TRAIL OF BITS

Advanced DeFi Invariants

Upcoming workshops

Beginner

- Part 1: The Basics
- Part 2: Breaking ABDKMath (Week of Nov 21, 2022)

Intermediate

- Part 3: Breaking Uniswap I (Week of Nov 28, 2022)
- Part 4: Breaking Uniswap II (Week of Dec 5, 2022)

Advanced

- **Part 5: Advanced DeFi Invariants I (Week of Dec 12, 2022)**
- Part 6: Advanced DeFi Invariants II (Week of Dec 19, 2022)

Who am I?

Nat Chin, Security Engineer II

Who You Should Follow

- Troy Sargent (@0xalpharush)
- Josselin Feist (@montyly)
- Anish Naik (@anishrnaik)
- Justin Jacob (@technovision99)

Who are we?

Trail of Bits (@trailofbits)

- We help developers to build safer software
- R&D focused: we use the latest program analysis techniques
- Slither, Echidna, Tealer, Amarna, solc-select, ..

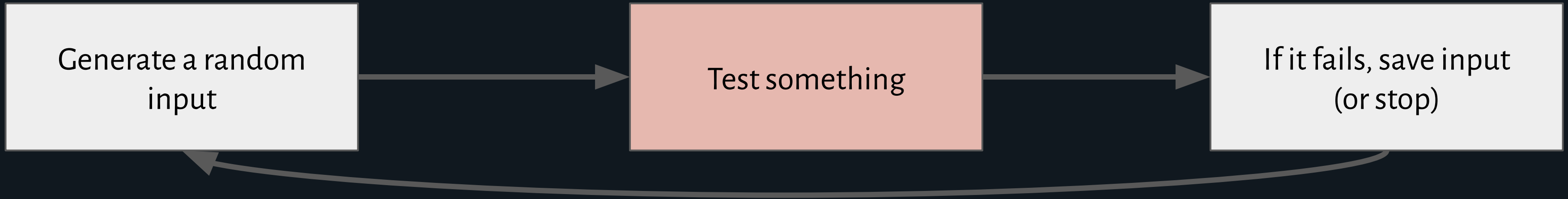
Agenda

- Recap: Fuzzing
- Introduction to Primitive
- Architecture
- How to find invariants
- Code Walkthrough
- Invariant Search
- Implementing basic invariants

So…how do I start fuzzing?

1. Identify your invariants / **system properties in English**
2. Convert your properties to code
3. **Run Echidna**
4. FIND BUGS

And… what is fuzzing?



Echidna vs Other Fuzzers

- Echidna is more mature
- Allows testing of high gas assumptions
- Works with any compilation framework
- Supports various API's for testing
- Supports hevm/dapptool cheatcodes

Tips on Identifying Invariants

- **Start with the smallest component first**
- **Analyze all preconditions and postconditions**
- **Determine safe bounds of inputs**
- **Identify inversely-related functions**
- **Focus on the happy and unhappy paths**

Useful Optimizations

- **Tests should have precondition, action, postcondition**
  - Pre-conditions: Scope the input space
  - Action: What we are testing
  - Post-conditions: The “truths” after the action
- **Coverage is your friend!**
  - Echidna saves coverage in corpus

Corpus **is** your friend! Especially Echidna 2.0.4

```
    |    //  -------------------- Margin.sol ----------------------
 *r |    function depositIncreasesBalance(uint256 risky, uint256 stable) public {
 *r |            uint256 pre_deposit_bal_risky = margin.balanceRisky;
 *r |            uint256 pre_deposit_bal_stable = margin.balanceStable;
 *r |            Margin.deposit(margin, risky, stable);
    |
 *  |            assert(margin.balanceRisky - pre_deposit_bal_risky == risky);
 *  |            assert(margin.balanceStable - pre_deposit_bal_stable == stable);
    |    }
    |    mapping (address => Margin.Data) margins;
 r  |    function withdrawDecreasesBalance(uint256 risky, uint256 stable) public {
 r  |            margins[address(this)] = margin;
 r  |            uint256 pre_deposit_bal_risky = margins[address(this)].balanceRisky;
 r  |            uint256 pre_deposit_bal_stable = margins[address(this)].balanceStable;
 r  |            Margin.withdraw(margins, risky, stable);
    |
    |            assert(pre_deposit_bal_risky - margins[address(this)].balanceRisky  == risky);
    |            assert(pre_deposit_bal_stable - margins[address(this)].balanceStable == stable);
    |    }
    |
```

Primitive

Primitive: What is it?

- **Replicating Market Maker**
- **Implements Black-Scholes interest options**

words, words, words

Primitive vs Uniswap

Primitive

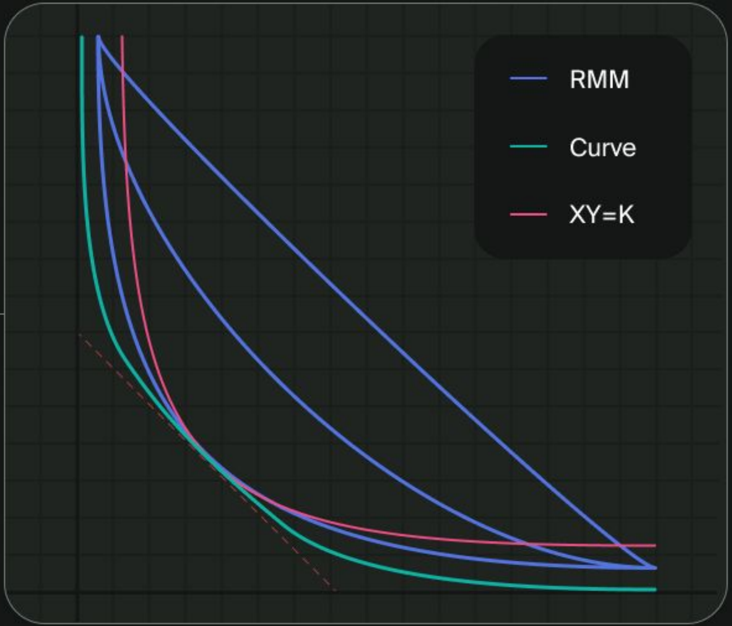
- Price changes on swap and over time
- At expiry, pool consists of an underlying token

Uniswap

- Price changes on swap
- Pools don't have a concept of time

Constant Function Market Makers

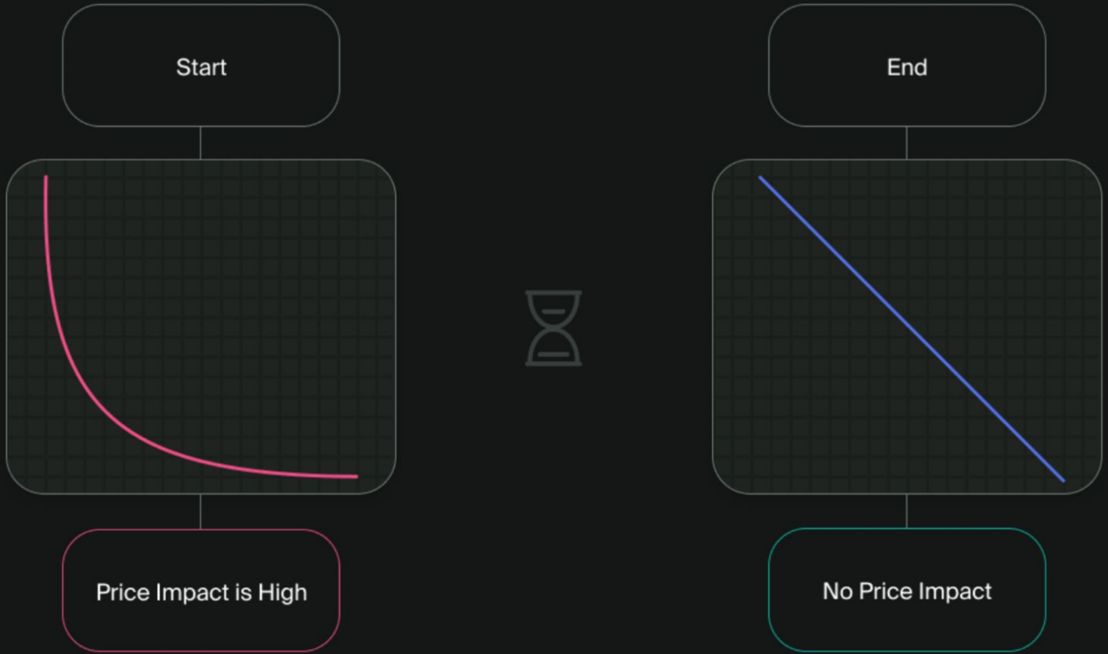
Tradng Functions



Features

- Allows creation of pools against 2 tokens
- Relies on spot price – no oracles
- Price curve continues to changes until maturity
- Price will converge to strike price at maturity

Liquidity Compression - Concentration Over Time



Black-Scholes Options

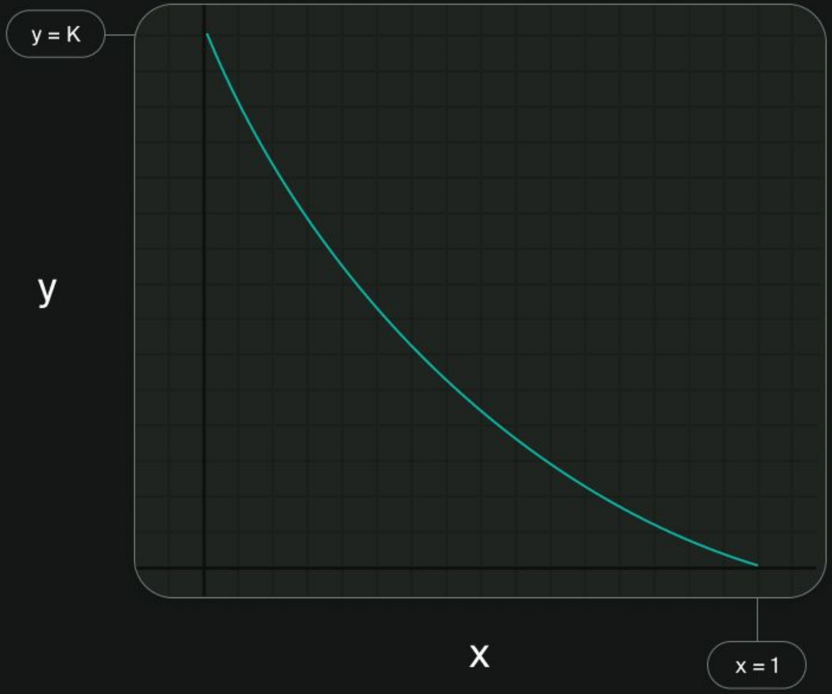
- **This is how the price changes over time**
- **European Option – right to exercise at expiration**
- **Pricing model depends on:**
    - Strike price (K)
    - Implied volatility (over a period of time)
    - Time to expiry (t)
    - Spot price of underlying (S)

Primitive RMM Curve

K	Strike Price	x	Underlying Asset Reserve
σ	Implied Volatility	y	Quote Asset Reserve
τ	Time until Expiry	Φ	CDF
k	Invariant	Φ^{-1}	Inverse CDF

$$y - K\Phi(\Phi^{-1}(1 - x) - \sigma\sqrt{\tau}) = k$$

Trading Function



At maturity

- Assets cannot be bought/sold
- Sells the second asset
- Pool prices at the strike price

System Architecture

Architecture

- **Core system**
  - PrimitiveEngine – controls pools
  - PrimitiveFactory – creates engines
- **Manager: Periphery**

Primitive Engine

- Create a new pool (curve)
- Allocate and remove liquidity from a pool
- Swap between tokens
- Deposit and withdraw from system

Creating New Pool

- Tokens – underlying and quote token
- Strike price – asset's worth at maturity
- Implied Volatility – how much price changes for swaps
- Maturity – expiration
- Gamma – trading fee percentage

Reference: Primitive Whitepaper

What can you do with a curve?

- **Allocate liquidity – supply tokens to curve**
- **Remove liquidity – remove tokens to curve**
- **Create liquidity position – deposit money to Engine**
- **Swap between tokens**

System Architecture

Engine

```
function deposit() external { }
function withdraw() external { }
function allocate() external { }
function remove() external { }
function swap() external { }
```

Pool

```
function allocate() external { }
function remove() external { }
function swap() external { }
```

Version 0.8.0+

Splitting up Deposit and Allocate

- **Allows users to deposit into the system first**

Libraries

- ABDK Math
- Primitive Math*
- Margin balances
- Reserve balances
- SafeCast
- Transfers
- Unit conversions

Let’s look at code!

Margin Library: Balance Tracking

```
struct Data {
    uint128 balanceRisky; // Balance of the risky token, aka underlying asset
    uint128 balanceStable; // Balance of the stable token, aka "quote" asset
}
```

Margin Library: Deposit into Margin

```
/// @notice             Adds to risky and stable token balances
/// @param  margin      Margin data of an account in storage to manipulate
/// @param  delRisky    Amount of risky tokens to add to margin
/// @param  delStable   Amount of stable tokens to add to margin
function deposit(
    Data storage margin,
    uint256 delRisky,
    uint256 delStable
) internal {
    if (delRisky != 0) margin.balanceRisky += delRisky.toUint128();
    if (delStable != 0) margin.balanceStable += delStable.toUint128();
}
```

Margin Library: Withdraw from Margin

```solidity
/// @notice          Removes risky and stable token balance from `msg.sender``'s internal margin account
/// @param  margins     Margin data mapping, uses `msg.sender``'s margin account
/// @param  delRisky    Amount of risky tokens to subtract from margin
/// @param  delStable   Amount of stable tokens to subtract from margin
/// @return margin      Data storage of a margin account
function withdraw(
    mapping(address => Data) storage margins,
    uint256 delRisky,
    uint256 delStable
) internal returns (Data storage margin) {
    margin = margins[msg.sender];
    if (delRisky != 0) margin.balanceRisky -= delRisky.toUint128();
    if (delStable != 0) margin.balanceStable -= delStable.toUint128();
}
```

Margins Library: First Invariants

- Deposit should result in balance increase
- Withdraw should result in balance decrease
- Deposit then withdraw should be identical

Lessons Learned

- **Use the corpus!**
- **Start with the libraries**
- **Don't be afraid to mock**
- **Understand the system *first***

Let’s give it a try! :)

So on your own time……

- **Brainstorm invariants**