Title	Context-Sensitive and Directional Concurrency Fuzzing for Data-Race Detection
Authors	Zu-Ming Jiang (Tsinghua University) <jjzuming@outlook.com> Jia-Ju Bai (Tsinghua University) <baijiaju@tsinghua.edu.cn> Kangjie Lu (University of Minnesota) <kjlu@umn.edu> Shi-Min Hu (Tsinghua University) <shimin@tsinghua.edu.cn>
Venue	The 29th Network and Distributed System Security Symposium (NDSS'22)
Date	February 27–March 3, 2022
Paper abstract	Fuzzing is popular for bug detection and vulnerability discovery nowadays. To adopt fuzzing for concurrency problems like data races, several recent concurrency fuzzing approaches consider concurrency information of program execution, and explore thread interleavings by affecting thread scheduling at runtime. However, these approaches are still limited in data-race detection. On the one hand, they fail to consider the execution contexts of thread interleavings, which can miss real data races in specific runtime contexts. On the other hand, they perform random thread-interleaving exploration, which frequently repeats already covered thread interleavings and misses many infrequent thread interleavings. In this paper, we develop a novel concurrency fuzzing framework named CONZZER, to effectively explore thread interleavings and detect hard-to-find data races. The core of CONZZER is a context-sensitive and directional concurrency fuzzing approach for thread-interleaving exploration, with two new techniques. First, to ensure context sensitivity, we propose a new concurrency-coverage metric, concurrent call pair, to describe thread interleavings with runtime calling contexts. Second, to directionally explore thread interleavings, we propose an adjacency-directed mutation to generate new possible thread interleavings with already covered thread interleavings and then use a breakpoint-control method to attempt to actually cover them at runtime. With these two techniques, this concurrency fuzzing approach can effectively cover infrequent thread interleavings with concrete context information, to help discover hard-to-find data races. We have evaluated CONZZER on 8 user-level applications and 4 kernel-level filesystems, and found 95 real data races. We identify 75 of these data races to be harmful and send them to related developers, and 44 have been confirmed. We also compare CONZZER to existing fuzzing tools, and CONZZER continuously explores more thread interleavings and finds many real data races missed by these tools.
DOI	10.14722/ndss.2022.24296

Context-Sensitive and Directional Concurrency Fuzzing for Data-Race Detection

Zu-Ming Jiang[1], Jia-Ju Bai[1], Kangjie Lu[2], Shi-Min Hu[1]
[1]Tsinghua University, [2]University of Minnesota

Tsinghua University

University of Minnesota
Driven to Discover℠

Research Problem

Existing fuzzing approaches are limited in detecting concurrency problems like data races. On one hand, they fail to consider the execution contexts of thread interleavings. On the other hand, they lack effective techniques to perform thread-interleaving exploration. In this paper, we propose a concurrency fuzzing framework named CONZZER which performs context-sensitive and directional thread-interleaving exploration for data-race detection.

Introduction

Data races can cause critical security problems like privilege escalation. To detect data races, many developers utilize fuzzing tools with data-race checkers to test concurrent programs. However, existing fuzzing approaches are limited in race detection. First, they use context-insensitive coverage metrics as fuzzing feedback and thus miss many deep data races that occur only in specific runtime contexts. Second, they perform random thread-interleaving exploration which is indicated to be inefficient.

To solve these limitations, we present CONZZER, a concurrency fuzzing framework that can explore infrequent thread interleavings and detect hard-to-find data races. CONZZER uses a new concurrency coverage metric, *concurrent call pair*, to describe thread interleavings with their runtime calling contexts as fuzzing feedback. Moreover, CONZZER incorporates *adjacency-directed mutation* and *deterministic breakpoint control* to perform directional thread-interleaving exploration.

Methodology

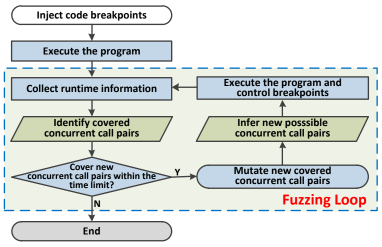
① Context-sensitive concurrency coverage

|--- Concurrent call pair

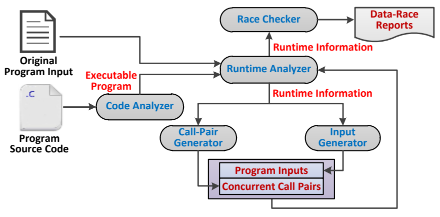
② Directional thread-interleaving exploration

|--- Adjacency-directed mutation

|--- Deterministic breakpoint control

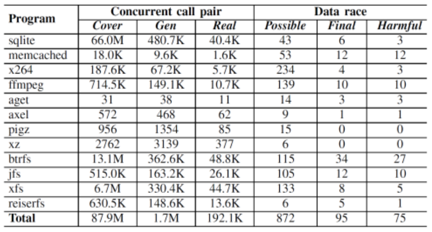

Inject code breakpoints
Execute the program
Collect runtime information
Execute the program and control breakpoints
Identify covered concurrent call pairs
Infer new posssible concurrent call pairs
Cover new concurrent call pairs within the time limit?
Y
Mutate new covered concurrent call pairs
N
Fuzzing Loop
End

Process of context-sensitive and directional fuzzing



CONZZER architecture

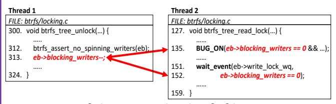
Evaluation



Program	Concurrent call pair			Data race		
	Cover	*Gen*	*Real*	*Possible*	*Final*	*Harmful*
sqlite	66.0M	480.7K	40.4K	43	6	3
memcached	18.0K	9.6K	1.6K	53	12	12
x264	187.6K	67.2K	5.7K	234	4	3
ffmpeg	714.5K	149.1K	10.7K	139	10	10
aget	31	38	11	14	3	3
axel	572	468	62	9	1	1
pigz	956	1354	85	15	0	0
xz	2762	3139	377	6	0	0
btrfs	13.1M	362.6K	48.8K	115	34	27
jfs	515.0K	163.2K	26.1K	105	12	10
xfs	6.7M	330.4K	44.7K	133	8	5
reiserfs	630.5K	148.6K	13.6K	6	5	1
Total	87.9M	1.7M	192.1K	872	95	75

Testing results

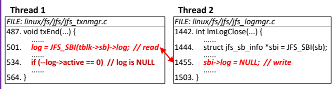
✓ CONZZER finds **95** (**75** harmful) data races in **12** programs.



Thread 1
```
FILE: btrfs/locking.c
300.  void btrfs_tree_unlock(...) {
          .....
312.      btrfs_assert_no_spinning_writers(eb);
313.      eb->blocking_writers--;
          .....
324.  }
```

Thread 2
```
FILE: btrfs/locking.c
127.  void btrfs_tree_read_lock(...) {
          ......
135.      BUG_ON(eb->blocking_writers == 0 && ...);
          ......
151.      wait_event(eb->write_lock_wq,
152.              eb->blocking_writers == 0);
          ......
159.  }
```

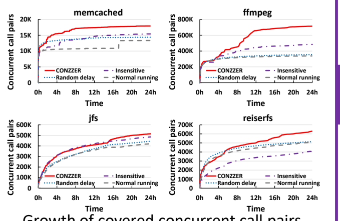
Harmful race in the *btrfs* filesystem



Thread 1
```
FILE: linux/fs/jfs/jfs_txnmgr.c
487. void txEnd(...) {
         ......
501.     log = JFS_SBI(tblk->sb)->log;  // read
         ......
534.     if (--log->active == 0)  // log is NULL
         ......
564. }
```

Thread 2
```
FILE: linux/fs/jfs/jfs_logmgr.c
1442. int lmLogClose(...) {
         ......
1444.    struct jfs_sb_info *sbi = JFS_SBI(sb);
         ......
1455.    sbi->log = NULL;  // write
         ......
1503. }
```

Harmful race in the *jfs* filesystem

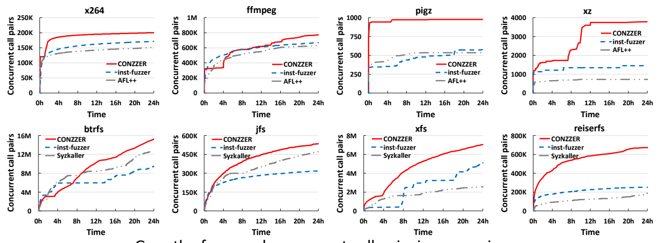


Growth of covered concurrent call pairs

✓ Many found data races cause severe security problems.

✓ CONZZER covers **19%** more thread interleavings than the method performing random-delay injection.

Comparison



Growth of covered concurrent call pairs in comparison

CONZZER vs. existing fuzzer (AFL++, Syzkaller, inst-pair-fuzzer):

✓ Additionally finds **13**, **58** and **32** races;

✓ Covers **88%**, **118%** and **58%** more thread interleavings.

Conclusion

- Existing fuzzers are limited in testing concurrent programs.
- We design a new concurrency fuzzing framework named CONZZER, which can effectively cover infrequent thread interleavings and detect hard-to-find data races.
- CONZZER finds many harmful data races in both user-level applications and kernel-level filesystems.