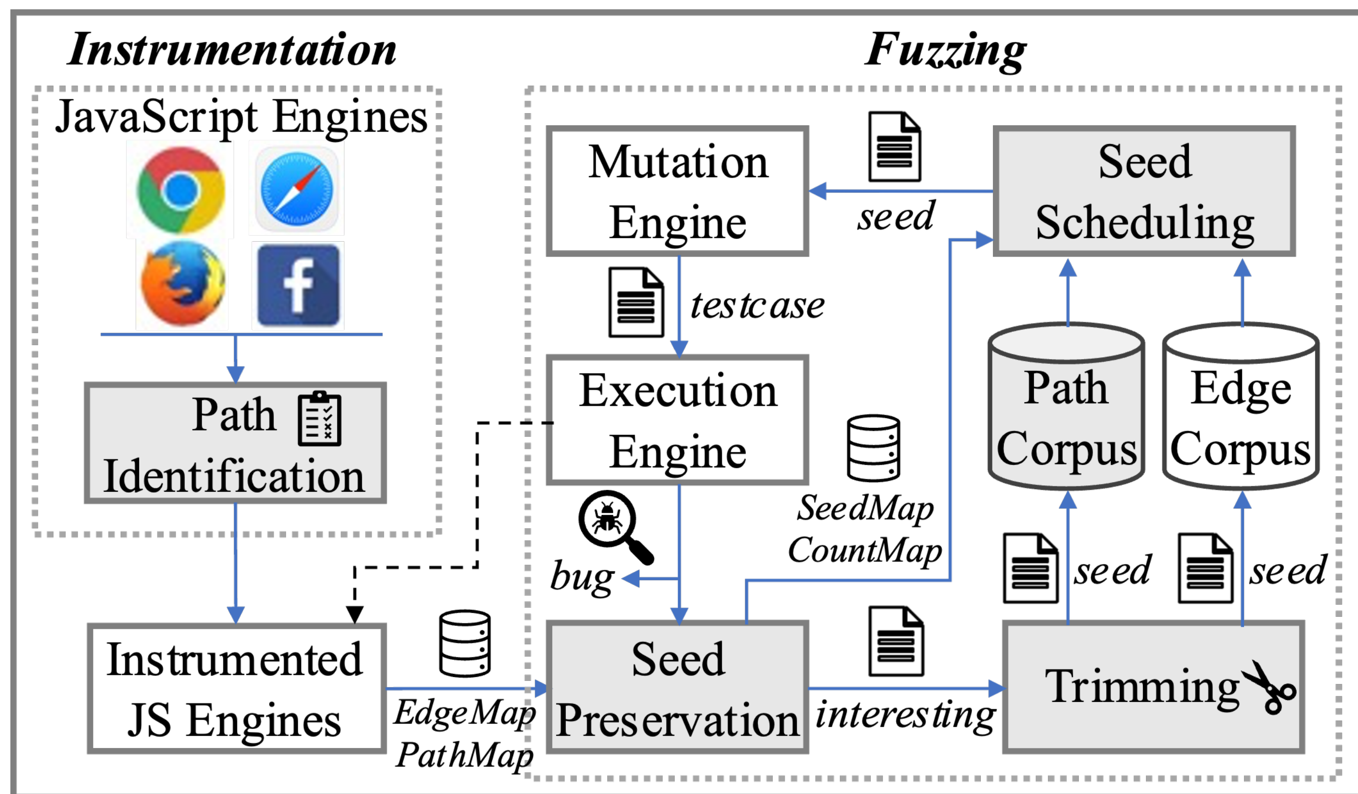
Seed Scheduling

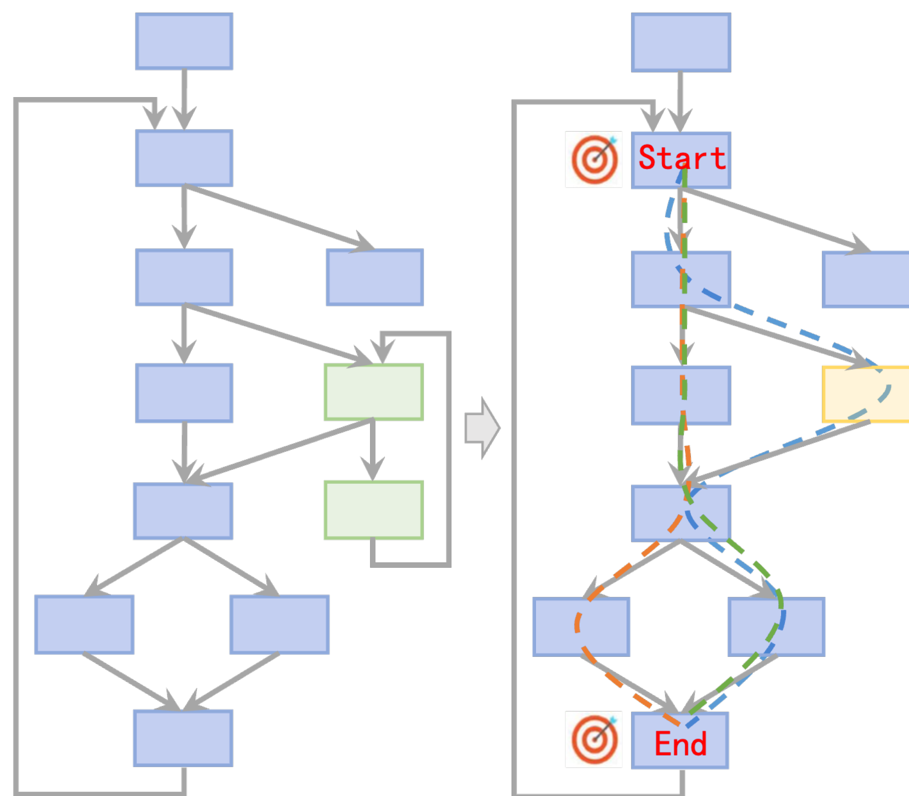
- A two-tier seed queue
- Make the OptPaths which are tested less get more chance to be tested.





Workflow





Discussion

- Early exit branches in OptPaths
  - Seed Scheduling
  - Exit branches that have been tested more frequently are not selected for mutation.
- Ignored inner loops
  - When outer loops undergo frequent testing, a majority of paths within inner loops also receive more comprehensive testing.
  - A more fine-grained feedback *vs.* Additional overhead

Evaluation

- Setup
  - Baseline
    - Superion
    - DIE
    - Fuzzilli
    - FuzzJIT
  - Target
    - V8 in Google Chrome
    - JavaScriptCore in Safari
    - SpiderMonkey in Firefox
    - Hermes developed by Facebook
- Bug Finding Ability
  - Bug List
  - Short-term and long-term experiment
- Exploration of OptPaths
- Ablation Experiment
  - Seed Preservation
  - Seed Scheduling
  - Trimming
  - Threshold
  - Switching Strategy
- Code Coverage

Bug Finding Ability

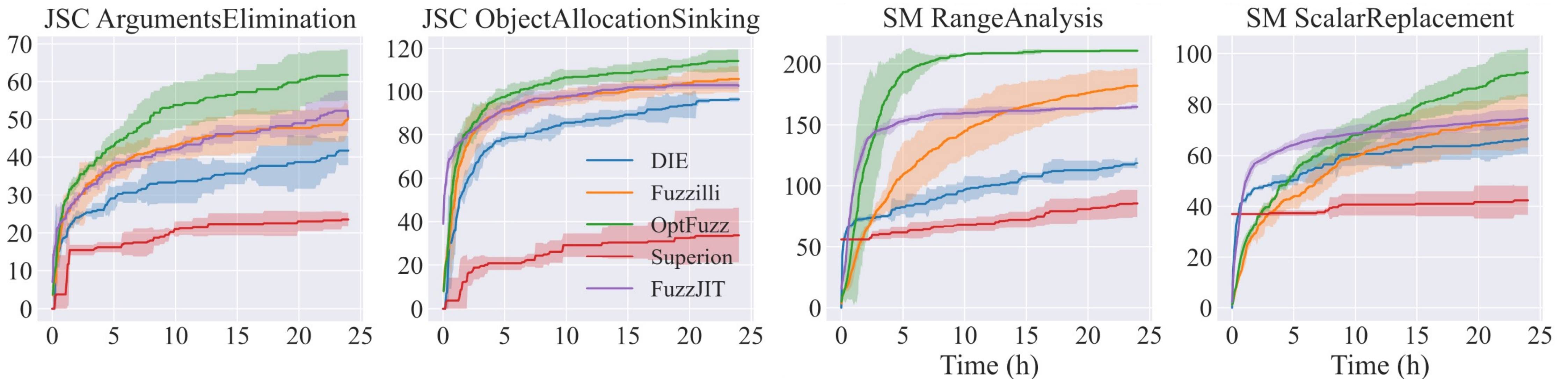
- 36 new bugs
  - 26 bugs are confirmed
  - 4 security-sensitive bugs, 3 CVE
  - 1 new bug in V8
  - 17 new bugs in JavaScriptCore
  - 5 new bugs in SpiderMonkey
  - 13 new bugs in Hermes

Engine	ID	Issue	JIT	Status	Description
JavaScriptCore	1	CVE-2023-38595 *	✓	Fixed	Integer overflow caused by multiplication optimization in b3
	2	Issue 255512	✓	Fixed	StringConstructor function inlining is incorrect
	3	crash	✓	Duplicate	B3 values become orphaned values
	4	Issue 256508		New	Error.stackTraceLimit can not be updated correctly
	5	Issue 256832		Fixed	Assertion Failure: m_setOp == CharacterClassSetOp::Default
	6	Issue 257949	✓	Fixed	JSC computes incorrectly in DFG
	7	Issue 258559	✓	Fixed	regExpFuncExec throw different messages in LLint and DFG
	8	Issue 256022 *	✓	Fixed	OOB Reading caused by implementation of For-In in FTL
	9	Issue 258518	✓	New	For-In in DFGSpeculativeJIT may destroy IndexGPR
	10	Issue 258552	✓	New	iterator_next becomes undefined after bailout from DFG
	11	Issue 263520	✓	New	AbstractInterpreter handles GetMyArgumentByVal incorrectly when callee is inlined
	12	Issue 263647	✓	Confirmed	Function.caller returns null when callee is inlined into apply function
	13	Issue 263954		Fixed	JSC should throw an exception when BigUint64Array copy value from Int32Array
	14	Issue 263881	✓	New	BitURShift is eliminated when toString has an effect
	15	Issue 263882	✓	New	ValueMod is eliminated incorrectly
	16	Issue 264034	✓	Fixed	Host function isPureNaN is inlined into true in handleIntrinsicCall
	17	Issue 264078	✓	New	Abstract Interpreter computes wrong value for GetLocal
	18	Issue 265978	✓	New	Error Object has more properties in DFG
SpiderMonkey	19	Issue 1830107	✓	Fixed	Assertion failure for unexpected range for value in LInt32ToIntPtr
	20	Issue 1833133	✓	Fixed	IC stub for arguments.callee is used even if callee has been redefined
	21	Issue 1833294	✓	New	Differential testing generates different values when target is Error
	22	Issue 1834038	✓	Fixed	Assertion failure: cx_→ hadResourceExhaustion in JIT
	23	Issue 1835579	✓	Confirmed	Differential testing generates different values when target is gcNumber
V8	24	Issue 14055	✓	New	v8 prints non-deterministic results for Worker with/without turbofan
	25	crash		Duplicate	Check failed in src/handles/maybe-handles.h
	26	crash		Duplicate	Check failed in src/objects/string-tq-inl.inc
	27	crash		Duplicate	Debug check failed: (*p).ptr() != to_check_.ptr()
Hermes	28	Issue 1033		Confirmed	Integer overflow for vector capacity in Hermes
	29	CVE-2023-38542 *	✓	Fixed	Use after free of hermes::PhiInst object in SimplifyInst
	30	Bug...8950 *	✓	Fixed	Hermes accesses uninitialized memory in Symbol.toStringTag
	31	Issue 1199	✓	Fixed	Hermes should throw an exception when implicitly convert Int to BigInt
	32	Issue 1200	✓	Fixed	SimplifyBinOp in InstSimplify generates wrong result
	33	Issue 1203	✓	Fixed	The generator function's body is executed incorrectly when next() method is not called
	34	Issue 1212	✓	Fixed	Hermes should throw an exception when BigInts use unsigned shift
	35	Issue 1215	✓	Confirmed	SimplifySwitchInst in SimplifyCFG ignores negative zero
	36	Issue 976		Confirmed	Too many handles in recursive invocation of stringPrototypeReplace()
	37	Issue 977		Fixed	Hermes crashes when call AggreageError constructor.
	38	Issue 988		Fixed	Assertion failure when executing StartGenerator
	39	Issue 1006		Fixed	Assertion failed: hasNamedOrIndex failed
	40	Issue 1015		Fixed	Assertion failed: defineOwnProperty threw in _setOwnIndexedImpl

* in *Issue* column means this bug is security sensitive.

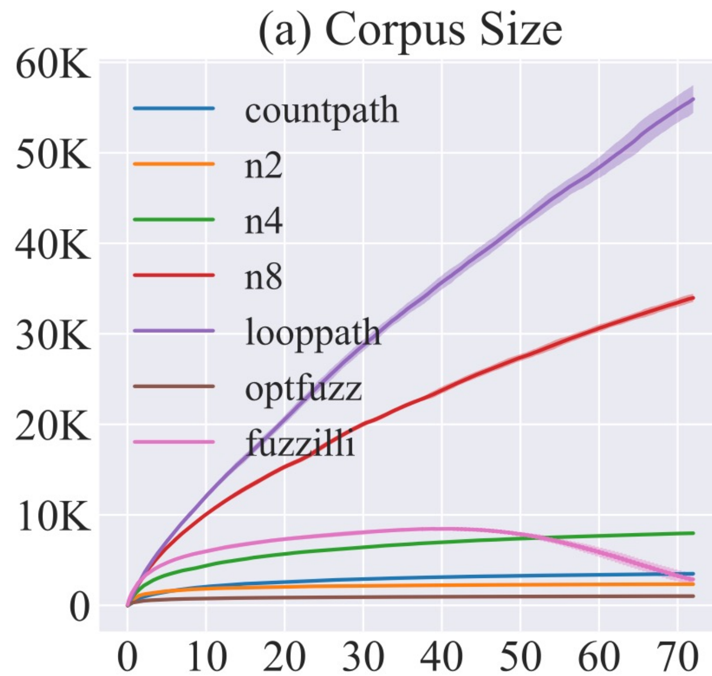
Exploration of OptPaths

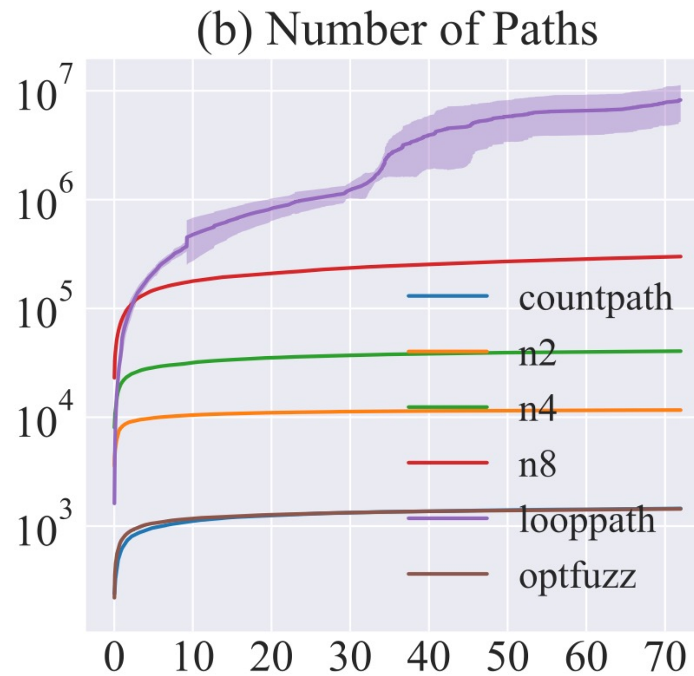
- OptFuzz can explore more OptPaths than other fuzzers.
- Compared to OptFuzz, Superion, DIE, Fuzzilli and FuzzJIT were able to explore 38.6%, 67.5%, 86.0% and 81.9% of the OptPaths, respectively.



Ablation Experiment

- Seed Preservation
  - Among various strategies, OptFuzz yield the fewest number of seeds but find the most number of bugs.





Seed Preservation		
Fuzzer	Bug	*p-value*
countpath	1	**0.007**
n2	2	**0.030**
n4	0	**0.002**
n8	0	**0.002**
looppath	0	**0.002**
Fuzzilli	2	**0.016**
OptFuzz	4	1.000

Conclusion

- This paper summarizes three observations:
  - Enter Optimization ≠ Tigger Optimization
  - Edge coverage-guided fuzzing may overlook test cases that trigger new optimization paths.
  - Imbalanced Testing for Optimization Path
- A novel fuzzing technique that leverage optimization trunk paths to guide fuzz testing.
- Through extensive testing, OptFuzz found 36 new bugs in JavaScriptCore, V8, SpiderMonkey, and Hermes.

Q & A

wangjiming21b@ict.ac.cn