ETHWORKS

truefi

TrueFi 2.0 Smart Contracts Internal Audit

Intro

The TrueFi team is working on upgrading the existing under-collateralized lending system powered by the TRU token incentives.

The code which is audited is located in the open-source repository **https://github.com/trusttoken/smart-contracts** and the audit scope is limited to the following contracts:

- TrueFi/
  - Liquidator.sol
  - LoanFactory.sol
  - LoanToken.sol
  - TrueFiPool.sol
  - TrueRatingAgencyV2.sol
  - TruePriceUniswapOracle.sol
- Governance/*
- TrustToken/TrustToken.sol

The upgrades include the following features:

- Governance ownership system similar to the one used by Compound
- Add checkpoint-based voting power to the TRU contract
- Introduce **StkTruToken**: the TRU staking contract with distribution system similar to the existing **TrueFarms**
- Introduce new loan rating mechanism: loans can only be rated using the **StkTruToken** using the **VoteToken** checkpoints
- Introduce liquidation mechanism for defaulted loans that slashes part of **StkTruToken** stake and sends compensation to the **TrueFiPool**
- Change loan fee beneficiary from the **TrueFiPool** to the **StkTruToken**

The version of the code that is being reviewed was published with the commit ***b8189766c4bf22660754c13dd7c178766b333fd0***.

All issues will be classified using the OWASP risk rating model.

IMPACT	LIKELIHOOD: LOW	LIKELIHOOD: MEDIUM	LIKELIHOOD: HIGH
HIGH	MEDIUM	HIGH	CRITICAL
MEDIUM	LOW	MEDIUM	HIGH
LOW	NOTE	LOW	MEDIUM

Image 1: OWASP risk model

Summary

The engineering team has demonstrated solid engineering skills in an effort to decentralize existing TrueFi protocol.

The design is clean and responsibilities are well divided between smart contracts. The code is readable, well covered by tests and documentation is sufficient to understand the code.

The Governance part of the smart contracts largely reuses contracts created by Compound, so all issues found by Open Zeppelin's audit[1] of Compound governance may apply to TrueFi code as well.

There have been **no critical** and **2 high severity issues** found. The main vulnerabilities include **DDoS attacks** and **abusing flash-loans**.

[1] https://blog.openzeppelin.com/compound-alpha-governance-system-audit/

1. Uniswap oracle price manipulation [HIGH]

Impact: high
Likelihood: medium

The TRU price oracle uses current Uniswap pair balance which is prone to price manipulation.

Attack:

1. Loan defaults
2. Get flashloan
3. Drop price of TRU on Uniswap
4. Call Liquidate
5. Slash 10% of the stake

Recommendation:
Use ChainLink oracle or use Uniswap oracle with time-weighted average prices.

2. payFee() DDoS attack [HIGH]

Impact: high
Likelihood: medium

payFee method of *StkTruToken* can be called by anyone. Somebody could call it multiple times with *amount = 0* and large *endTimes*. This would fill the tail of *sortedScheduledRewardIndices* with garbage that will have to be moved on every *payFee* execution which may result in out of gas errors.
This exploit is expensive to execute and would not make any profit for the attacker.

Recommendation:
Either add a minimum fee amount to make this type of attack more expensive or restrict visibility of the method to *TrueLender* only.

3. StkTRU holders will not have voting power before calling delegate() [MEDIUM]

Impact: low
Likelihood: high

Users will not get any voting power when they perform staking for the first time. This might be confusing for the stakers: they had votes by holding TRU, but will lose them on staking until they call *delegate()*.

Recommendation:
In *_mint()* method of VoteToken if the delegate of the account is 0x000...000, set delegate to themselves. Or make a getter for delegates that will return self if mapping is set to 0x00000...

4. Liquidation does not increase pool value [MEDIUM]

Impact: medium
Likelihood: medium

TRU tokens transferred into the pool via liquidation are not included into the pool value but are returned on exit. There is also no possibility to swap TRU inside the pool for anything else.
The implication is that the pool value drops in case of defaulted loan but is not compensated by the liquidation which contradicts with the idea of liquidation introduction.
Another implication is that with TRU on the Pool's balance, it might be profitable to stake and unstake straight away taking the TRU. This might open ways for flash loan attacks.

Recommendation:
Convert TRU balance in the pool to TUSD (see issue #1 with oracles), or add TRU to the pool value. Add a way to exchange TRU for TUSD.

5. Staking off cooldown [MEDIUM]

Impact: low
Likelihood: high

In StkTruToken, line 168:

```
if (cooldowns[msg.sender] != 0 && cooldowns[msg.sender].add(cooldownTime) >
block.timestamp) {
```

When cooldown has passed, users can stake and unstake without any restrictions.
This even allows to create several accounts with different unstake windows and transfer staked tokens freely between them avoiding cooldown.
It is not obvious how this could harm the protocol. Possible exploits could include arbitrage abuse or flash loan attacks.

Recommendation:
Change *if* condition to

```
if (cooldowns[msg.sender] != 0 &&
cooldowns[msg.sender].add(cooldownTime).add(unstakePeriodDuration) >
block.timestamp) {
```

6. Some methods should no longer be onlyOwner [MEDIUM]

Impact: low
Likelihood: high

Some methods, for example, TrueFiPool's CRV and TRU management methods, or whitelisting methods are not designed to be run by governance – onlyOwner modifier was seemingly added to prevent users from running it. These methods do not require this level of protection and delays caused by the voting process may make some methods impossible to call.

Recommendation:
Add roles for calling the methods that don't change protocol parameters.

7. No argument validation in setFetchMaxShare [LOW]

Impact: medium
Likelihood: low

There is no check if the new max share is not above 100%. Setting *fetchMaxShare* to over 10000 will make liquidation impossible.

Recommendation:
Add *require(newShare <= 10000)* to *setFetchMaxShare*

8. Unsafe casting to uint96 in VoteToken [LOW]

Impact: medium
Likelihood: low

When fetching delegator's balance, value is cast to uint96 in an unsafe way. Issue occurs in *_delegate* method.

Recommendation:
Use *safe96()* method for casting.

9. No validation in setStakeToken [LOW]

Impact: medium
Likelihood: low

Stake token is not expected to ever change, so the setter is expected to be called only once.

Recommendation:
Add *require(_stakeToken == address(0))* to *TrueFiPool.setStakeToken* method

10. Lack of consistency in code [NOTE]

- All *private* and *internal* methods should either have underscores before their names or not. Example in *LoanToken.sol*:
  - *_transfer(address,address,uint256) internal*
  - *interest(uint256) internal*
- Different cases for constant variables between files. Example:
  - LoanToken.sol: *constant lastMinutePaybackDuration*
  - TrueRatingAgencyV2.sol: *constant TOKEN_PRECISION_DIFFERENCE*
- *Claimable* and *Ownable* used interchangeably

11. Liquidation affects voting power in the past [NOTE]

If some proposal is introduced on block A when user has X amount of votes in StkTRU and stake slash happens in block B > A, the user will have less votes on that proposal. But if the user has managed to vote for the proposal before liquidation, the voting power does not change.

12. Misleading comment in VoteToken & Timelock [NOTE]

Comment in line 2 of VoteToken to should reference to this file
https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/Comp.sol
And in TImelock to
https://github.com/compound-finance/compound-protocol/blob/master/contracts/Timelock.sol

13. Incorrect commit version in comment of GovernorAlpha, VoteToken & Timelock [NOTE]

Comment in line 2 of GovernorAlpha, VoteToken and Timelock should refer to the specific commit version of the file instead of the master branch.