Fuzzing the Solidity Compiler

Bhargava Shastry
Ethereum Foundation

@bshastry
@ibags
bshastry
whoami

- Security engineer, Solidity team
- Semantic testing of Solidity compiler

Find security-critical bugs in the compiler before it is shipped
tl;dr:

- Threat model: Incorrect code generation
- Randomly generated valid Solidity (yul) programs test compiler
- Found 9 bugs using semantic fuzzing
- Continuous fuzzing for early bug discovery
Introduction
Threat model

- Compiler user (developer) is not malicious
- Bugs introduced by the optimizer
Fuzz testing in a nutshell

while not ctrl + c

do

    input=gen_input()

    runProgram(input)

done
Limitation of random fuzzing

contract C {
    function foo() public {
        do_something();
    }
}

contract C {
    function foo() public {
        do_something();
    }
}

Accepted by parser

contract C {
    function foo() public {
        do_something();
    }
}

contract C {
    function foo() public {
        do_something();
    }
}

Rejected by parser

Mutation
Fuzzing a compiler requires generating valid programs...

... generating a valid program requires structure awareness
Approach
Write a specification

Specification written in protobuf language

```protobuf
message Block {
    repeated Statement stmts;
}
...
message program {
    repeated Block blocks;
}
```

Full spec: https://github.com/ethereum/solidity/blob/develop/test/tools/ossfuzz/yulProto.proto
Input generation

- Input generated and mutated by libprotobuf-mutator
- Each input is a tree

```plaintext
blocks { stmts { ifstmt { condition {
  binaryOp { eq { op1: varref{id: 0} op2: 0}
}}}}}}}}}
```
Input conversion

- Converter is source-to-source translator
- Input: protobuf serialization format
- Output: yul program
Example

blocks { stmts { ifstmt { condition {
  binaryOp { eq { op1: varref{id: 0} op2: 0}
  }
}
}

if $x_0 == 0$
Test program generation

**Protobuf specification**

```
Message func {
  Block b = 1;
}
```

**Libprotobuf + mutator**

```
function f()
{
...
}
```

**Protobuf Converter**

```
{
  function f()
  {
    ...
  }
}
```

**Test program**
Correctness testing requires encoding expectation somehow
Differential fuzzing

- Track side-effects of execution
- Run program
- Run optimized program
- Compare side-effects
Yul interpreter

- Interprets arbitrary yul program
- Outputs side-effects as a trace (string)
Yul interpreter

```solidity
{ 
  function f() 
  { 
    ... 
  }
}
```

Test program

Interpreter

Execution trace

- `MLOAD`
- `MSTORE`
- `DATACOPY`
Fuzzing Setup

Program generator

Optimizer

Interpreter

MLOAD
MSTORE
...
DATACOPY

Trace
Custom Fuzz Mutator

Program Generator → Custom Mutator

if x_0 == 0

if x_0 != 0
Results
Bugs by component

- Optimizer Rule: 22.2% (2 bugs)
- Yul optimizer: 77.8% (7 bugs)
Bugs by impact

- Production: 44.4%
- Experimental: 55.6%
Bugs by severity

- **NA**: 55.6%
- **Medium**: 11.1%
- **Low**: 22.2%
- **Very low**: 11.1%

Found via custom mutation
Conclusion
Conclusion

● Continuous structure-aware fuzzing for early bug discovery
● Useful for testing optimizer and data en/decoding
● Decent assurance
  ○ Evidence that it works
  ○ No formal guarantees though
Thank you!

Solidity talks @ EthCC3

Mathias and Eric:
What’s New in Solidity,
Day 1, Monge, 15:20

Chris: Metadata and
Source Code Repository,
Day 2, Monge, 15:55