lldb-eval fuzzer: Finding bugs in an LLDB based expression evaluator

LLVM Developers Meeting 2021

Speaker: Tonko Sabolčec

Started as Alexandros' (github.com/octurion) intern project

What is lldb-eval?

Name	Value	Type
map	Map of 5 elements.	HashMap
[0]	{ "apple", 4 }	KeyValuePair
[1]	{ "banana", 2 }	KeyValuePair
[2]	{ "orange", 5 }	KeyValuePair
[3]	{ "cherry", 6 }	KeyValuePair
[4]	{ "tomato", 1 }	KeyValuePair
rectangle	<Rectangle> (5, 5) - (10, 20)	Rectangle
a	5	int
b	15	int
area	75	int
[Raw View]	{x1=5 y1=5 x2=10 ...}	Rectangle
x1	5	int
y1	5	int
x2	10	int
y2	20	int
rectangle.x2 - rectangle.x1	5	int
Add item to watch		

Watch window in Visual Studio

Expected to evaluate **hundreds** of custom expressions in a single step

LLDB API

- **`SBFrame::EvaluateExpression`**
- **`SBValue::EvaluateExpression`**
- Supports all kinds of expressions
- Too slow for IDE integration

lldb-eval

- Interpreter for C++ expressions implemented from scratch
- Uses LLDB API
- Supports limited set of expressions
- https://github.com/google/lldb-eval

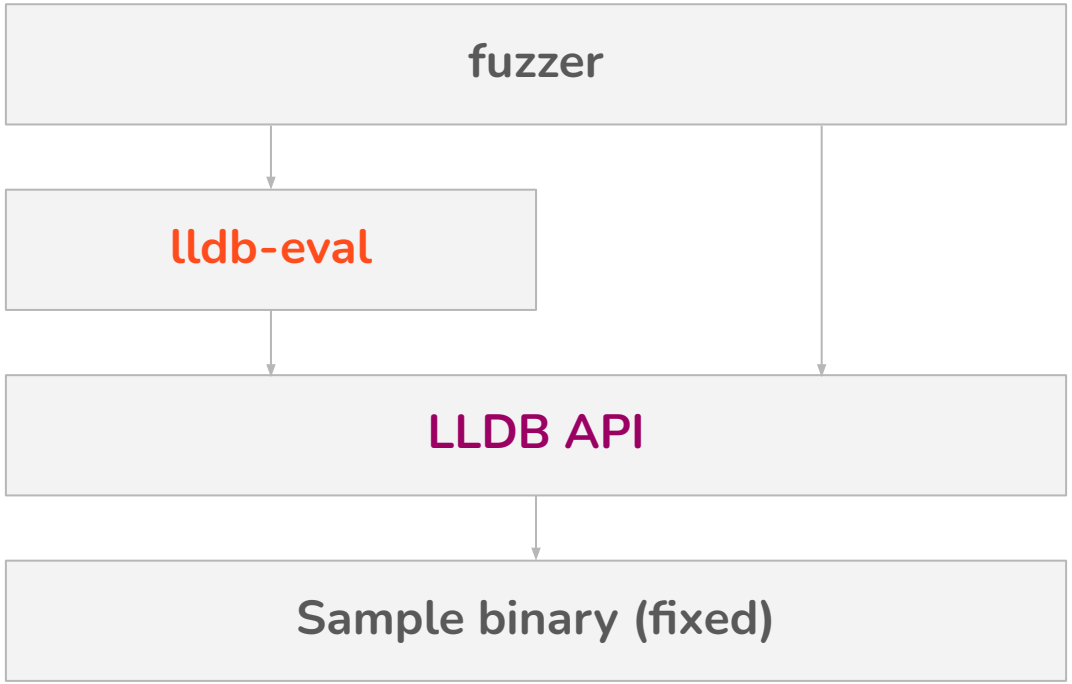
Testing lldb-eval

- Ensuring lldb-eval's success
- Unit testing?
  - OK for testing single concepts
  - Testing combinations of different concepts?
  - **Combinatorial explosion!** 💣
- Fuzz testing
  - Randomly generated tests

```
while (true) {
  expr = generate_random_expr()
  res1 = LLDB_evaluate(expr)
  res2 = lldb_eval_evaluate(expr)
  if (res1 != res2)
    notify_user()
}
```

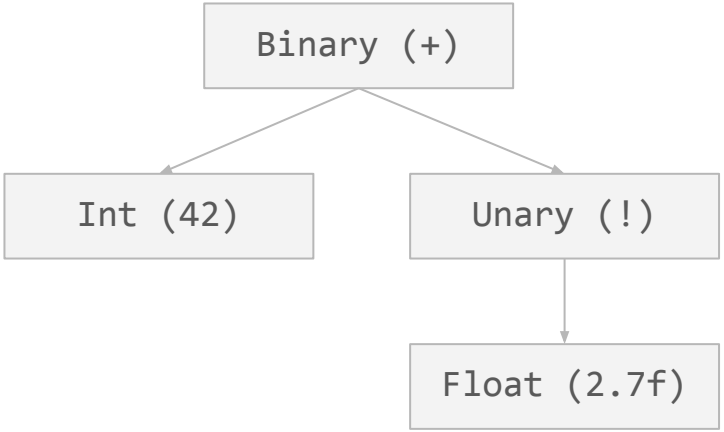
lldb-eval fuzzer in a nutshell

High-level overview



Expression generation

- Constructs abstract syntax tree
- Top down construction
- Depth limit to avoid infinite recursion
- Uses **constraints** to regulate what kind of (sub)expressions are acceptable



AST for **`42 + !2.7f`**

Constraints

- What types of expression can be generated at specific point?

Type constraints

- Are scalar types (int, float, char, etc.) allowed?
- Are pointer or array types allowed? Of what size?
- Other: classes, structs, enums

Memory constraints

- Is it OK to generate an invalid pointer?
- If not, how many times is the subexpression going to be dereferenced?

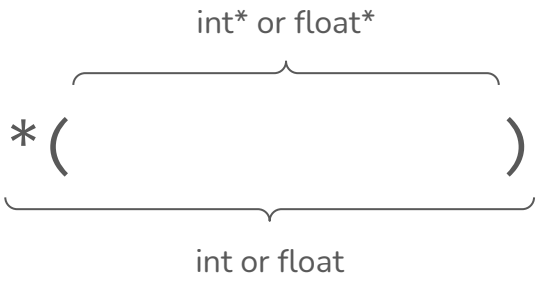
L-value constraint

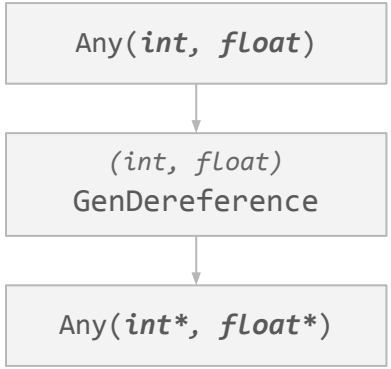
Type constraints (example)

```
Any(int, float)
```

int or float

Type constraints (example)





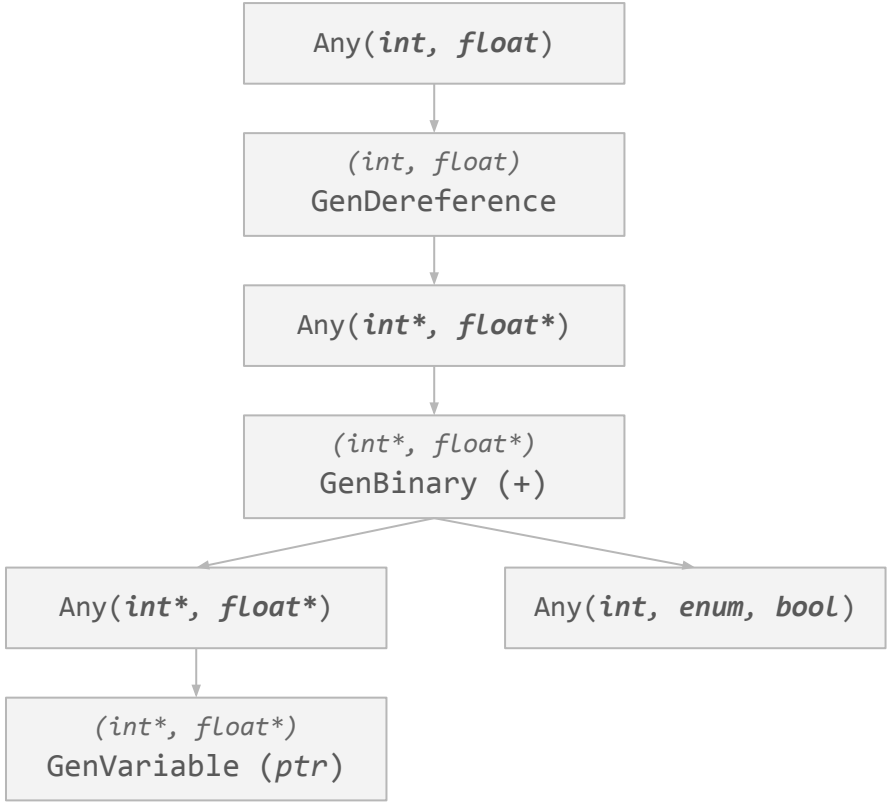
Type constraints (example)

int* or float*

int context

```
*(ptr + offset)
```

int or float



Challenges

- Not all mismatches are bugs in lldb-eval

Undefined behaviour (UB)

- **False reports** (produce unwanted noise)

```
> 1 / 0
LLDB      : 58926239
lldb-eval : 0
```

```
> 0 + *(bool*)int_ptr
LLDB      : 2
lldb-eval : 1
```

- More strict constraints
- UB detection in lldb-eval

Results (lldb-eval bug)

```
int var = 1;
int value = 2;  // global variable
```

`var < 0 > ::value`

LLDB: false (OK)

lldb-eval: undeclared identifier ‘`var<0>::value`’

Different context,
different interpretation

```
template <int N>
struct var {
  static int value = N;
};
```

Results (LLDB bug)

```
enum Enum : unsigned char { A, B }
```

```
-(Enum::B << 1)
```

LLDB: 4294967294

lldb-eval: -2 (OK)

libFuzzer integration

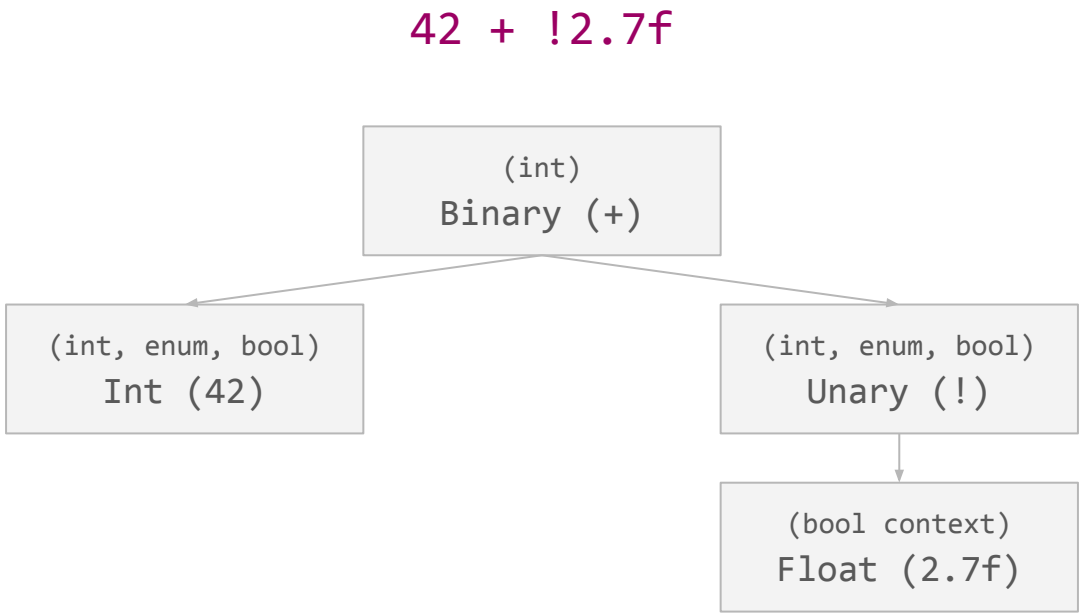
- libFuzzer is a **coverage-guided**, **mutation-based** fuzzing engine

Expressions are ranked by code coverage triggered in lldb-eval

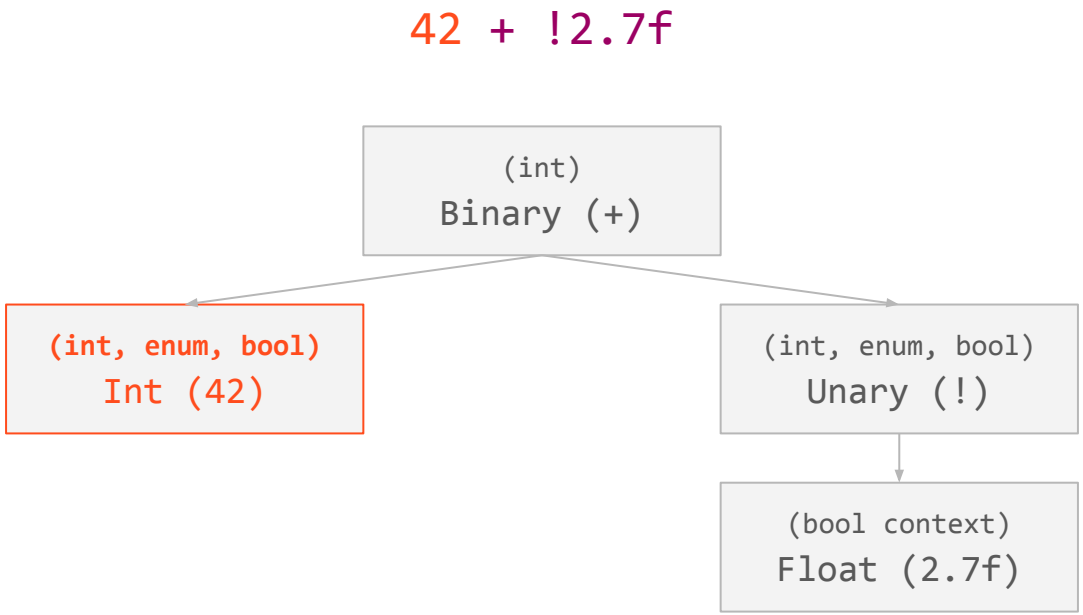
New expression is generated by applying mutation on existing one

- Faster progress
- Easy integration with continuous fuzzing services (OSS-Fuzz)

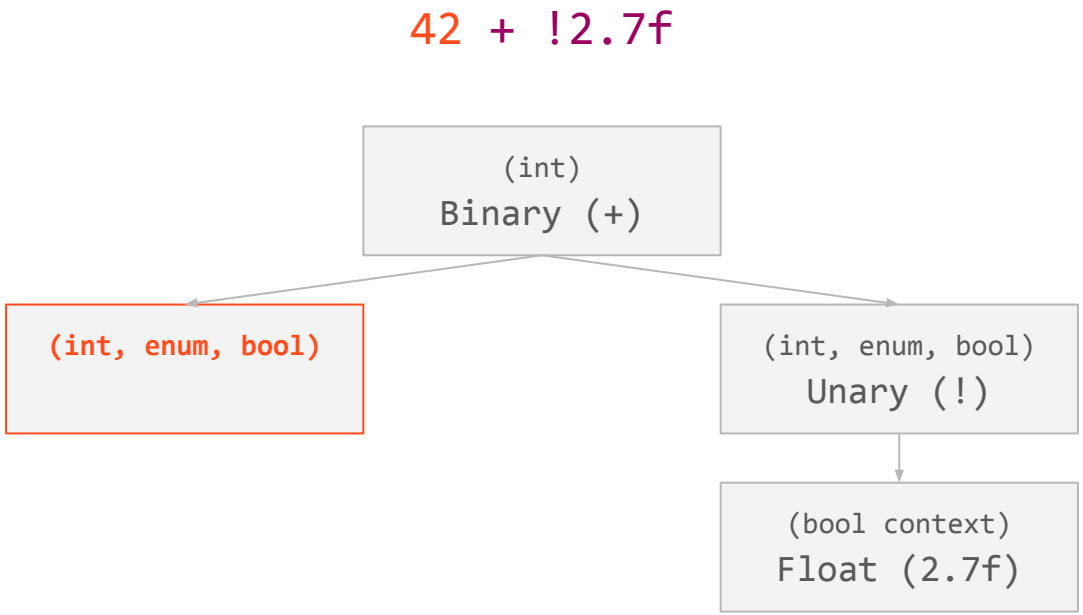
libFuzzer integration (mutation)



libFuzzer integration (mutation)

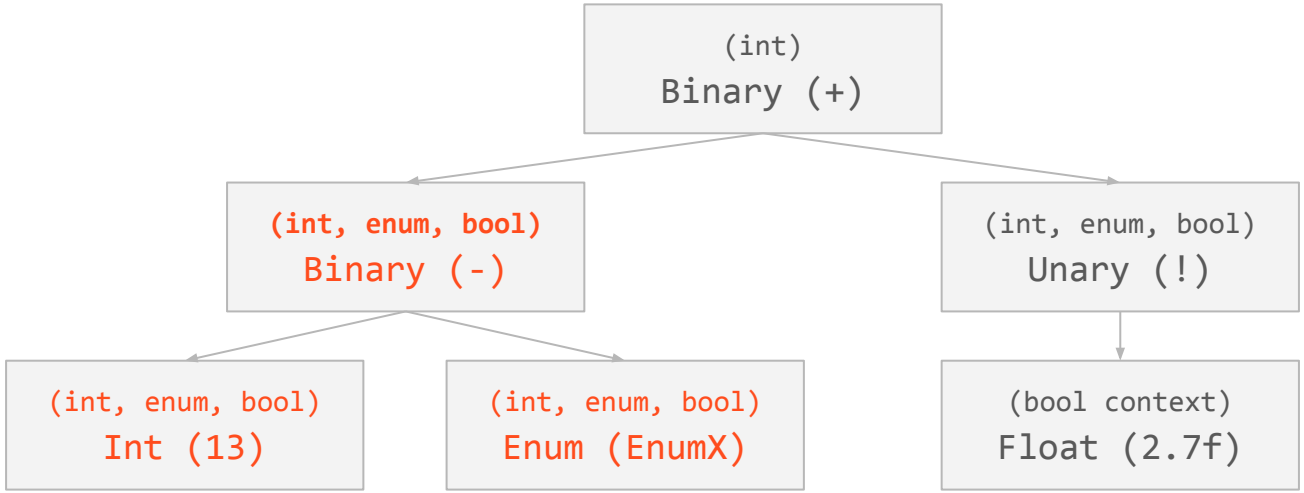


libFuzzer integration (mutation)



libFuzzer integration (mutation)

```
13 - EnumX + !2.7f
```



Conclusions

- Fuzz testing discovered dozens of bugs in lldb-eval
  - https://github.com/google/lldb-eval/blob/master/docs/bugs.md
- It is also beneficial to the LLDB project
- 5-10 millions of expressions per day with OSS-Fuzz

Thank you!