Fuzzing Error Handling Code using Context-Sensitive Software Fault Injection

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

[1]Tsinghua University, [2]University of Minnesota

Background

- Error handling code
  - Critical for handling various runtime errors
  - Error-prone
  - Hard to test

- Fuzzing
  - Widely used to detect bugs in various software
  - Existing fuzzers are mostly ***input-driven***
  - Cannot effectively test ***error handling code***

Background

- Software fault injection
  - Effectively test error handling code
  - Perform ***context-insensitive*** fault injection
  - Miss bugs that can be only triggered in specific context

```
int main() {
   x = malloc(...);
   y = malloc(...);
   ......
   FuncA(x);
   FuncB(y);
   ......
}
```

```
void FuncA(x) {
   FuncP(x);
   ......
}
void FuncB(y) {
   free(y);
   FuncP(y);
   ......
}
```

```
void FuncP(arg) {
   z = malloc(...)
   if (!z) {
      free(arg);
      exit(-1);
   }
   ......
}
```

Fault 1: main -> FuncA -> FuncP -> malloc **exit abnormally...**
Fault 2: main -> FuncB -> FuncP -> malloc **double free!**

Background

- Error that can trigger error handling code
  - Input-related error: *strcmp(), strlen(), memcmp()* …
  - Occasional error: *malloc(), open(), pipe()* …

Study

- Error handling code
  - 9 widely-used applications

Application	Studied file	Call site	Input-related	Occasional
vim	100	1163	530 (46%)	633 (54%)
bison	100	184	96 (52%)	88 (48%)
ffmpeg	100	881	518 (59%)	363 (41%)
clamav	100	1089	522 (48%)	567 (52%)
clfow	100	286	170 (59%)	116 (41%)
gif2png+libpng	95	830	556 (67%)	274 (33%)
openssl	100	989	571 (58%)	418 (42%)
Total	824	6,168	**3,570 (58%)**	**2,616 (42%)**

42% of the call sites that can trigger error handling code are related to occasional errors

Study

- CVEs found by existing fuzzers
  - 6 state-of-the-art fuzzers

Tool	CVE	Error handling	Occasional error
AFL	218	85	3
Honggfuzz	57	17	3

Existing fuzzers may miss bugs in error handling code especially those triggered by occasional errors

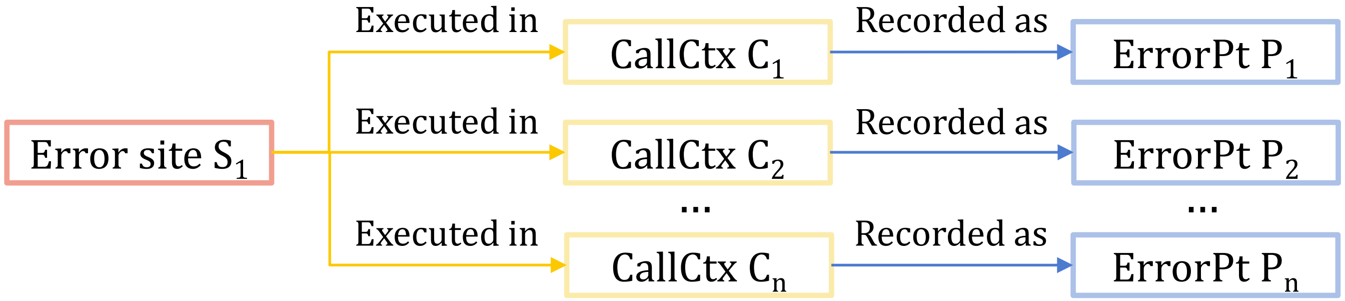
QSYM	6	0	0
REDQUEEN	11	2	1
Total	393	121 (31%)	**11 (9%)**

Basic Idea

- ***Error point***
  - Error site: call site that can fail and trigger error handling code
  - Composed of ***ErrLoc*** and ***CallCtx*** of each error site
    - ***ErrLoc:*** the location of each error site in source code
    - ***CallCtx:*** corresponding calling context when each error site is executed

$$ErrPt =< ErrLoc, CallCtx >$$

  - Perform context-sensitive fault injection



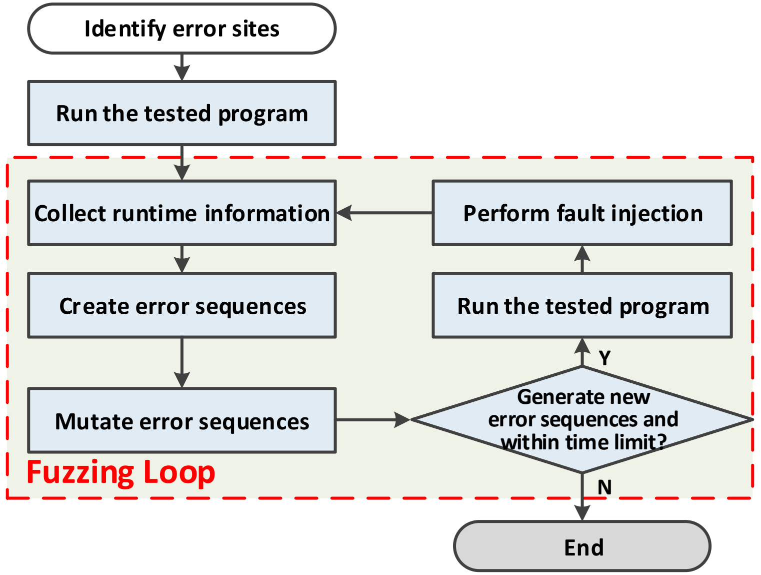
Basic Idea

- ***Error sequence***
  - Consist of multiple ordered ***Error points***
  - Describe the failure situation of the error points
    - 0 => Normally run
    - 1 => Fail by injecting faults
  - 0-1 sequence

$$ErrSeq = [ErrPt_1, ErrPt_2, ErrPt_3, \ldots ErrPt_x], ErrPt_i = \{0,1\}$$

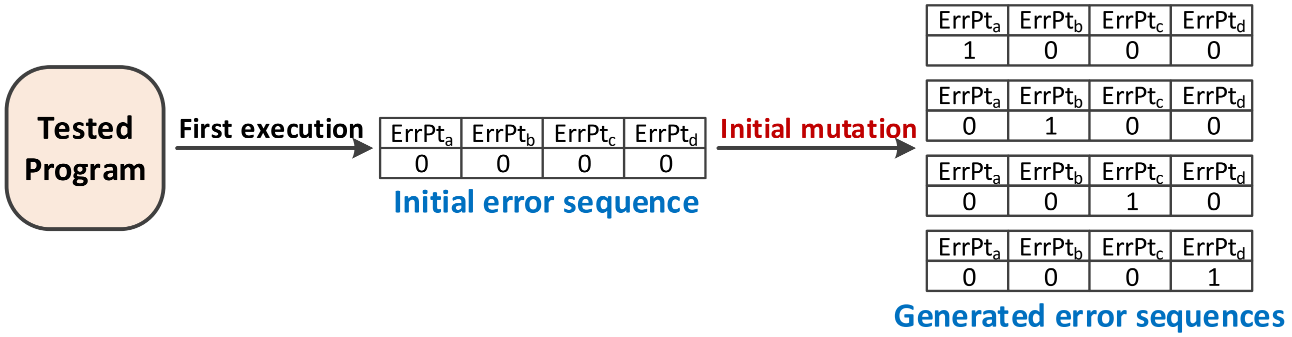
Context-Sensitive SFI-based Fuzzing

- Overview
  - Run the tested program
  - Collect runtime information
  - Create error sequences
  - **Mutate error sequences**
  - Run the tested program again and perform fault injection



Error Sequence Mutation

- Initial mutation
  - Collect the executed error points in runtime
  - The initial error sequence is an all-zero sequence
  - Make one executed error point fail (0 → 1)



Initial error sequence

$ErrPt_a$	$ErrPt_b$	$ErrPt_c$	$ErrPt_d$
0	0	0	0

Generated error sequences

$ErrPt_a$	$ErrPt_b$	$ErrPt_c$	$ErrPt_d$
1	0	0	0

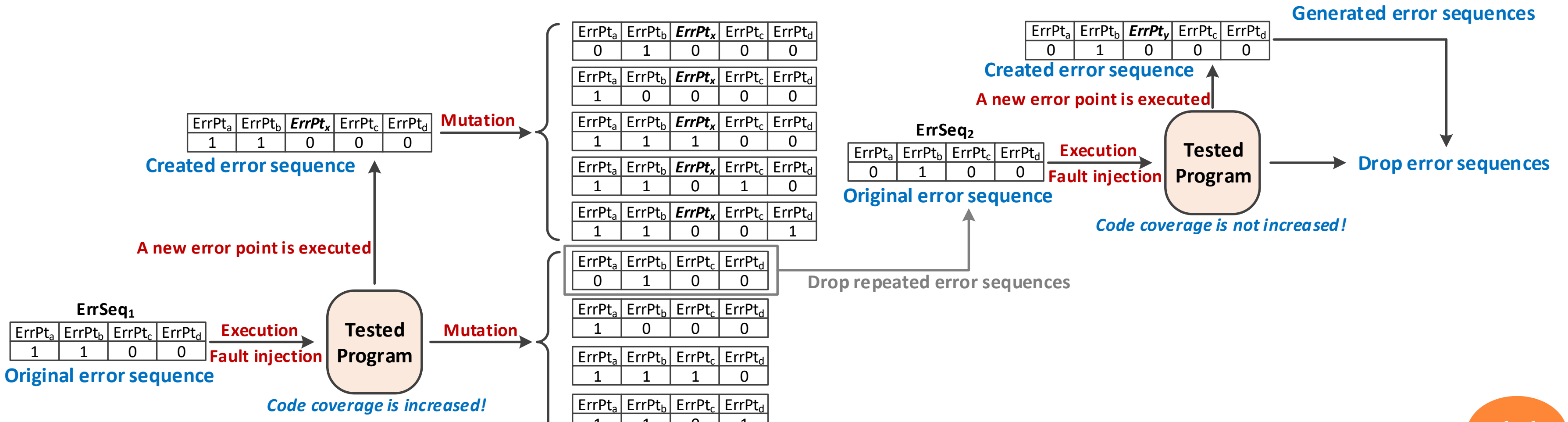
$ErrPt_a$	$ErrPt_b$	$ErrPt_c$	$ErrPt_d$
0	1	0	0

$ErrPt_a$	$ErrPt_b$	$ErrPt_c$	$ErrPt_d$
0	0	1	0

$ErrPt_a$	$ErrPt_b$	$ErrPt_c$	$ErrPt_d$
0	0	0	1

Error Sequence Mutation

- Subsequent mutation
  - Select error sequence which increases the code coverage
  - Change only one executed error point (0→1 or 1 → 0)
  - Drop repeated error sequences



ErrSeq$_1$

ErrPt$_a$	ErrPt$_b$	ErrPt$_c$	ErrPt$_d$
1	1	0	0

Created error sequence

ErrPt$_a$	ErrPt$_b$	***ErrPt$_x$***	ErrPt$_c$	ErrPt$_d$
1	1	0	0	0

ErrPt$_a$	ErrPt$_b$	***ErrPt$_x$***	ErrPt$_c$	ErrPt$_d$
0	1	0	0	0
1	0	0	0	0
1	1	1	0	0
1	1	0	1	0
1	1	0	0	1

ErrPt$_a$	ErrPt$_b$	ErrPt$_c$	ErrPt$_d$
0	1	0	0
1	0	0	0
1	1	1	0
1	1	0	1

ErrSeq$_2$

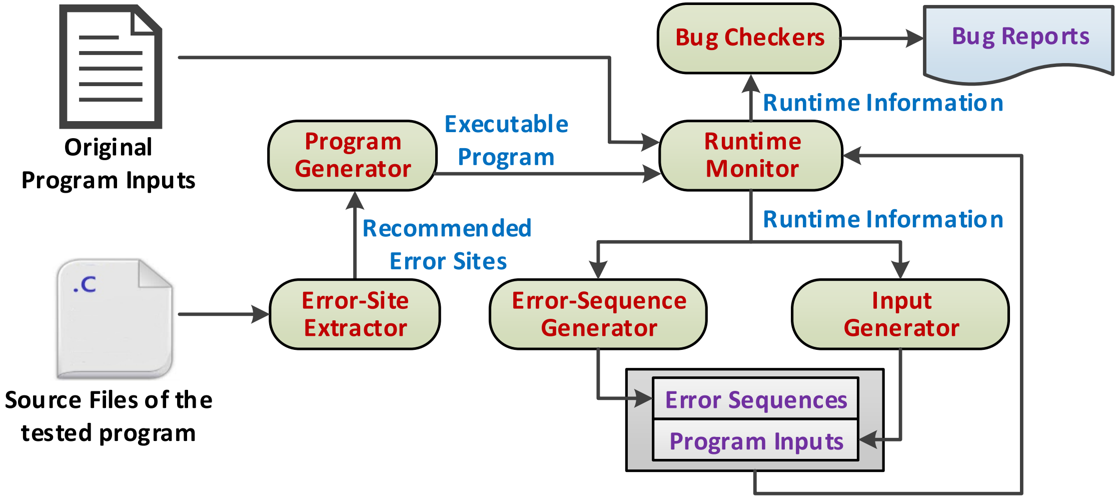
ErrPt$_a$	ErrPt$_b$	ErrPt$_c$	ErrPt$_d$
0	1	0	0

Created error sequence

ErrPt$_a$	ErrPt$_b$	***ErrPt$_y$***	ErrPt$_c$	ErrPt$_d$
0	1	0	0	0

Framework

- FIFUZZ
  - Input-driven fuzzer combined with context-sensitive SFI
  - Detect bugs hiding deeply in error handling code



Evaluation

- Experimental setup
  - 9 widely-used applications
  - 1822 extracted error sites
  - Time budget: 24 hours

Application	Description	Version	LOC
vim	Text editor	v8.1.1764	349K
bison	Parser generator	v3.4	82K
ffmpeg	Solution for media processing	n4.3-dev	1.1M
nasm	80x86 and x86-64 assembler	v2.14.02	94K
catdoc	MS-Word-file viewer	v0.95	4K
clamav	Antivirus engine	v0.101.2	844K
clfow	Code analyzer of C source files	v1.6	37K
gif2png+libpng	File converter for pictures	v2.5.14+v1.6.37	59K
openssl	Cryptography library	v1.1.1d	415K

Evaluation

- Bug detection
  - 50 new real bugs
  - 32 of them are confirmed

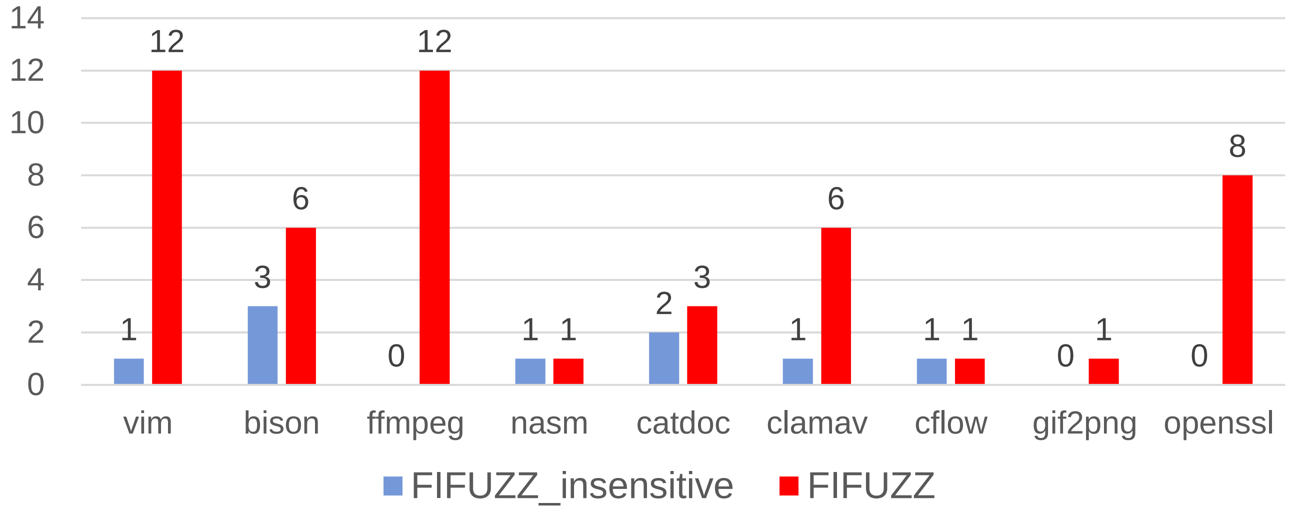
Bug type	**Crash/DOS**	**Memory corruption**	**Arbitrary read**	**Memory overread**
Null pointer dereference	36	0	0	0
Double free	0	5	0	0
Use after free	0	1	2	2
Buffer overflow	0	0	0	1
Free invalid pointer	2	0	0	0
Assertion failure	1	0	0	0
Total	39	6	2	3

Evaluation

- Bug features
  - 46 bugs are related to incorrect error handling
  - Error handling bugs are often triggered by 1 error
    - 42 bugs are triggered by 1 error
    - 4 bugs are triggered by 2 or more errors
  - Improper error handling in error propagation

Comparison

- Compared to context-insensitive SFI
  - Build error sequence using error site
    - Do not consider the calling context of each error site
  - Bug detection

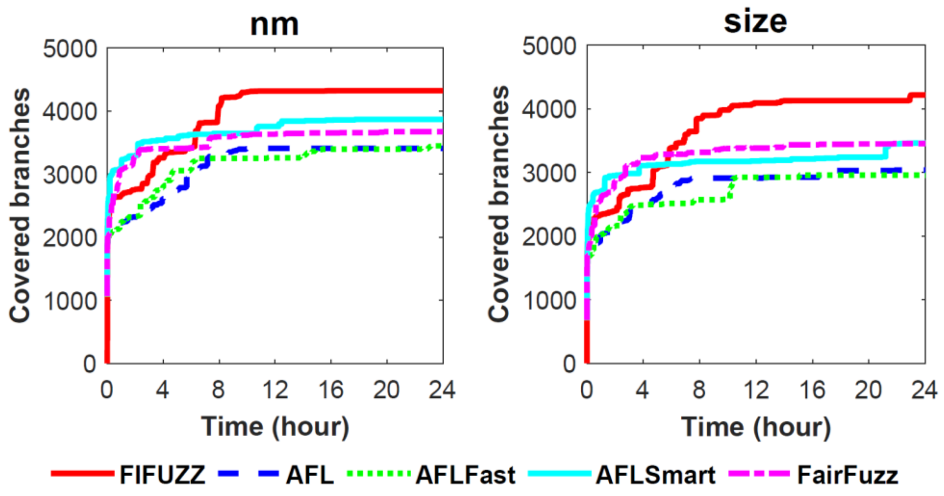


Comparison

- Compared to state-of-the-art fuzzers
  - 4 state-of-the-art fuzzers
    - AFL, AFLFast, AFLSmart and FairFuzz
  - 5 tested programs in Binutils 2.26
    - nm, objdump, size, ar and readelf

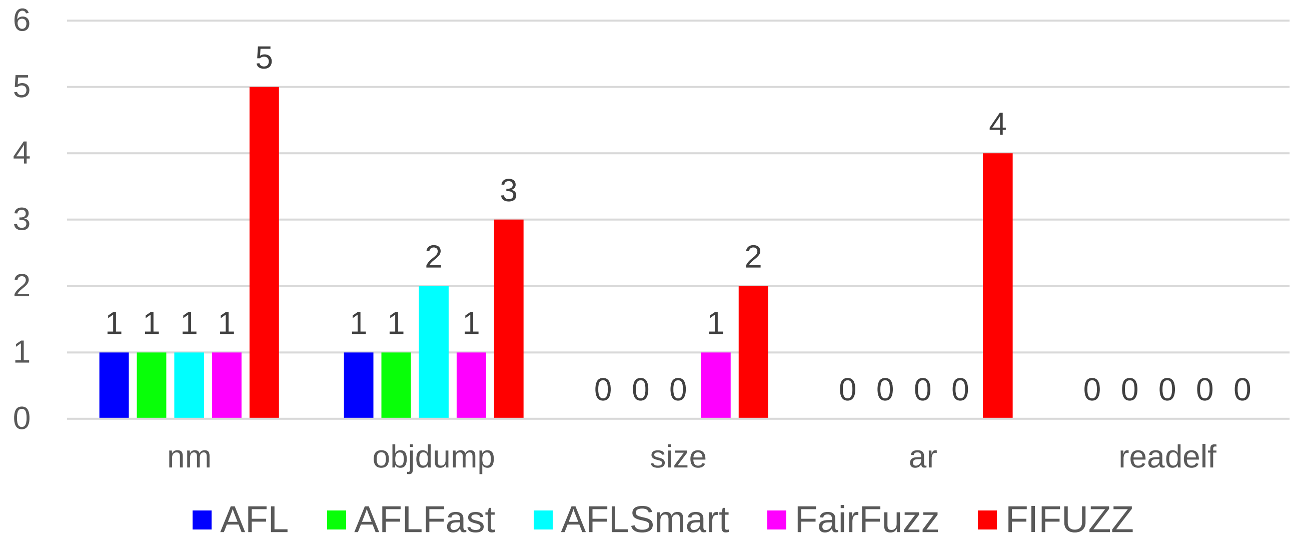
Comparison

- Compared to state-of-the-art fuzzers
  - Code coverage
    - Overall, FIFUZZ covers more branches than other fuzzers
    - FIFUZZ can cover much more error handling code



Comparison

- Compared to state-of-the-art fuzzers
  - Bug detection
    - Overall, FIFUZZ finds more bugs than other fuzzers
    - FIFUZZ can find bugs hiding deeply in error handling code



	AFL	AFLFast	AFLSmart	FairFuzz	FIFUZZ
nm	1	1	1	1	5
objdump	1	1	2	1	3
size	0	0	0	1	2
ar	0	0	0	0	4
readelf	0	0	0	0	0

Conclusion

- Existing fuzzers cannot cover error handling code effectively
- FIFUZZ
  - Propose a novel context-sensitive SFI-based fuzzing strategy
  - Design a promising fuzzing framework
- Find 50 real bugs in 9 widely-used C applications
- Outperform existing fuzzers

Thanks

Zu-Ming Jiang	*jjzuming@outlook.com*
Jia-Ju Bai	*baijiaju1990@gmail.com*
Kangjie Lu	*kjlu@umn.edu*
Shi-Min Hu	*shimin@tsinghua.edu.cn*