Compound

Secure By Design

August 2022

🗂️ Overview

1. SPEC
2. CODE
3. TEST
4. SHIP

📋 SPEC

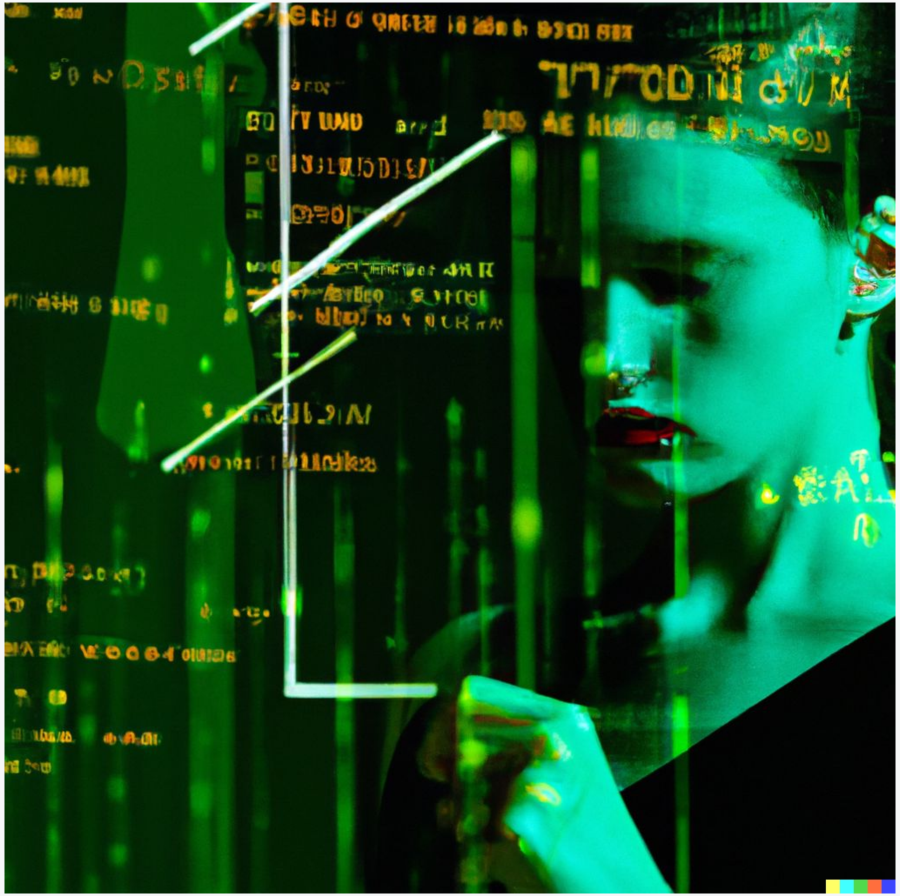
📝 Start With A Spec

- **what are your goals? uses cases? non-use cases?**
- **what invariants *must* hold?**
  - e.g. for Compound, safety first
    - **S + R = C + B (fundamental accounting equation)**
- **what attributes are desirable? what guiding principles?**
  - e.g. for Compound III, when in doubt:
    - **capital efficiency, gas efficiency**
    - **deployable on other chains**
    - **simplicity, especially for users**

🤷🏽‍♀️ Decisions, Decisions

- **enormous design space for a protocol**
  - goal is to explore space collectively, as exhaustively as possible
  - tons of decisions, only partial information
- **do what it takes to move discourse forward *now***
  - model / simulate unknowns to gain insight
  - e.g. Compound III liquidation analysis
- **the more you do it, the better you get**
  - like anything else
  - incorporate learnings/regrets from previous experience
- **highest level design doc**
  - cheapest to iterate + easiest to collaborate
  - code and tests should fall directly out, if done well

🖥️ CODE



Compound

🧑🏾‍💻 Compile Spec To Code

- **mostly just a matter of translating the spec**
  - e.g. into Solidity
- **hard decisions should already have been made**
  - if not, go back and fix the spec

Compound

💯 Unit Test Contracts Before Merging

- **contract code should have ~100% unit test coverage**
  - keep the main development branch covered
  - only merge PRs which cover all contract changes
- **branches in code should be covered by branches in spec**
  - at least 1 test for each branch
  - writing these together 1:1 is most straightforward

✅ TEST

⚖️ Think In Terms Of Invariants

- **high level properties of the system, such as:**
  - fundamental accounting equation holds
  - exchange rate should always increase
  - assets supplied should be able to be withdrawn
  - an account which goes underwater should be liquidatable
- ***avoid* more imperative steps, such as:**
  - initiate a supply of X for user A, then withdraw X
  - create a borrower, move the prices against them, then liquidate
- **start with the strongest invariants you'd like to claim**
  - then figure out strategies for getting at them
  - uncover any implicit assumptions or pre-conditions and make explicit

🧾 Prove What You Can

- **write invariants in something like CVL**
    - might prove exhaustively over entire state space
- **chances are you’ll need some approximations**
    - use state constraints, lemmas, and harnesses to model as closely as possible
- **exercise is beneficial in itself**
    - the act alone causes you to think differently about the problem

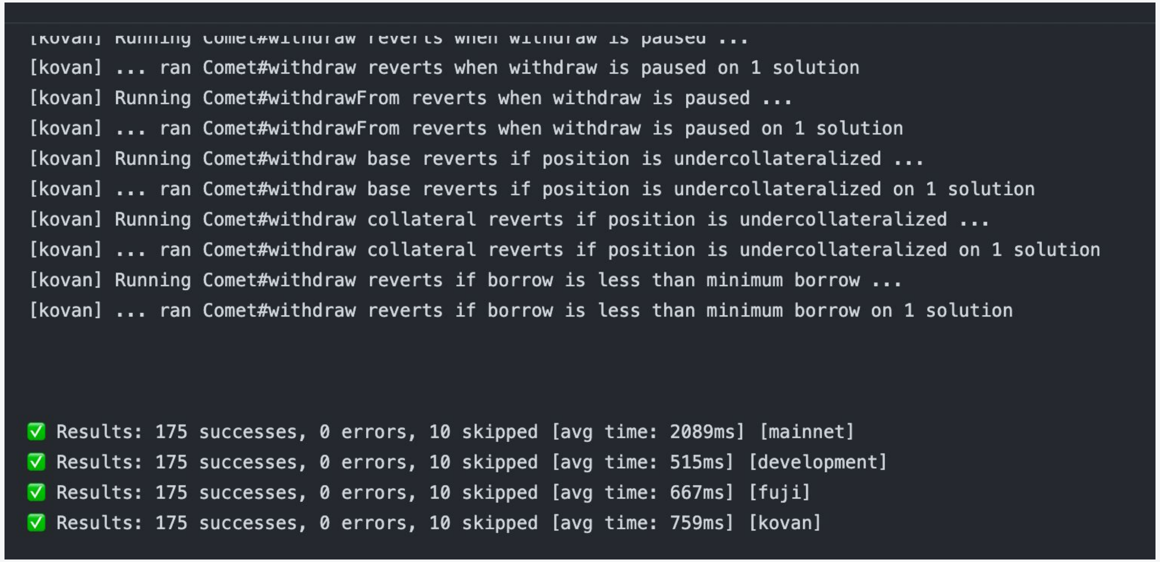
🧪 Scenarios For Everything Else

- **Compound III built using a brand new ‘scenario’ framework**
  - define properties which are sequences of actions by actors in a world & context
  - each property can have explicit assumptions declared
    - **e.g. assume utilization is 50%, or Alice has token balance ≥ X**
- **framework produces sets of worlds in which assumptions hold**
  - worlds are derived from real chain state, either deployed locally or forked
  - properties are checked against each of these worlds
  - assumptions can also embed *fuzzed* terms
- **actual proposals to change protocol written as migration scripts**
  - automatically run hypothetical changes against entire scenario suite
  - same exact code used to propose changes for real

🧪 Scenarios…

```
scenario(
  'Comet#supply > base asset',
  {
    tokenBalances: {
      albert: { $base: 100 }, // in units of asset, not wei
    },
  },
  async ({ comet, actors }, context) => {
    const { albert } = actors;
    const baseAssetAddress = await comet.baseToken();
    const baseAsset = context.getAssetByAddress(baseAssetAddress);
    const scale = (await comet.baseScale()).toBigInt();

    expect(await baseAsset.balanceOf(albert.address)).to.be.equal(100n * scale);

    // Albert supplies 100 units of base to Comet
    await baseAsset.approve(albert, comet.address);
    const txn = await albert.supplyAsset({ asset: baseAsset.address, amount: 100n * scale })

    const baseIndexScale = (await comet.baseIndexScale()).toBigInt();
    const baseSupplyIndex = (await comet.totalsBasic()).baseSupplyIndex.toBigInt();
    const baseSupplied = getExpectedBaseBalance(100n * scale, baseIndexScale, baseSupplyIndex);

    expect(await comet.balanceOf(albert.address)).to.be.equal(baseSupplied);

    return txn; // return txn to measure gas
  }
);
```

🧪 Scenarios…



```
[kovan] Running Comet#withdraw reverts when withdraw is paused ...
[kovan] ... ran Comet#withdraw reverts when withdraw is paused on 1 solution
[kovan] Running Comet#withdrawFrom reverts when withdraw is paused ...
[kovan] ... ran Comet#withdrawFrom reverts when withdraw is paused on 1 solution
[kovan] Running Comet#withdraw base reverts if position is undercollateralized ...
[kovan] ... ran Comet#withdraw base reverts if position is undercollateralized on 1 solution
[kovan] Running Comet#withdraw collateral reverts if position is undercollateralized ...
[kovan] ... ran Comet#withdraw collateral reverts if position is undercollateralized on 1 solution
[kovan] Running Comet#withdraw reverts if borrow is less than minimum borrow ...
[kovan] ... ran Comet#withdraw reverts if borrow is less than minimum borrow on 1 solution



✅ Results: 175 successes, 0 errors, 10 skipped [avg time: 2089ms] [mainnet]
✅ Results: 175 successes, 0 errors, 10 skipped [avg time: 515ms] [development]
✅ Results: 175 successes, 0 errors, 10 skipped [avg time: 667ms] [fuji]
✅ Results: 175 successes, 0 errors, 10 skipped [avg time: 759ms] [kovan]
```

👀 Get More Eyes On It

- **triple and quadruple check your work**
  - hire auditors
  - share with community
- **share your ideas and the spec**
  - the more you share, the better the responses will be
  - the better people understand the intention, the more useful the feedback
- **don’t take feedback personally**
  - goal is to deliver the best product
  - the sooner you discover issues the better
- **be grateful when people take the time to help you**
  - whether they find issues or not
  - many thanks to ChainSecurity, Certora, OpenZeppelin, Gauntlet for Compound III

🚢 SHIP

Compound

🚀 Deploy The Contracts

- **more holistic approach to Compound III**
    - deployments and migrations integrate closely with scenarios
    - practices designed around emergent governance processes
- **create a deployment for a network**
    - deploy script describes how each of the ‘root’ contracts gets created
    - set of relations to all other contracts which are deployed
        - **these can be crawled and discovered from the roots**
    - automatically simulated hypothetically and tested against scenario suite

🐒 Plan For Evolution

- **create a migration script in the deployment directory**
    - primary purpose is to make an actual governance proposal
    - automatically simulated hypothetically and tested against scenario suite
- **straightforward story for contributors**
    - open a PR with script and any new tests needed, and share with community
        - **the PR should pass CI**
        - **artifacts can be seen/generated directly from CI**
        - **easy for governance to review**
    - run the migration script for real and make the proposal
        - **can also be done through CI**
    - once the proposal is executed, merge the PR

🙇‍♂️ Thanks! Questions?

- @jmflatow
- Compound Discord
- We're Hiring →

Compound