TrueFi Phase 3 Smart Contracts Audit

Intro

The TrustToken team is building an uncollateralized lending platform. The initial launch was in November 2020, I audited the February 2021 Phase 2 update, and they have now asked me to audit the next update in preparation for its Phase 3 launch in May 2021. The TrueFi v2 spec can be found here. I audited the code located in the open-source github repository trusttoken/smart-contracts/, with scope limited to these contracts:

contracts/governance
- GovernorAlpha.sol
- StkTruToken.sol
- Timelock.sol
- VoteToken.sol

contracts/truefi2
- Liquidator2.sol
- LoanFactory2.sol
- LoanToken2.sol
- PoolFactory.sol
- TrueFiPool2.sol
- TrueLender2.sol
- strategies/CurveYearnStrategy.sol

The version of the code I reviewed was published with the commit a96a83e6fd8511f9aad748a4a5194685a27d06f3.

I have classified issues using the OWASP risk rating model, which estimates an issue's overall severity by combining my estimates for both the likelihood of occurrence and the impact of consequences.

Summary

- No Critical or High severity issues were found.
- One Medium severity issue was found: in the event that a loan defaults and some portion ends up written off, the pool will start overvaluing itself.
- One Low severity issue was found: if a small enough loan were to be approved, there would be no way of reclaiming the funds afterward (without upgrading the contracts).
- Several minor Notes for improvement, most notably about addressing an edge case where NO votes can cause an otherwise failing loan-rating vote to pass.

Issues found

Pool potentially overvalued after default (MEDIUM)

Can't reclaim small loans (LOW)

Notes

- NO votes can help a loan succeed
- Pool is harder to pause
- Typo in error message
- Duplicate tests
- Redundant argument
- Unexpected keystone
- Lack of documentation for reclaim()
- Staking doesn't have to restart cooldown

Pool potentially overvalued after default

Severity: MEDIUM (Impact: MEDIUM, Likelihood: MEDIUM)

Problem
In TrueFiPool2.sol, the pool computes its total value assuming that all loan tokens are worth their full nominal value.

Consequences
If a loan defaults and arbitration plus SAFU (insurance) fail to recover/cover the missing funds, some or all of the loan must be written off so the loan tokens will not actually be worth their full nominal value. This would cause the pool to overvalue its assets, thus (at least to some degree) overcharging future lenders who join the pool and overpaying lenders who leave.

Recommendations
- Have a robust SAFU so this doesn't happen often.
- Incorporate a way to remove written-off loan tokens from the pool or mark them as valueless, so the pool has a way to stop overvaluing itself after such an event.

Can’t reclaim small loans

Severity: LOW (Impact: MEDIUM, Likelihood: LOW)

Problem

In TrueLender2.sol, the reclaim function will reject reclaiming the funds from any loan that generates less than $100 in fees.

Consequences

If a loan legitimately generates less than $100 in fees ($1000 in interest, at the current 10% fee rate), it will be impossible to reclaim the funds (without upgrading the contract first). This would not have been a problem with any loan from TrueFi so far - the smallest amount of interest to date was ~$9.6k, for a 9.6x margin of safety. However this margin could easily disappear entirely if TrueFi expands into smaller loans, especially if interest rates drop or they decide to use less of the interest as fees.

Recommendations

Either remove the **minFee** check entirely, reduce it by a factor of at least 10, or compute a conservative minimum dynamically based on the loan parameters. (Also, fix the misleading comment - it says “Assume...interest is not below 1000USD”, but this is implicitly assuming that the fee will always be 10% of interest earned.)

Notes

Issues with a very low impact

NO votes can help a loan succeed

In TrueLender2.sol, one requirement for a loan being approved is that the sum of YES and NO votes is over some threshold. This means that a NO vote can actually cause a vote to succeed which would otherwise have failed, if that vote is enough to push the total over the threshold without being enough to tip the ratio towards NO winning. To remove this counterintuitive behavior (which is also inconsistent with the way voting is implemented in GovernorAlpha.sol), only YES votes should be counted towards the threshold.

Pool is harder to pause

In the old TrueFiPool.sol, the pool implements **IPauseableContract**, a standardized way of pausing TrueFi contracts. The new TrueFiPool2.sol implements the same functionality, but since it renamed the functions and variables, it no longer actually meets the **IPauseableContract** standard. The pool's long-term owner is supposed to be Timelock.sol; once that happens there would be no way to quickly invoke the pool's pause functionality in an emergency since Timelock's **setPauseStatus** requires the **IPauseableContract** interface. (Timelock could still pause the pool using the more extreme **emergencyPause**.)

Typo in error message

In Timelock.sol, an error message beginning "Timelock::setDelay" in **initialize** should instead say "Timelock::constructor" (or "Timelock::initialize").

Duplicate tests

In Timelock.test.ts, the tests on lines 67 and 73 are duplicates.

Redundant argument

In LoanToken2.sol, the event **Liquidated(status)** can only ever be emitted with **Status.Liquidated**. The existence of the argument misleadingly implies that we might see it emitted with various statuses; as it is the simpler nullary **event Liquidated()** would suffice.

Unexpected keystone

In LoanToken2.sol, **fund** has the modifier **onlyLender**, meaning that only the intended lender can fund the loan. This may at first appear not very important - why would it be so bad if someone else funded the loan? And indeed I believe I heard that in a future version, a LoanToken might be fundable by anyone, a belief which is also supported by the currently-inaccurate comment on **fund** which says that it *sets the lender* as well as the status and start time. However right now if **onlyLender** were removed, an attacker could create a loan, fund it, withdraw the funds, wait for it to default, and then liquidate it from Liquidator2.sol, slashing staked TRU. Thus **onlyLender** is actually critical for the platform's security. Since I think this is not at all obvious, I would suggest adding a comment on **fund** documenting this, so that future devs do not remove **onlyLender** without an appropriate backup plan.

Lack of documentation for reclaim()

TrueLender2.sol attempts to decentralize the reclaiming of successful loans by making the function **reclaim** callable by anyone. However its **data** argument is undocumented, and without providing a correct value the function will revert. A comment should be added so users other than the TrustToken team can know how to call this function. (**sellLiquidationToken** in TrueFiPool2.sol has the same issue.)

Staking doesn't have to restart cooldown

In StkTruToken.sol, staking more TRU resets the cooldown timer for when you are allowed to unstake. While this was the case in the previous version as well, now that the new **receivedDuringCooldown** mechanism exists for transfers anyway, it could also be used to track newly staked TRU so that staking would not reset cooldown but TRU staked during the cooldown would not be eligible for unstaking. I think this method would be more user-friendly, primarily by being less likely to result in a bad surprise for users who do not realize that staking will reset their cooldown.

Benjamin Cosman