Fuzzing the Solidity Compiler

Bhargava Shastry
Ethereum Foundation

@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 2 security relevant bugs in EVM optimizer
  ○ Low or very low security impact
● Found 5 other bugs in experimental optimizer
● Continuous fuzzing for early bug discovery
Introduction
Threat model

- Compiler user (developer) is not malicious
- Bugs introduced by the optimizer

```solidity
function foo() -> x {
    x := 2
}
mstore(0, foo())
```
Fuzz testing in a nutshell

```python
while not ctrl + c
    do
        input=gen_input()
        runProgram(input)
    done
```
Limitation of random fuzzing

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

Mutation

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

Accepted by parser

Rejected by parser
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

```
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

Libprotobuf + mutator

Protobuf Converter

Test program

Message func {
    Block b = 1;
}

{ function f() {
    ... 
  }
}
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

{  
  function f()  
  {        ...  
  }  
}

Test program

Interpreter

Execution trace

MLOAD
MSTORE
...  
DATACOPY
Fuzzing Setup

Program generator → Interpreter

Optimizer

MLOAD
MSTORE
...
DATACOPY

MLOAD
MSTORE
...
DATACOPY

Trace
Results
Bugs by component

- Optimizer Rule: 28.6%
- Yul optimizer: 71.4%
Bugs by impact

- Production: 28.6% (2 bugs)
- Experimental: 71.4% (5 bugs)
Bugs by severity

- Low: 1 bug (14.3%)
- Very low: 1 bug (14.3%)
- NA: 5 bugs (71.4%)

Devcon5 | Fuzzing the Solidity Compiler
Challenges

● Find high-severity bugs using fuzz testing
  ○ Slow test throughput (~1 test per second)
● Test Abiv2encoder
  ■ Generate test program (Reasonably fast)
  ■ Compile program (Slowest)
  ■ Run program on EVM (Slow)
  ■ Assert output validity (Very fast)
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!

https://github.com/ethereum/solidity