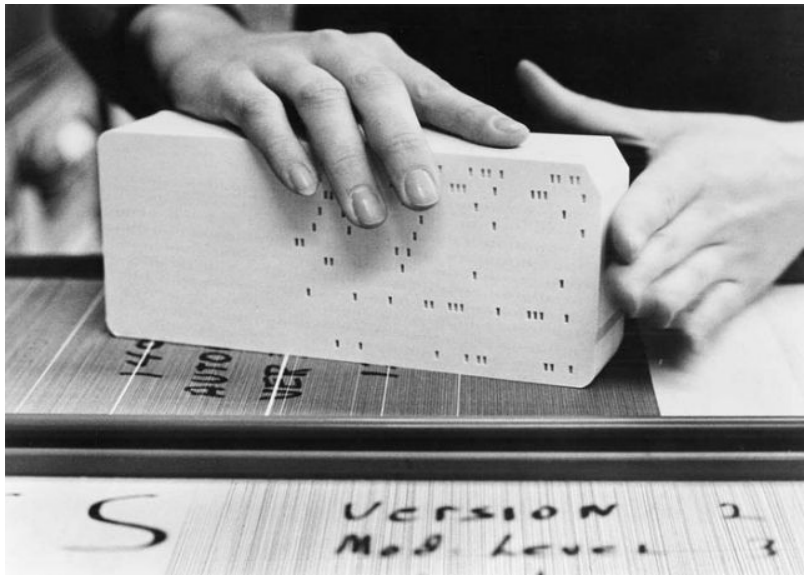
Data Coverage for Guided Fuzzing

Mingzhe Wang, Jie Liang, Chijin Zhou, Zhiyong Wu, Jingzhou Fu, and Zhuo Su, *Tsinghua University;* Qing Liao, *Harbin Institute of Technology;* Bin Gu, *Beijing Institute of Control Engineering;* Bodong Wu, *Huawei Technologies Co., Ltd;* Yu Jiang, *Tsinghua University*

USENIX '24

Background - Fuzzing

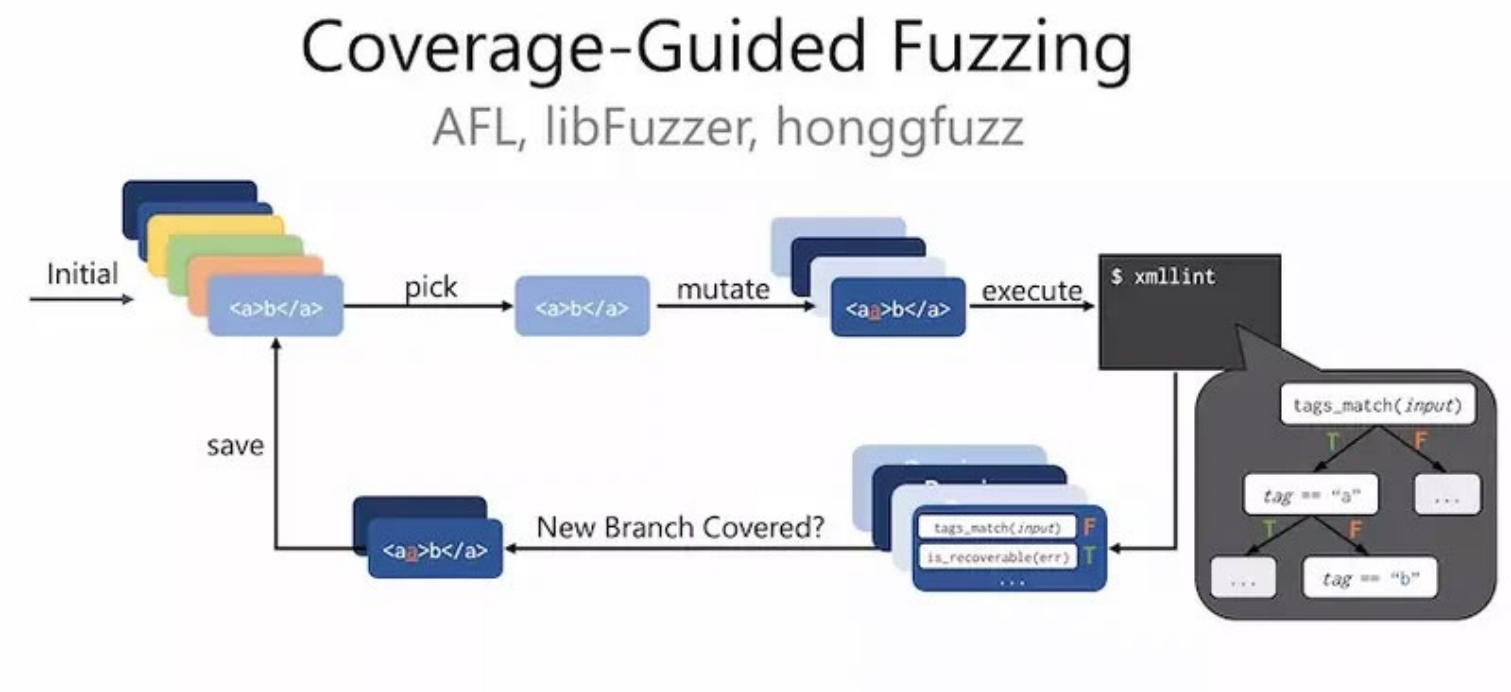
- Black box software testing technique
  - Search for implementation bugs using malformed/semi-malformed data injection in an automated fashion



https://www.ibm.com/history/punched-card
https://owasp.org/www-community/Fuzzing

Background - Guided Fuzzing

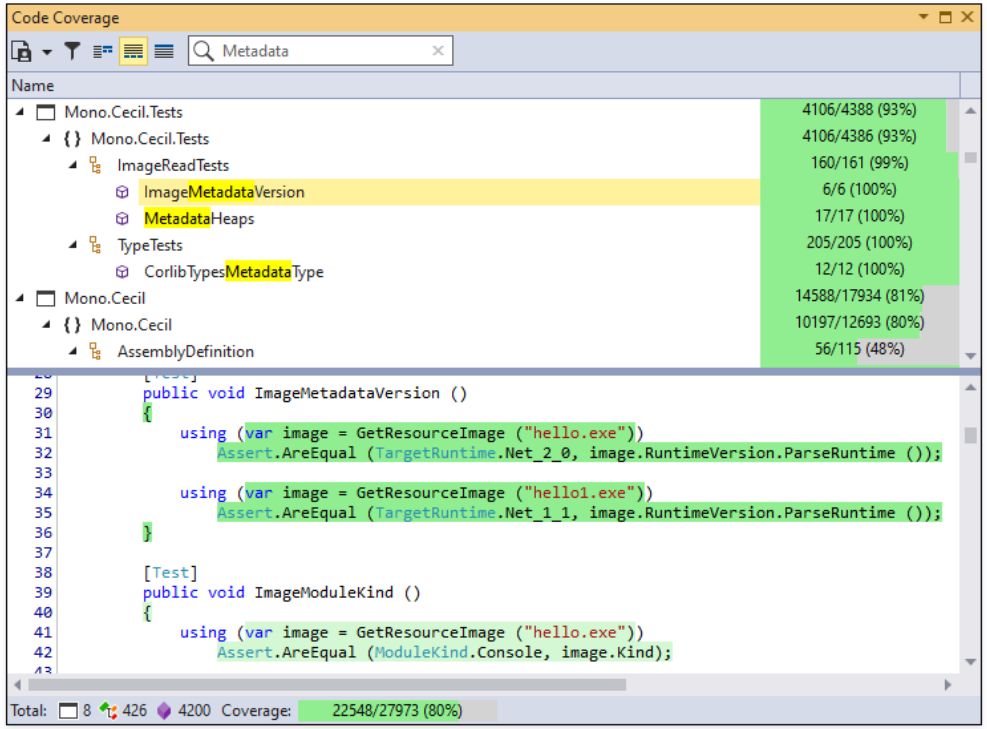
- How do we generate the input data for fuzzing?



https://www.code-intelligence.com/blog/caroline-lemieux-expanding-fuzzing

Background - Why Code Coverage?

- Determine how much source code gets executed during the run of a program.



https://docs.devexpress.com/CodeRushForRoslyn/115735/unit-testing-assistance/analyze-code-coverage

Motivation

The use of code coverage in guided fuzzing has been shown to be highly effective, …
code coverage in guided fuzzing has been responsible for the discovery of **40,000 bugs** in **650 popular projects**.
…

However, code coverage is *not* sufficient to fully reflect a program's semantics.

…

when a compiler converts a source into machine code, the algorithms are transformed into *machine instructions*, while the data structures are encoded as *constant data*.

Motivation Breakdown

- #1. Code Coverage is not sufficient.
- #2. Data Coverage is as important because data section contains the data structures.
- We propose and implement a guided fuzzing technique using data coverage.

Vocabulary used

- Immediate Value – Data embedded in code. (0xdeadbeef)

- Static Value –String literals, static variables, and global variables. (e.g., .data section)

Limitation of Code Coverage

- Constraint Solving
  - Concolic Execution
    - State explosion (8^m paths to solve (Figure 2))
  - Intelligent branch solving
    - Require human intervention
    - Table 2

```
int cmsstrcasecmp(const char* s1, const char* s2) {
  // [...]
  while (toupper(*us1) == toupper(*us2++))
    if (*us1++ == '\0')
      return 0;
}
```

Figure 2: Manual string comparison in LCMS.

Table 1: Hard-Coded Buffer Comparison Routines in AFL++

Program	Count	Example
C++	4	`operator==(string&, char*)`
C	10	`strcmp`
apache	3	`ap_cstr_casecmp`
busybox	1	`memcmpct`
curl	5	`Curl_safe_strcasecompare`
glib	9	`g_strcasecmp`
lcms	1	`cmsstrcasecmp`
libxml	7	`xmlStrcasecmp`
openssl	4	`OPENSSL_memcmp`
samba	1	`strcsequal`
Total	45	

Data: Transition Table (42 KiB, 17% of binary size)

```
static uint8_t yy_ec[256], yy_meta[53];
static int16_t yy_accept[1780], yy_base[2240], yy_nxt[7789], yy_chk[7789];
```

Code: State Transition

```
do {
  // Read the input byte.
  uint8_t yy_c = yy_ec[*yy_cp];
  // Drive the state machine.
  while (yy_chk[yy_base[yy_state] + yy_c]
          != yy_state) {
    yy_state = (int)yy_def[yy_state];
    if (yy_state>=1780) yy_c=yy_meta[yy_c];
  }
  yy_state=yy_nxt[yy_base[yy_state]+yy_c];
  ++yy_cp;  // Bump to the next byte.
} while (yy_state != 1779);
```

Code: Action Execution

```
yy_state = yyg->
  yy_last_accepting_state;
yy_act = yy_accept[yy_state];
switch (yy_act) {
  case 1: return DST;
  case 2: return SRC;
  case 3: return LINK;
  // Many cases...
  case 186: return LEX_ERROR;
  case 187: ECHO;
  default: YY_FATAL_ERROR();
}
```

Figure 3: Finite-state machine implementation of libpcap. Based on the predefined *transition table* (shown in top), the *transition logic* (shown in the left) alters the machine's state according to input. When a final state is entered, the *execution logic* (shown in the right) performs user-defined actions. Data references in code are highlighted in yellow.

Benefit of Data Coverage

- Static Values
  - Complex structures (lookup tables, binary trees, di-graph)
    - Figure 3
  - Data coverage can assist with this problem by tracking data usage

- Immediate Values

```
#define cmsMagicNumber  0x61637370     // 'acsp'

// Validate file as an ICC profile
if (_cmsAdjustEndianess32(Header.magic) != cmsMagicNumber) {
  cmsSignalError(... "not an ICC profile, invalid signature");
  return FALSE;
}
```

Figure 1: Predicate in LCMS testing for magic numbers.

Overall Design

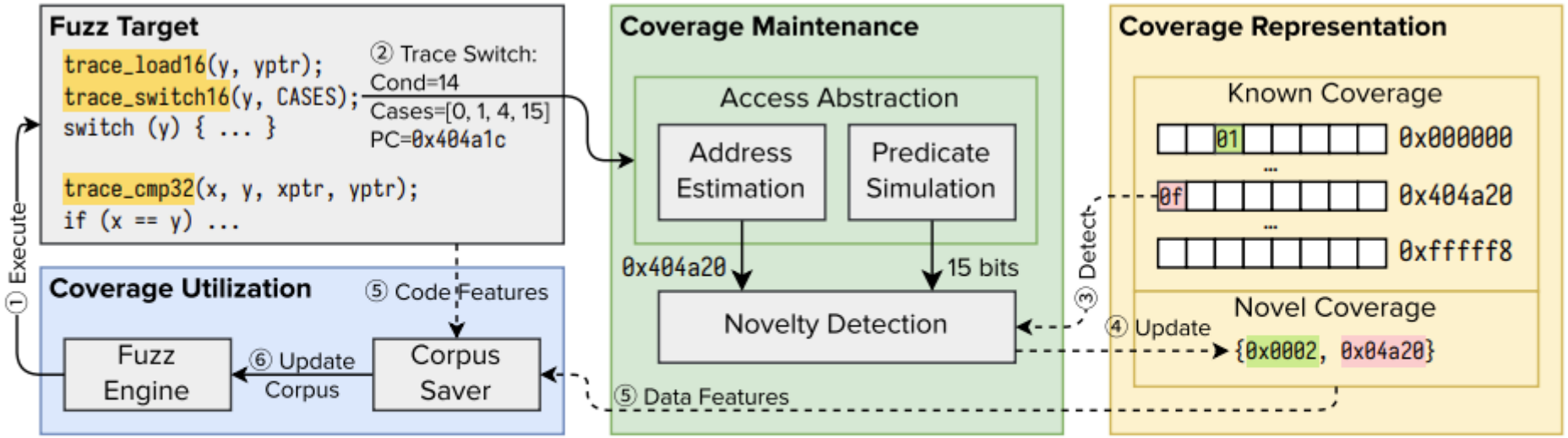
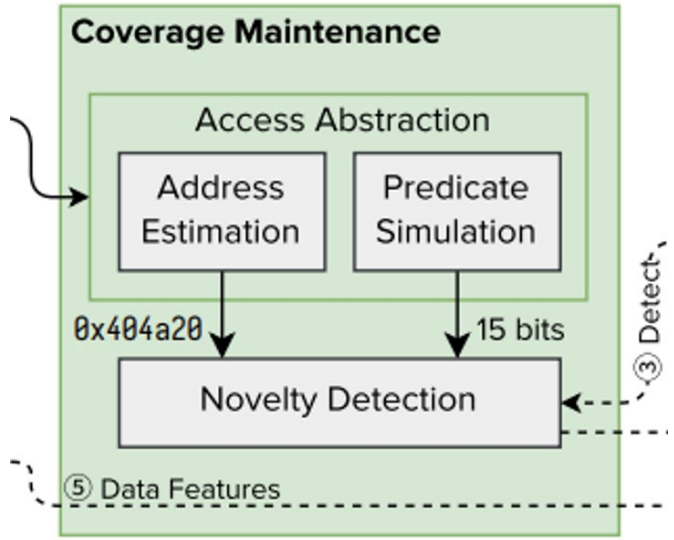


Figure 4: Fuzzing with data coverage. The *fuzz target* is instrumented during compilation. During program execution, the *coverage maintenance* component intercepts the program's data accesses and updates the coverage representation. After the test case execution completes, the *coverage utilization* component adjusts the fuzzing directions based on the novel coverage.

Implementation

- Coverage Maintenance
  - Data access is represented as tuple: (address, length)
  - Address:
    - Static Values: Instrument load operations (search for load addresses)
    - Immediate Values: Program Counter
  - Length:
    - Predicate Emulation



Predicate Emulation?

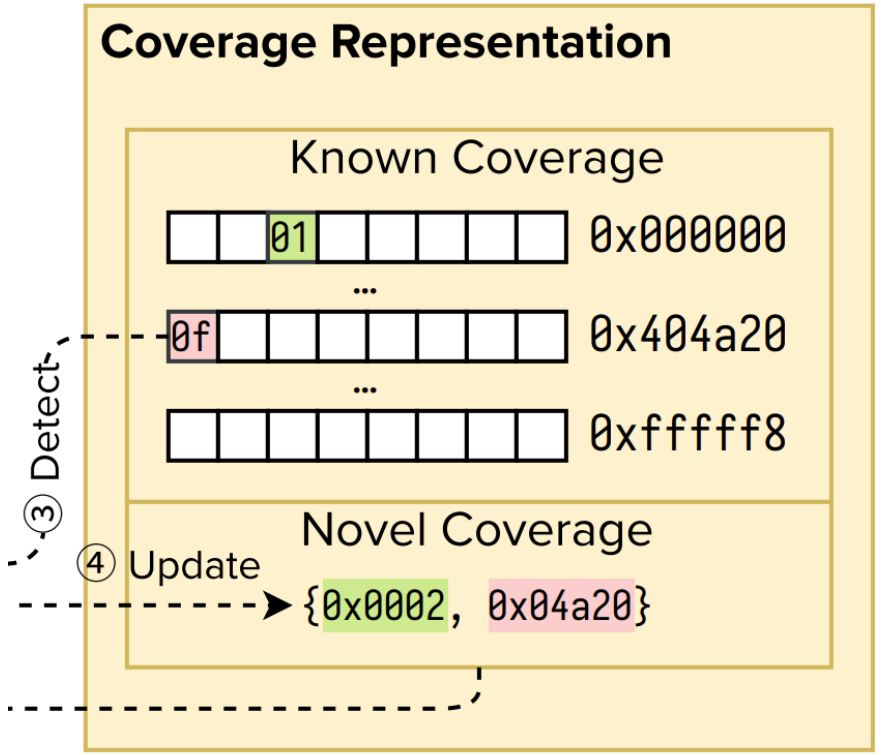
- 0b0001 == 0b1110 ?
- 0b1110 == 0b1110 ?
- 0b0001 > 0b 1110 ?

- 1. *"Effectively-used bit" of equivalence relations is the number of equal bits in the corresponding positions for both operands"*

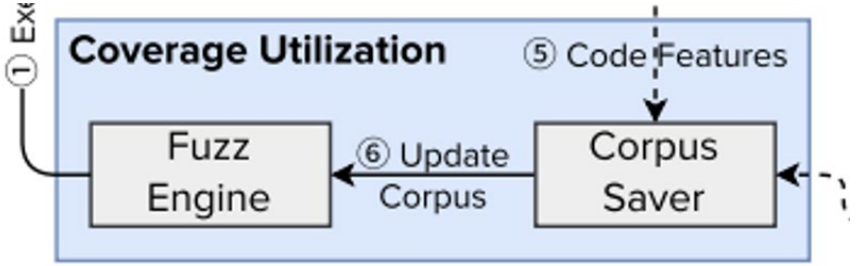
- 2. *"Effectively-used bits" of partial order relations is the number of consecutive equal bits of both operands starting from the most significant bit"*

Implementation

- Coverage Representation
  - Known Coverage: Array[addr] = length
  - Novel Coverage: Set{addr}



Implementation



- Coverage Utilization
  - Reduce the **number of saved seeds**
  - New data reads 8 bits while old one reads 7 bits
    - Directly replace the 7-bit seed with the new one.

Algorithm 1: Refinement-based Corpus Update

Input: novel code coverage C
Input: novel data coverage D
Input: code feature summaries S
Input: testcase t and code feature summary s
Data: M: maps a data feature to the saved seed
if $C = \emptyset \wedge D = \emptyset$ **then** *// No discovery.*
  return;
if $C = \emptyset$ **then**
  // Data discovery only: try refinement.
  for $f \in D$ **do**
    if $S_{M_f} = s$ **then**
      REPLACESEED(M_f, t);
      for $f' \in D$ **do**
        $M_{f'} \leftarrow M_f$;
      return;
// Refinement is not possible: save a new seed.
$n \leftarrow$ SAVESEED(t, s);
for $f \in D$ **do**
  $M_f \leftarrow n$;

Overall Design

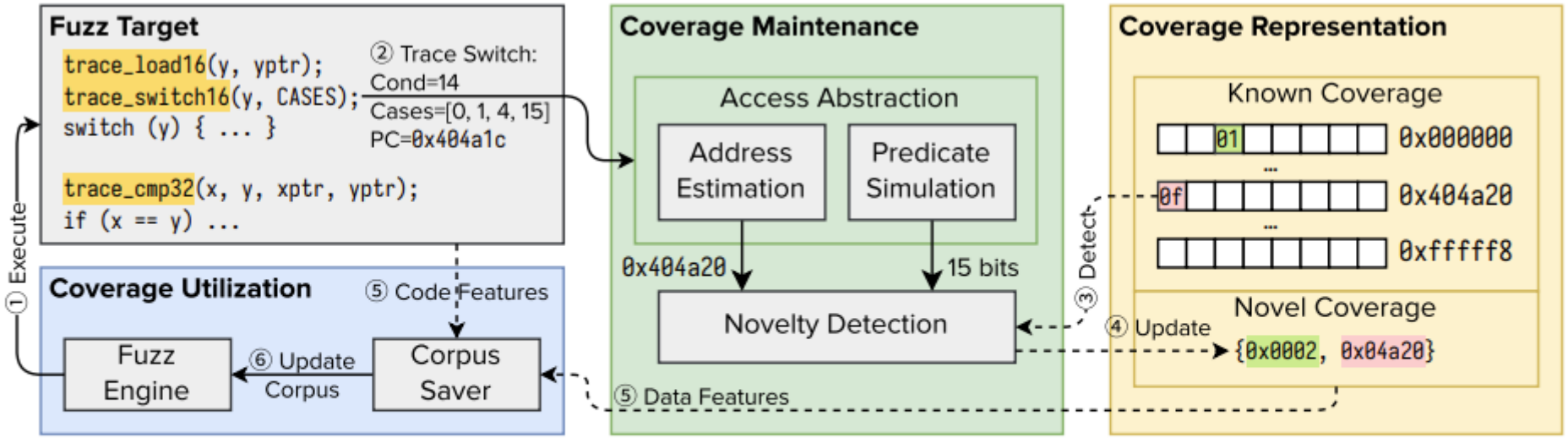


Figure 4: Fuzzing with data coverage. The *fuzz target* is instrumented during compilation. During program execution, the *coverage maintenance* component intercepts the program's data accesses and updates the coverage representation. After the test case execution completes, the *coverage utilization* component adjusts the fuzzing directions based on the novel coverage.

Various Data Accesses

Table 2: Data Access Semantics and Their Address-Length Estimations

Data	Access Semantics	Example	Address Estimation	Length Estimation
Immediate Value	Integer Comparison	`x == 0`	Program Counter	Predicate Emulation
	Switch Statement	`switch (x) case 1:`	PC + Case Index	Switch Emulation
	Trivial Value	`y = x + 2`	Not Available (Represented by Code Coverage)	
Static Value	Load-Based Comparison	`p[x * y] - 1 > z`	Static Analysis	Predicate Emulation
	Region Comparison	`memcmp(p, q, 5)`	Base Pointer	Region Emulation
	Simple Load	`*p`	Pointer	Data Size

Experiment

- WingFuzz (proposed)
  - Based on libFuzzer (LLVM)
  - LLVM-14
- FuzzBench (default setup)
  - 19 target programs
  - 23 hours run
  - Each trial repeated 20 times

Experiment

- RQ1: Can data coverage improve fuzzing performance?
- RQ2: Is data coverage orthogonal to prior techniques?
- RQ3: How does individual component contribute?
- RQ4: What is the runtime overhead of data coverage?
- RQ5: Will the extra guidance cause state explosion?

Performance

Table 3: FuzzBench Evaluation Summary

Fuzzer	Average Score	Average Rank
WingFuzz	**98.31**	3.08
AFL++ [9]	96.91	**3.06**
Honggfuzz [31]	96.26	4.12
Entropic [5]	94.71	4.03
MOPT [21]	93.10	5.79
Eclipser [8]	92.86	6.31
AFLSmart [25]	92.84	5.28
AFL [37]	92.30	5.16
AFLFast [6]	88.77	7.70
libFuzzer [32]	87.65	7.09
LAFIntel [15]	86.47	8.17
FairFuzz [17]	84.67	7.44

- WingFuzz is based on libFuzzer.
- The full report can be found at https://www.fuzzbench.com/reports/experimental/2022-10-08-wingfuzz.

Table 4: Performance of WingFuzz Compared to libFuzzer

Benchmark	libFuzzer 23h Cov.	WingFuzz 23h Cov.	WingFuzz's Cov. Gains	Hours to 23h Cov.
sqlite3	11449.5	18142.0	+58.45%	0.25
freetype2	8173.5	12019.5	+47.05%	1.00
libxslt	8526.0	11446.0	+34.25%	0.25
woff2	987.0	1155.0	+17.02%	0.25
openthread	2247.0	2585.0	+15.04%	7.75
curl	8185.5	9046.5	+10.52%	0.25
libjpeg	1865.5	2046.5	+9.70%	1.00
libxml2	7885.0	8548.5	+8.41%	2.25
harfbuzz	7206.5	7760.0	+7.68%	1.00
mbedtls	2367.5	2527.5	+6.76%	0.50
php	15880.5	16785.5	+5.70%	0.25
proj4	4360.0	4512.0	+3.49%	4.00
bloaty	5723.0	5766.5	+0.76%	19.75
jsoncpp	519.0	518.0	-0.19%	/
re2	2566.0	2555.0	-0.43%	/
zlib	463.0	461.0	-0.43%	/
Average			+13.99%	2.96

Orthogonality

- AFL++ includes various data-sensitive fuzzing techniques

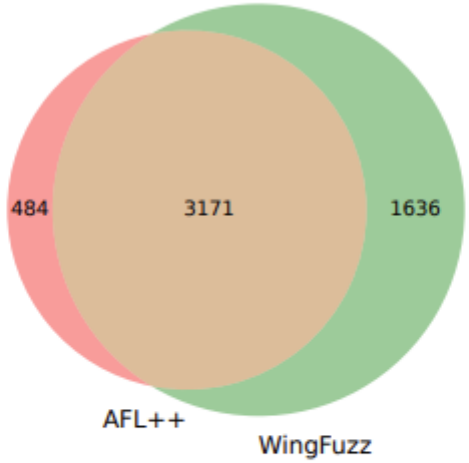


Figure 6: Extra coverage of WingFuzz and AFL++ over the baseline libFuzzer.

Orthogonality - cont

- Identified two new bugs in Little-CMS
  - Popular package integrated into software such as:
    - Chromium
    - Firefox
    - OpenJDK
  - Issue #373 and Issue #374
  - Google OSS-Fuzz , AFL++ continuous fuzz since 2016 + manual audit in 2018

```
// BUG happens here.
nSamples = satoi(cmsIT8GetProperty(it8, "NUMBER_OF_FIELDS"));

const char* cmsIT8GetProperty(cmsHANDLE hIT8, const char* Key) {
  IsAvailableOnList(GetTable(it8)->HeaderList, Key, ...)
}

bool IsAvailableOnList(KEYVALUE* p, const char* Key, ...) {
  // Enumerates the input data by walking a linked list.
  for (; p != NULL; p = p->Next) {
    // String camparison, see Figure 2.
    if (cmsstrcasecmp(Key, p->Keyword) == 0) break;
  }
}
```

(a) Buggy code: linked list walk using static data.

```
NUMBER_OF_FIELDS 4
S
BEGIN_DATA_FORMAT
```

```
I R G B
END_DATA_FORMAT
I
NUMBER_OF_FIELDS 8
```

(b) Bug-triggering input with multiple key-value records.

Figure 7: Case study: previously-unknown bug of Little-CMS.

Additional bugs identified on Serenity OS

- Google's OSS-Fuzz cluster continuously fuzzed it for more than two years (found 140 bugs)

- Using default initial seed from OSS-Fuzz + duration of 23 hours
  - Identified 26 bugs in 17 components of the newest version of the OS (2023-03-20)

Table 5: New Bugs in Serenity OS Detected by WingFuzz

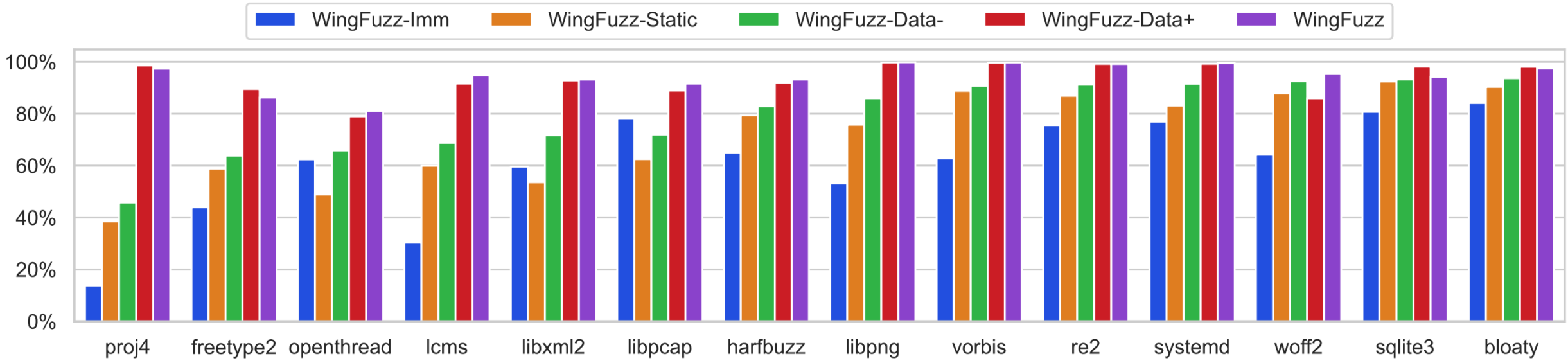
Kind	Issue ID
Assertion failure	17936, 17938, 17939, 18303, 18305–18307, 18310, 18316–18321
Resource exhaustion	17937, 18309, 18312–18315
Memory safety	18036, 18044, 18302, 18304, 18324–25
Total	26 bugs

Contribution of each Component

Table 6: Fuzzing Guidance of WingFuzz Variants

Variant	Data Guidance		Code Guidance		Coverage
	Static	Immediate	Spatial	Temporal	
WingFuzz-Imm		Partial			60.78
WingFuzz-Static	✓				71.91
WingFuzz-Data^{-}	✓	Partial			79.25
WingFuzz-Data^{+}	✓	✓	✓		93.73
WingFuzz	✓	✓	✓	✓	94.48

Contribution of each Component



- Occasionally WingFuzz-Data+ outperforms WingFuzz !

Execution Overhead

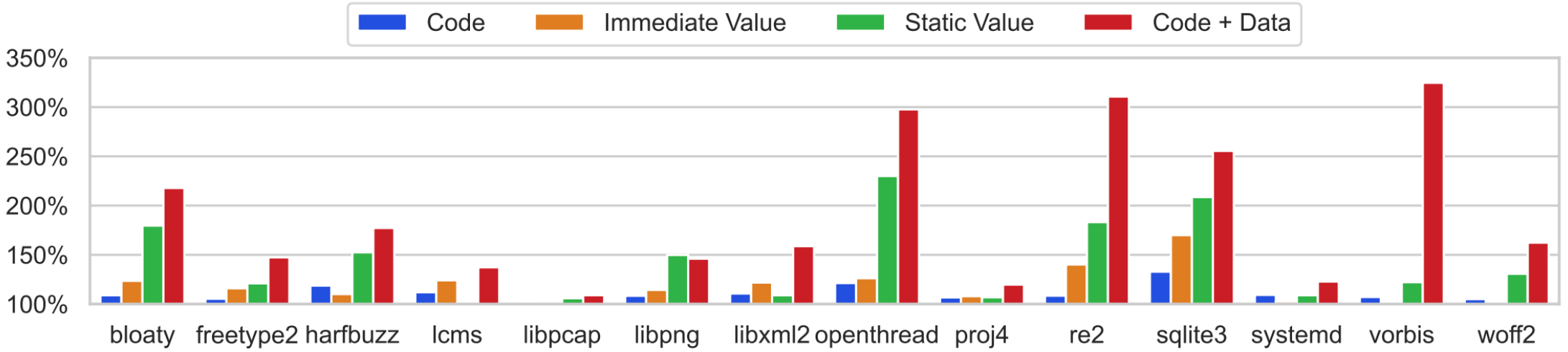


Figure 9: Normalized execution duration of programs in different instrumentation modes. Lower bars indicate better performance. For example, a value of "200%" indicates that the program executes at half the speed of the baseline version.

- Fuzzing throughput reduction of 34%

Execution Overhead

- End-to-end fuzzing performance
  - For short trials (15 minutes)
  - WingFuzz demonstrated highest coverage score
    - WingFuzz: avg. Cov 97.0 avg. rank 2.5
    - AFL++: avg. Cov 96.0 avg. rank 2.7
    - libFuzzer: avg. Cov 87.7 avg. rank 7.2

Queue Size and State Explosion

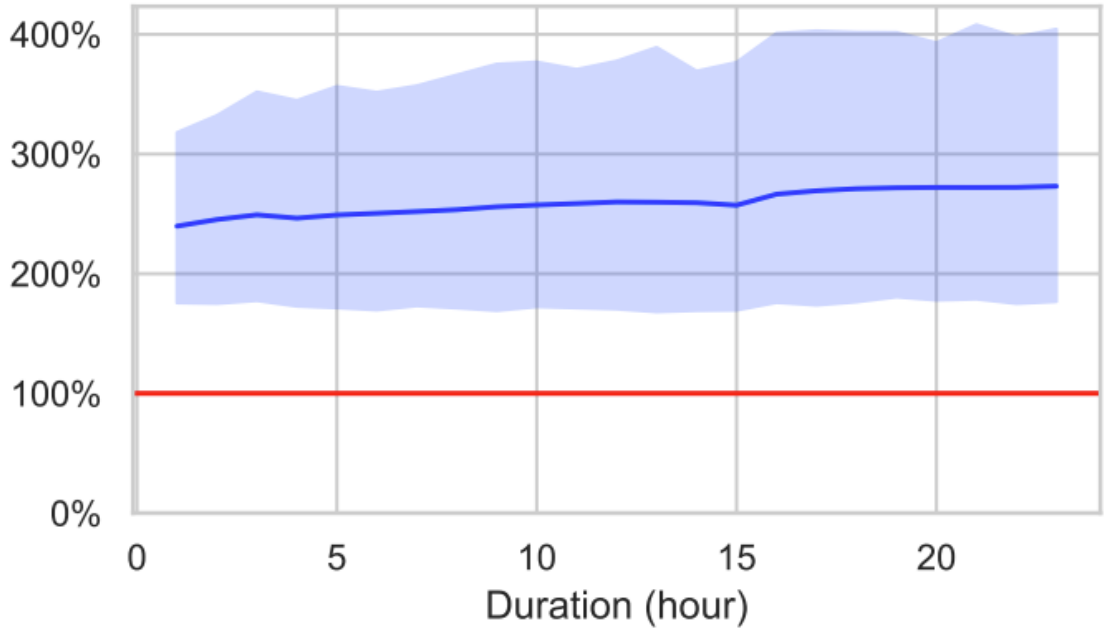


Figure 10: Temporal trend of normalized queue sizes using mean estimator with error band of 95% CI. The red horizontal line (100%) marks the queue size of libFuzzer.

Discussion

- Seed Replacement mechanism
  - Explore if this method can be extrapolated to enhance the performance of fuzzers utilizing advanced code coverage feedback.

- Data-Access Categories
  - Optimal trade-off between precision and efficiency
  - Open to exploring additional data collection techniques to enhance understanding of the application