Improving Fuzzing Performance by Applying Interval Mutations

Hovhannes Movsisyan

Sevak Sargsyan

Jivan Hakobyan

Matevos Mehrabyan

Vahagn Sirunyan

Shamil Kurmangaleev

System Programming Laboratory, YSU, ISP RAS

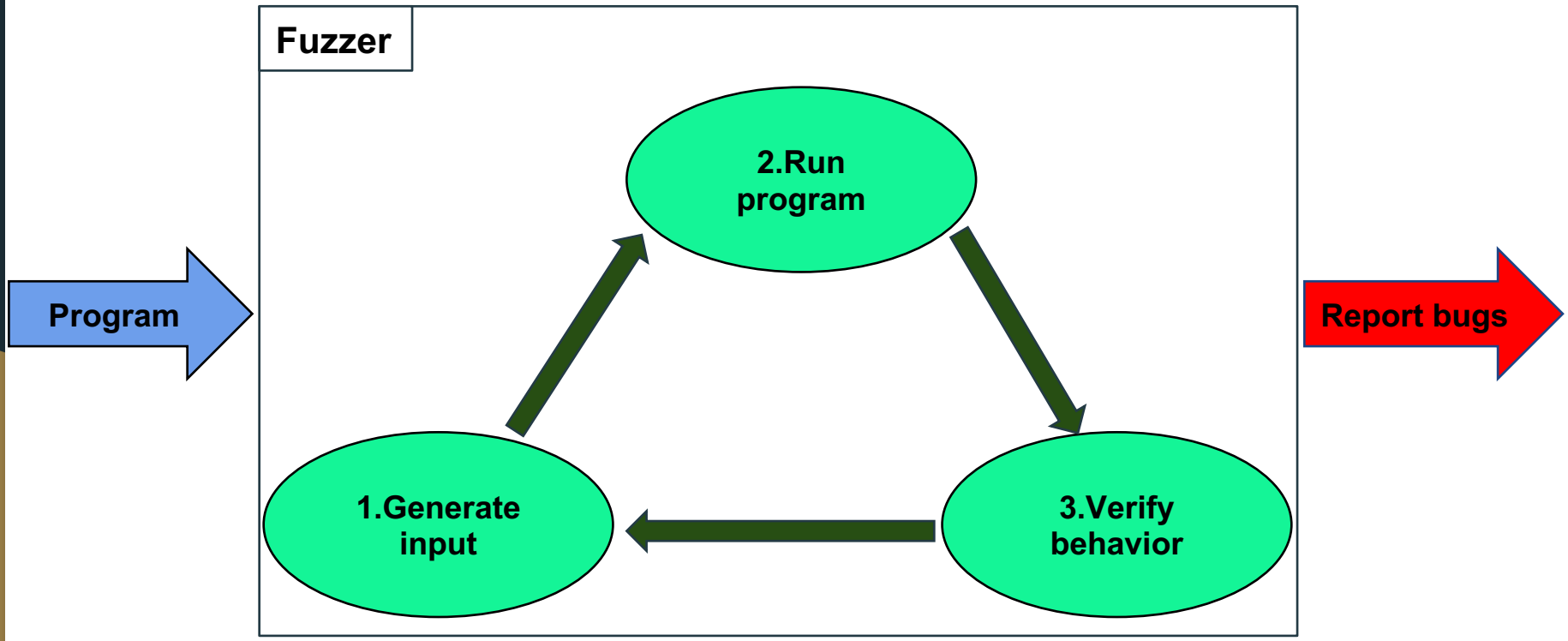
2019

Introduction

- In the modern world the development of reliable software is still an essential aspect in the field of information technologies

- **Fuzzing** is one of the most popular and efficient methods of dynamic analysis

Process of Fuzzing

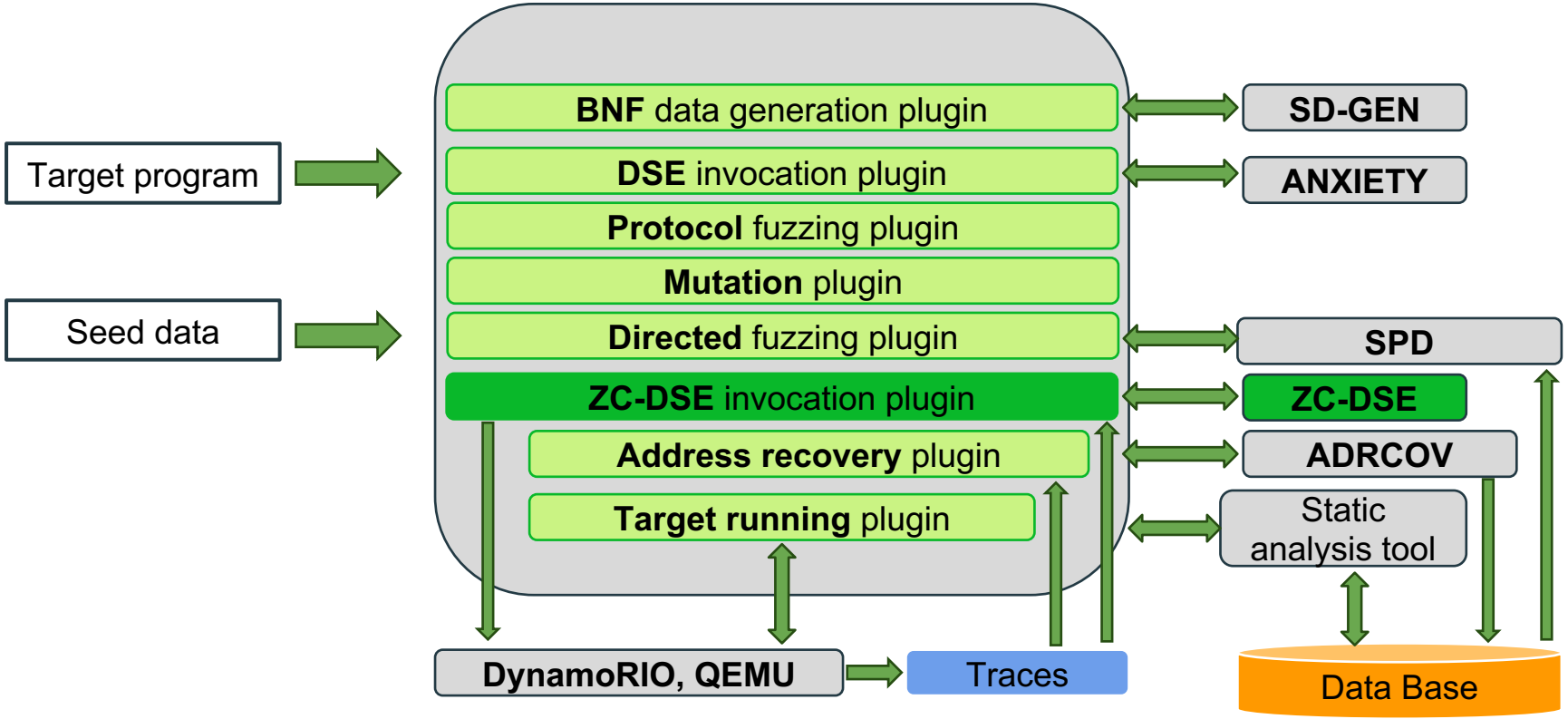


Fuzzing Tools

ISP RAS

Fuzzing tool	Extendable	Uses input structure information
AFL	No	No
WinAFL	No	No
kAFL	No	No
Syzkaller	No	No
ISP-FUZZER	Yes	No

ISP-FUZZER


Target program
Seed data
BNF data generation plugin
DSE invocation plugin
Protocol fuzzing plugin
Mutation plugin
Directed fuzzing plugin
ZC-DSE invocation plugin
Address recovery plugin
Target running plugin
SD-GEN
ANXIETY
SPD
ZC-DSE
ADRCOV
Static analysis tool
DynamoRIO, QEMU
Traces
Data Base

Header Information and Magic Values

```
bool is_linux_binary(char* binary) {
  if (binary[0] == 0x7F && strncmp("ELF", binary + 1, 4) == 0)
{
    return true;
  } else {
    return false;
  }
}
```

ELF - **'0x7f E L F'**

Random mutations may change header information or magic values of the input data

Goal of The Work

ISP RAS

- *Find intervals of input data that influence on execution of a particular basic block*

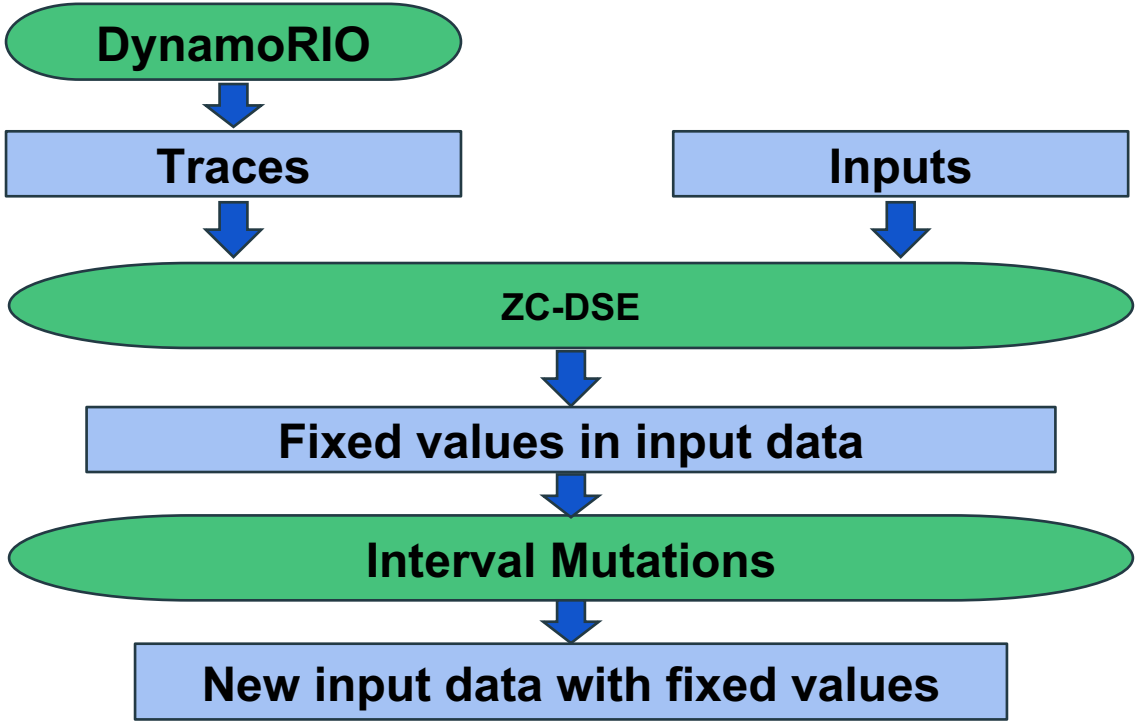
- *Mutate parts of the input data that do not influence on the execution of a particular basic block*

Implementation

Design and develop plugins for ISP-FUZZER

- Trace generation
- ZC-DSE (Zero Cost DSE)
- Interval mutations

Plugins Workflow



Trace Generation

DynamoRIO - runtime code manipulation system that supports code transformations on any part of a program during its execution

```
bool is_linux_binary(char* binary) {
  if (binary[0] == 0x7F &&
strncmp("ELF", binary + 1, 3) == 0) {
    return true;
  } else {
    return false;
  }
}
```

```
0000000000400a16 <_Z15is_linux_binaryPc>:
 400a16:      55                   push   %rbp
 400a17:      48 89 e5             mov    %rsp,%rbp
 400a1a:      48 83 ec 10          sub    $0x10,%rsp
 400a1e:      48 89 7d f8          mov    %rdi,-0x8(%rbp)
 400a22:      48 8b 45 f8          mov    -0x8(%rbp),%rax
 400a26:      0f b6 00             movzbl (%rax),%eax
 400a29:      3c 7f                cmp    $0x7f,%al
 400a2b:      75 25                jne    400a52
<_Z15is_linux_binaryPc+0x3c>
 400a2d:      48 8b 45 f8          mov    -0x8(%rbp),%rax
 400a31:      48 83 c0 01          add    $0x1,%rax
 400a35:      ba 03 00 00 00       mov    $0x3,%edx
 400a3a:      48 89 c6             mov    %rax,%rsi
 400a3d:      bf f4 0b 40 00       mov    $0x400bf4,%edi
 400a42:      e8 29 fe ff ff       callq  400870
<strncmp@plt>
 400a47:      85 c0                test   %eax,%eax
 400a49:      75 07                jne    400a52
<_Z15is_linux_binaryPc+0x3c>
 400a4b:      b8 01 00 00 00       mov    $0x1,%eax
 400a50:      eb 05                jmp    400a57
<_Z15is_linux_binaryPc+0x41>
 400a52:      b8 00 00 00 00       mov    $0x0,%eax
 400a57:      c9                   leaveq
 400a58:      c3                   retq
```

```
"trace" : [
……
{
 "parent_start":2582,
 "parent_end":2603,
 "child_start": 2642,
 "child_end": 2648
},
 ……
]
```

ZC-DSE

ZC-DSE (Zero Cost Dynamic Symbolic Execution) is developed as a plugin to find intervals of input data that influence on the execution of a particular basic block of the target binary.

1. Selecting ***“interesting”*** basic blocks
2. Binding ***“interesting”*** basic blocks to intervals of the input values

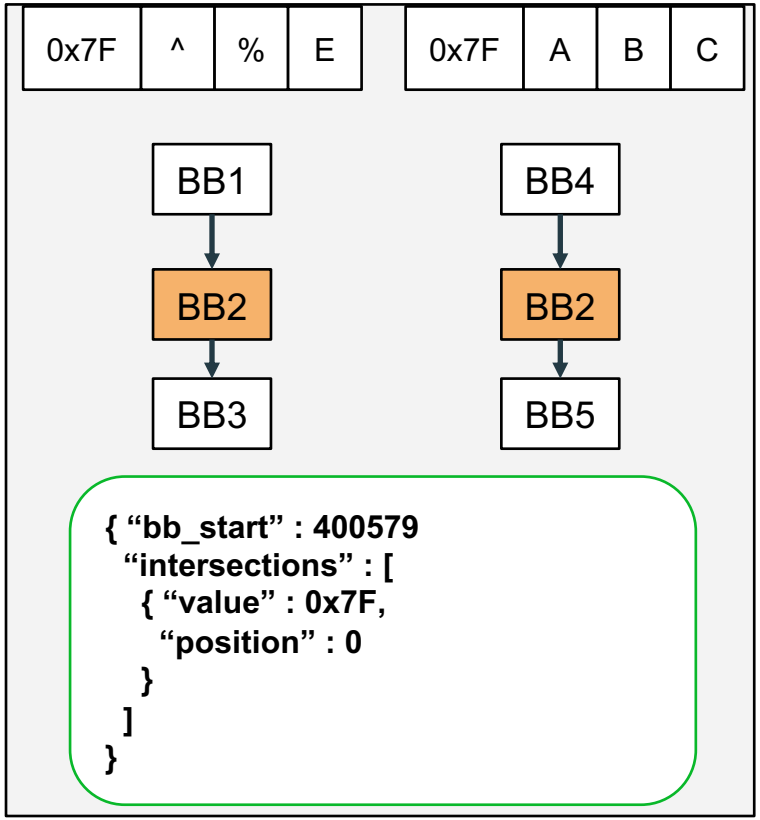
Interesting Basic Blocks

- Most common: at least ***P*** percent of all traces contain basic block
- Custom: Implemented and integrated by user

Basic Blocks to Intervals

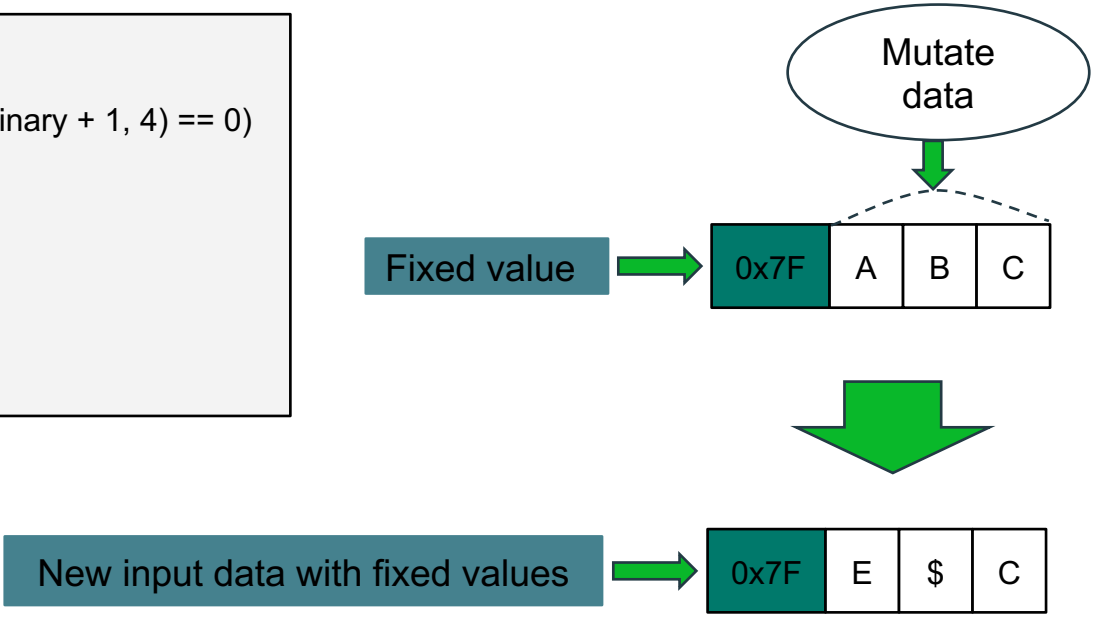
```
bool is_linux_binary(char* binary) {
  if (binary[0] == 0x7F && strncmp("ELF", binary + 1, 4) == 0)
{
    return true;
  } else {
    return false;
  }
}
```

***Note**: If intersection of corresponding input data was empty then we stop processing of this basic block. Empty result of intersection proves that there is no fixed fragment of input data influencing this basic block execution.*



Interval Mutations

```
bool is_linux_binary(char* binary) {
  if (binary[0] == 0x7F && strncmp("ELF", binary + 1, 4) == 0)
{
    return true;
  } else {
    return false;
  }
}
```



Interval Mutations Strategy

Previous ***X*** executions were not able to detect at least ***Y*** new execution paths

Results (1/2)

ISP RAS

Program name	ISP-FUZZER		ISP-FUZZER + ZC-DSE		Difference	
	Found paths	Found faults	Found paths	Found faults	Paths	Faults
readelf	1175	0	2081	1	+906	+1
djpeg	48	0	45	1	-3	+1
objdump	391	0	407	0	+16	0
ar	11	0	9	0	-2	0
latex	80	0	196	1	+116	+1
optipng	465	34	504	41	+39	+7
gif2png	309	10	366	37	+57	+27
tiff2ps	166	0	187	0	+21	0

Results (2/2)

ISP RAS

Program name	ISP-FUZZER		ISP-FUZZER + ZC-DSE		Difference	
	Found paths	Found faults	Found paths	Found faults	Paths	Faults
tiff2bw	54	0	63	0	+9	0
tiff2pdf	88	0	109	0	+21	0
tiff2rgba	37	0	44	0	+7	0
jasper	4	0	4	0	0	0
zipclock	91	0	108	1	+17	+1
dvi2tty	391	0	459	0	+68	0
strings	7	0	71	0	+64	0
pdftk	9	0	11	0	+2	0

ISP RAS

Questions?

ISP RAS

Thank You for Your Attention!