A Comparative Analysis of Smart Contract Fuzzers' Effectiveness

@agfviggiano

Antonio Viggiano

DeFi Security Summit 2023

Introduction

Who I am

- Independent Security Researcher
- yAcademy ZK Auditing Fellow
- Building a scalable fuzz testing platform at fuzzy.fyi
- Previously
  - Co-founded many web2/web3 companies
  - Developed a distributed columnar database in C
  - Contributed to Redis

Presentation agenda

1. Introduction
2. How fuzzers work
3. How to measure fuzzers' effectiveness
4. Methodology
5. Results
6. Conclusion
7. Future research

Goal

Comparing different fuzzers' effectiveness

How fuzzers work

```
corpus ← initSeedCorpus()
queue ← ∅
observations ← ∅
while ¬isDone(observations,queue) do
      candidate ← choose(queue, observations)
      mutated ← mutate(candidate,observations)
      observation ← eval(mutated)
      if isInteresting(observation,observations) then
            queue ← queue ∪ mutated
            observations ← observations ∪ observation
      end if
end while
```

How to measure fuzzers' effectiveness

Challenges

1. Fundamentally random nature of fuzzing
2. Performance can vary substantially from run to run
3. Performance can vary over the course of a run
4. Different ways of finding bugs
    a. Introducing synthetic bugs
    b. Comparing different revisions of the same code
5. Different measures of effectiveness
    a. Code coverage
    b. Number of crashing inputs
    c. Number of unique bugs found
        i. Coverage profile
        ii. Stack hashes
        iii. Manual counting

How to measure fuzzers' effectiveness

Challenges

1. Fundamentally random nature of fuzzing
2. Performance can vary substantially from run to run
3. Performance can vary over the course of a run
4. Different ways of finding bugs
5. Different measures of effectiveness

Solutions

1. Compare the median and dispersion over many runs
2. Use different seeds, Use different test environments
3. Fuzz over long hours, Plot performance over time
4. Measure unique bugs by counting each single conceptual bugfix
5. Directly measure the number of bugs found, Use a statistical test to compare performance

Methodology

Comparing different fuzzers

- Choose algorithms to compare
  - echidna `2.2.0` (slither `0.9.3`)
  - foundry `588ad27` (forge `0.2.0`)
- Choose a representative set of target programs to test
  - Uniswap v2
- Choose how to measure performance
  - Time to find 12 injected bugs (gambit, manual)
- Fill in algorithm parameters
  - 30 different seeds
  - 24h timeout or $2^{32} - 1$ runs
  - m3.medium EC2 instance (1 vCPU)
- Carry out multiple runs and statistically compare their performance
  - Compare medians with the Mann Whitney U-test

Methodology

Uniswap v2 invariants

Property	Description
P-01	Adding liquidity increases k
P-02	Adding liquidity increases the total supply of LP tokens
P-03	Adding liquidity increases reserves of both tokens
P-04	Adding liquidity increases the user's LP balance
P-05	Adding liquidity decreases the user's token balances
P-06	Adding liquidity does not decrease the `feeTo` LP balance
P-07	Adding liquidity for the first time should mint LP tokens equals to the square root of the product of the token amounts minus a minimum liquidity constant
P-08	Adding liquidity should not change anything if it fails
P-09	Adding liquidity should not fail if the provided amounts are within the valid range of `uint112`, would mint positive liquidity and are above the minimum initial liquidity check when minting for the first time
P-10	Removing liquidity decreases k
P-11	Removing liquidity decreases the total supply of LP tokens if fee is off

Property	Description
P-12	Removing liquidity decreases reserves of both tokens
P-13	Removing liquidity decreases the user's LP balance
P-14	Removing liquidity increases the user's token balances
P-15	Removing liquidity does not decrease the `feeTo` LP balance
P-16	Removing liquidity should not change anything if it fails
P-17	Removing liquidity should not fail if the returned amounts to the user are greater than zero
P-18	Swapping does not decrease k
P-19	Swapping increases the sender's tokenOut balance
P-20	Swapping decreases the sender's tokenIn balance by swapAmountIn
P-21	Swapping does not decrease the `feeTo` LP balance
P-22	Swapping should not fail if there's enough liquidity, if the output would be positive and if the input would not overflow the valid range of `uint112`

Methodology

Invariant tests

- Stateful handler-based invariant testing
- Single-instance state initialization
- Success/failure invariant verification

```
contract Tester {
    function addLiquidity(uint amount1, uint amount2) public {
        //PRECONDITIONS:
        amount1 = clampBetween(amount1, 1, type(uint256).max);
        amount2 = clampBetween(amount2, 1, type(uint256).max);

        _mintTokensOnce(amount1, amount2);

        _before();

        //ACTION:
        (bool success, ) = user.proxy(
            address(router),
            abi.encodeWithSelector(
                router.addLiquidity.selector,
                address(token1),
                address(token2),
                amount1,
                amount2,
                0,
                0,
                address(user),
                type(uint256).max
            )
        );

        _after();

        //POSTCONDITIONS
        if (success) {
            gt(
                vars.kAfter,
                vars.kBefore,
                "P-01 | Adding liquidity increases K"
            );
            gt(
                vars.lpTotalSupplyAfter,
                vars.lpTotalSupplyBefore,
                "P-02 | Adding liquidity increases the total supply of LP tokens"
            );
            // ...
```

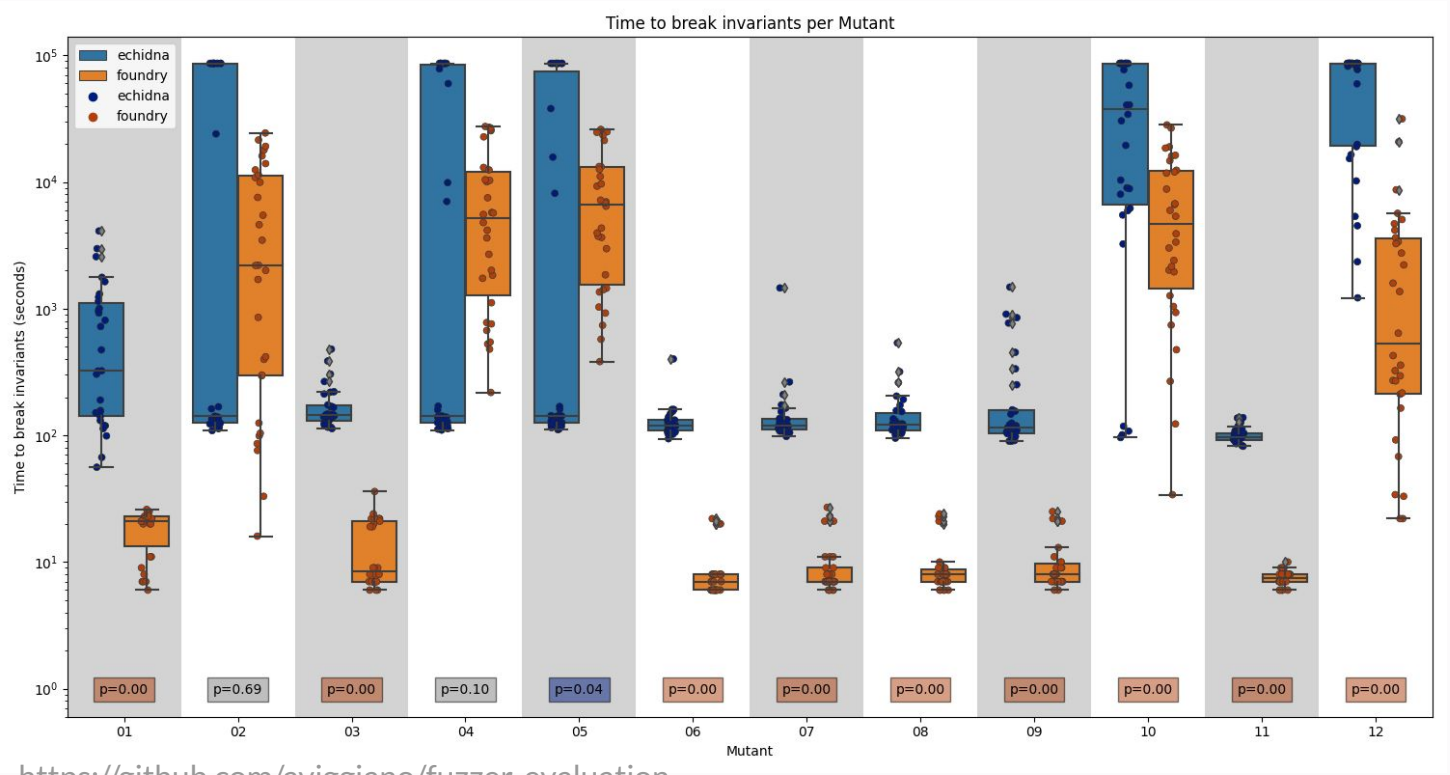
Methodology

Sample mutants

```
diff --git a/protocols/uniswap-v2/src/UniswapV2Pair.sol
b/protocols/uniswap-v2/src/UniswapV2Pair.sol
index 83cfddc..65f1ed9 100644
--- a/protocols/uniswap-v2/src/UniswapV2Pair.sol
+++ b/protocols/uniswap-v2/src/UniswapV2Pair.sol
@@ -155,7 +155,7 @@ import "./interfaces/IUniswapV2Callee.sol";
             liquidity =
Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
             _mint(address(0), MINIMUM_LIQUIDITY); // permanently
lock the first MINIMUM_LIQUIDITY tokens
         } else {
-            liquidity = Math.min(
+            liquidity = Math.max(
                 amount0.mul(_totalSupply) / _reserve0,
                 amount1.mul(_totalSupply) / _reserve1
             );
```

```
diff --git a/protocols/uniswap-v2/src/libraries/Math.sol
b/protocols/uniswap-v2/src/libraries/Math.sol
index b86df89..80a2e49 100644
--- a/protocols/uniswap-v2/src/libraries/Math.sol
+++ b/protocols/uniswap-v2/src/libraries/Math.sol
@@ -11,7 +11,8 @@ library Math {
     function sqrt(uint y) internal pure returns (uint z) {
         if (y > 3) {
             z = y;
-            uint x = y / 2 + 1;
+            /// BinaryOpMutation(`+` |==> `*`) of: `uint x = y / 2 + 1;`
+            uint x = y / 2*1;
             while (x < z) {
                 z = x;
                 x = (y / x + x) / 2;
```

Results



https://github.com/aviggiano/fuzzer-evaluation

Conclusion

Key takeaways

- Performance varies substantially from run to run (up to 1000x)

  → **Perform many runs and for long hours**

- Foundry breaks invariants faster in 9 tests, Echidna in 1, and the difference is not statistically significant in the other 2
- Some Foundry runs finished after only ~8h with false positives, while some Echidna runs timed out after ~24h

  → **Use multiple fuzzers**

Future research

Next steps

- Expand the number of tested protocols to derive generality
- Expand the number of fuzzers evaluated
- Evaluate fuzzers on different test environments
- Evaluate against "in the wild" issues instead of synthetic bugs
- Analyse performance over time
- Derive a solid, independently defined benchmark suite

References

Sources list

1. Klees et al.: Evaluating Fuzz Testing
2. Nazgul: Fuzzy DeFi: Pre-built security properties for commonly forked DeFi protocols
3. horsefacts: Invariant Testing WETH With Foundry
4. Trail of Bits: Fuzzing Workshop
5. Trail of Bits: Slither, the Solidity source analyzer
6. Certora: Gambit: Mutant Generation for Solidity